HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

IFS INSTITUTE FOR SOFTWARE

iTIXI – Dispositionssoftware für TIXI-Betriebe

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2014

Autor(en):	Carmelo Schumacher, Reto Schelbert
Betreuer:	Prof. Stefan Keller
Projektpartner:	TIXI Zug
Experte:	Claude Eisenhut
Gegenleser:	Prof. Dr. Josef M. Joller

1 Aufgabenstellung

HSR UNIVERSITY OF APPLIED SCIENCES RAPPERSWIL COMPUTER SCIENCE

Bachelorarbeit FS 2014

1

iTIXI – Disposoftware für TIXI-Betriebe (CH)

Bachelorarbeit Frühlingsemester 2014
Studiengang Informatik

Aufgabenstellung

Der gemeinnützige Verein TIXI ZUG, bietet einen Fahrdienst für Menschen mit einer Behinderung an. TIXI ZUG unterstützt behinderte, kranke und betagte Menschen, welche die öffentlichen Verkehrsmittel nicht benützen können, oder in ihrer Mobilität eingeschränkt sind.

Durch den Verein TIXI ZUG wird es möglich, dass mit 18 Fahrzeugen und über 250 Freiwilligen Fahrern, diese Menschen soziale Kontakte pflegen, an Kursen und Ausflügen teilnehmen, den Arzt- oder Therapiebesuch einhalten, oder sogar einen Coiffeur-Termin wahrnehmen können.

TIXI ZUG beabsichtigt für die Disposition und den Fahr-Betrieb ihre bestehende Software abzulösen und durch eine neuzeitliche und nachhaltige Lösung zu ersetzen. Diese Ablösung soll durch eine Open Source Lösung (LAMP, Symfony, HTML5), die in einem Rechenzentrum gehosted wird, realisiert werden.

TIXI ZUG vergibt den Entwicklungsauftrag für das Open Source Projekt an den HSR Studenten:

1. Herr Carmelo Schumacher
2. Herr Reto Schelbert

im Rahmen einer Bachelorarbeit. Die Projekt-Architektur, -Anforderungen und Storyboards sind publiziert in https://sourceforge.net/projects/itixi/.

Aufgaben

Software Entwicklung:

ID	Name	Paket	Realisierung
F-001	Applikationsleiste	1	HSR
F-010	Einstellungen Seite	1	HSR
F-020	Stammdaten verwalten	2	HSR
F-030	Fahrgast Stammdaten verwalten	2,3	HSR
F-040	Fahrer Stammdaten verwalten	2,3	HSR
F-050	Fahrzeug Stammdaten verwalten	2,3	HSR
F-060	Orte von Interesse (OVI) Stammdaten verwalten	2,3	HSR
F-070	Disposition Monatsplanung	3	HSR
F-080	Fahrer und Fahrzeug zuordnen (Tagesplanung, FFPlan)	3	HSR
F-090	Einzel-Auftrag (Tagesplanung, Disposition)	3	HSR
F-100	Disposition (Tagesplanung)	4,5	HSR
F-110	Fahrwegoptimierung	5	HSR
F-120	*Bereitstellen*	*6*	*TIXI ZUG*
F-130	*Exportieren (Schnittstelle FIBU: BUSPRO)*	*6*	*TIXI ZUG*
F-140	*Unterhalt*	*2, 6*	*TIXI ZUG*
F-150	*Anruf-Erkennung (TAPI, Swisscom Business Connect)*	*7*	*späterer Release*
F-160	*System Eigenschaften*	*2*	*TIXI ZUG*

- Es sind die Anforderungen F-001 bis F-110 zu realisieren, ausser die in F-070 bis F-100 enthaltenen Funktionen Sammelauftrag (Frontend und Backend) und Tagesplanung (Frontend).
- Die Anforderungen F-001 bis F-070 wurden ursprünglich durch TIXI ZUG erbracht und sind dann während der Arbeit vollständig ersetzt worden.
- Die Anforderungen F-120 bis F-160 werden durch TIXI ZUG erbracht
- Weitere Details sind in den Dokumenten iTIXI_SRS.docx, iTIXI_SRS_Liste.xlsm auf sourceforge.net.

Vorgaben/Rahmenbedingungen

- Die Projektabwicklung ist „RUP lite" (SCRUM). Mindestens 3 Sprints sind vorgesehen, der Backlog wird anhand der Requirements Liste vereinbart (Betreuer, TIXI ZUG).
- Die Kommunikation in der Projektgruppe, in der Dokumentation und an den Präsentationen erfolgt auf Deutsch.
- Die Nutzungsrechte an der Software: GNU General Public License Version 2.0
  - Laut Symfony 2 Webseite, verträgt sich die MIT Lizenz mit der GPL.
  - Ebenso ist die Nutzung der Google Maps API V3 geklärt und unproblematisch.
- Ansonsten gelten die Rahmenbedingungen, Vorgaben und Termine der HSR.
- *Eigenleistungen von TIXI ZUG:*
  - *TIXI ZUG stellt die Software für iTIXI Iteration 4, Sprint 1 bis 17.02.2014 bereit.*
  - *TIXI ZUG bietet dem Diplomanden eine ein-tägige Einführung im TIXI Betrieb an.*
  - *Die erstellte Software soll durch TIXI ZUG laufend geprüft werden, dazu wird ein Test-System (z.B. mit Selenium), und ein Test Engineer zur Verfügung gestellt.*
  - *Am Ende eines jeden Sprints wird ein Testbericht zu Handen von TIXIZUG (Geschäftsleitung) und HSR (Betreuung) zur Verfügung gestellt.*
  - *TIXI ZUG stellt Test Daten, die aus dem Produktiv-System migriert werden, bereit.*
  - *Bei Bedarf, stellt TIXI ZUG einen Software Engineer bereit (Teilzeit).*
  - *Die Diplomanden sind für die Erreichung der vereinbarten Ziele (F-100 und F-110) verantwortlich. Sie leiten daher diese Projektphasen (weisungsbefugt), es steht ihnen frei diese Eigenleistungen abzulehnen, oder anzunehmen.*

Inhalt der Dokumentation

- Die Projektdokumentation (Prosa) und die Benutzerschnittstelle sind in Deutsch.
  Der Code, die Kommentare und die Versionsverwaltung sind in Englisch.
- Die fertige Arbeit muss folgende Inhalte haben:
  1. Abstract, Management Summary, Aufgabenstellung
  2. Technischer Bericht
  3. Projektdokumentation
  4. Anhänge (Literaturverzeichnis, Glossar, CD-Inhalt)
- Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind.
- Zitate sind zu kennzeichnen, die Quelle ist anzugeben.
- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen.
- Dokumentation des Projektverlaufes, Planung etc.
- Weitere Dokumente (z.B. Kurzbeschreibung, Poster) gemäss www.hsr.ch und gemäss Absprache mit dem Betreuer.

Form der Dokumentation

- Bericht (Struktur gemäss Beschreibung) gebunden (2 Exemplare) und in Ordner (1 Exemplar „kopierfähig" in losen, gelochten Blättern).
- Alle Dokumente und Quellen der erstellten Software auf sauber angeschriebenen CD (3 Ex.).

Bewertungsschema

Es gelten die üblichen Regelungen zum Ablauf und zur Bewertung der Arbeit des Studiengangs Informatik der HSR mit besonderem Gewicht auf moderne Softwareentwicklung.

Beteiligte

Diplomanden

1. Carmelo Schumacher
2. Reto Schelbert

Industriepartner

TIXI Zug, General Guisan Strasse 22, 6301 Zug.

- Yeldez Gwerder, Geschäftsleitung — Requirements Provider
- Josef Conrad, Vorstandsmitglied — Test Engineer
- Martin Jonasse, Fahrer — Software Engineer

Betreuung HSR

Verantwortlicher Dozent: Prof. Stefan Keller (sfkeller@hsr.ch), Geometa Lab am IFS der HSR

2 Erklärung zur Urheberschaft

Rapperswil, den 12.Juni 2014

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde.
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Reto Schelbert

Carmelo Schumacher

3 Abstract

Der gemeinnützige Verein TIXI ZUG bietet mit über 250 freiwilligen Mitarbeiterinnen und Mitarbeitern einen Fahrdienst für Menschen mit einer Behinderung an, die aufgrund ihrer Lebensumstände den öffentlichen Verkehr nicht benutzen können. Seit der Gründung des Vereins im Jahre 1984 ist die Nachfrage nach Beförderung kontinuierlich gestiegen. Um dem Bestreben einer möglichst uneingeschränkten Integration der möbilitätsbehinderten Menschen in Gesellschaft und Arbeitswelt auch in Zukunft nachkommen zu können, fasste TIXI ZUG den Entscheid, die bestehende Software für die Disposition und den Fahrbetrieb, die an ihre Grenze gekommen ist, durch eine Neuentwicklung zu ersetzen. Diese sollte einerseits die bestehende Funktionalität auf eine moderne und nachhaltige technologische Basis heben und andererseits mittels intelligenter Fahrwegsoptimierung dank effizienteren Routen aktiv Kosten einsparen helfen.

Um ein Maximum an Flexibilität und zukünftiger Erweiterbarkeit zu garantieren, sollte die Lösung als webbasierte Open-Source-Software unter der GNU Public Library License (GPL) auf einem LAMP-Stack entwickelt werden. Die funktionalen Anforderungen an das System sowie weitere Rahmenbedingungen wurden von TIXI ZUG erhoben und lagen in Form eines Pflichtenheftes vor. Die nötigen Technologien, Frameworks und Services von Drittanbietern zur bestmöglichen Erfüllung der
Vorgaben wurden im Rahmen dieser Bachelorarbeit evaluiert und eingesetzt. Die wichtigsten Eckdaten lauten wie folgt:

- Technologien: PHP 5.5, MySQL 5.6, HTML 5, CSS 3, Javascript
- Frameworks: Symfony 2.3 (mit Doctrine2-ORM und TWIG), jQuery
- Services: OSRM (Open Source Routing Machine), Google Maps API

Mit iTIXI wurde eine Webanwendung realisiert, die als operative Basis den TIXI-Betrieb bei der Umsetzung der für die Disposition notwendigen Prozesse unterstützt. Dies umfasst im Wesentlichen die folgenden Punkte:

- Die effiziente und sinnvolle Verwaltung von Fahrern, Fahrgästen, Fahrzeugen und weiteren Stammdaten
- Die Erfassung, Prüfung und Verteilung von Fahraufträgen auf möglichst effiziente Routen durch Lösen eines NP-schweren Optimierungsproblems
- Die quantitative Einsatzplanung von Fahrern und Fahrzeugen auf Jahres- und Monatsebene

4 Management Summary

4.1 Ausgangslage

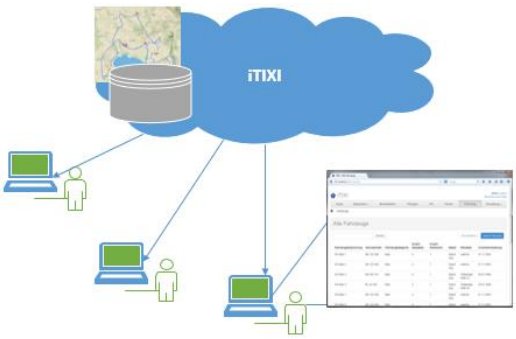
TIXI ZUG bietet einen Fahrdienst für Menschen mit einer Behinderung an und unterstützt damit kranke und betagte Menschen, welche die öffentlichen Verkehrsmittel nicht benützen können und somit in Ihrer Mobilität eingeschränkt sind. Durch den Verein TIXI ZUG wird es dank über 250 freiwilligen Mitarbeitern möglich, dass diese Menschen soziale Kontakte pflegen, an Kursen und Ausflügen teilnehmen oder Arzt- oder Therapiebesuche einhalten können. Die derzeitige windowsbasierte Softwarelösung für die Stammdatenverwaltung und Disposition der Fahrtenzuteilung ist überholt und entspricht nicht mehr den Ansprüchen der TIXI. Die dabei genutzten Technologien sind schlecht ausbaufähig, aufwändig in der Wartung und kostenintensiv für weitere Funktionalitäten. Deshalb soll eine neuzeitliche und nachhaltige Lösung in Form einer Webapplikation entwickelt werden, welche die Mitarbeiter der TIXI Zug bei der Planung, Datenverwaltung und Einteilung der Fahrten unterstützt und die jetzige Software komplett ablöst.

4.2 Vorgehen

Die Ablösung soll dabei durch eine rein webbasierte Open-Source Software realisiert werden und die Nutzungsrechte an der Lösung sollen durch die GNU Public Library License (GPL) frei verfügbar sein. Unsere Arbeit zielt dabei ab, grösstmögliche Flexibilität und zukünftige Erweiterbarkeit zu garantieren. Durch die webbasierte Lösung soll von jeder Betriebsplattform mit einem geeigneten Webbrowser gearbeitet werden können. Um Infrastrukturkosten zu reduzieren wird die entwickelte Webapplikation auf einem extern betreuten Webserver laufen, welcher die gesamte Logik und Datenablage beinhaltet und ein Inhouse Hosting überflüssig macht. Mit einer umfassenden Anforderungsanalyse soll die bereitgestellte Lösung auf die Bedürfnisse eines TIXI Betriebs abgestimmt und mit den benötigten Technologien realisiert werden.

4.3 Ergebnisse

Mit iTIXI wurde eine Basis erschaffen, die eine umfassende Stammdatenverwaltung anbietet, die Ressourcenplanung von Fahrer und Fahrzeugen unterstützt und Fahrtenberechnungen für die Tagesdisposition durchführt. Durch eine algorithmisch gestützte Optimierung der geplanten Fahrten kann eine effizientere Auftragsbehandlung durchgeführt werden, bei dem durch günstigere aufeinanderfolgende Fahrten Kosten eingespart werden dank minimal zurückgelegter Distanzen. Dank der Verwendung von OpenStreetMap und einer Open Source Routing Machine, können autonom und ohne weitere Nutzungs- oder Lizenzkosten Fahrt-Routen berechnet werden.



4.4 Ausblick

Es sind noch viele weitere Schritte notwendig, um eine vollständige Software für den Dispositionsbetrieb zu erhalten. Den zukünftigen Weiterentwicklungen sollte aber durch die erarbeiteten Grundlagen dieser Arbeit nichts mehr im Wege stehen und jegliche Erweiterungen werden durch die realisierte Architektur begünstigt.

5 Dokumentinformationen

5.1 Aufbau

Die Arbeit fängt mit einer ausführlichen Anforderungsspezifikation an, behandelt einige Grundlagen für weitere Kapitel und stellt in einem Umsetzungskonzept verwendete Technologien und Vorgehensweisen zur Implementierung vor. Im Design wird die Domain zum Problem aufgeführt und die Realisierung des Projekts kommt im Kapitel Implementierung zum Vorschein, das mit einem kurz aufgeführten Deployment zum Projekt abgerundet wird. Mit Projektmanagement und Projektstand werden weitere Informationen über die organisatorische Durchführung und den zukünftigen Weiterentwicklungen zum erhaltenen Resultat beigetragen. Einige Dokumente, die keinem Kapitel unterzuordnen sind, können aus dem Appendix bezogen werden. Weitere Arbeitsresultate, die nicht im Bericht enthalten sind, findet man wie folgt:

Arbeitsresultat	URL
iTIXI Build	http://tixi.seiout.ch
Code Dokumentation	http://tixi.seiout.ch
Repository	https://github.com/hertus/sfitixi
Sourceforge	http://sourceforge.net/projects/itixi/

5.2 Änderungsnachweis

Datum	Version	Änderung	Autor
06.05.2014	1.0	Erste Version des Dokuments	RS
05.06.2014	1.1	Abstract und Management Summary	CS
06.06.2014	1.2	Anforderungsspezifikation	RS
07.06.2014	1.3	Design	CS
07.06.2014	1.4	Grundlagen, Projektplanung	RS
08.06.2014	1.5	Umsetzungskonzept	RS
10.06.2014	1.6	Implementation	CS
10.06.2014	1.7	Implementation Weiterführung, Deployment	RS
11.06.2014	1.8	Design Domain Model	RS
11.06.2014	1.9	Projektstand	CS
12.06.2014	2.0	Appendix	RS

5.3 Inhaltsverzeichnis

1 Aufgabenstellung 1
2 Erklärung zur Urheberschaft 4
3 Abstract 5
4 Management Summary 6
4.1 Ausgangslage 6
4.2 Vorgehen 6
4.3 Ergebnisse 6
4.4 Ausblick 6
5 Dokumentinformationen 7
5.1 Aufbau 7
5.2 Änderungsnachweis 7
5.3 Inhaltsverzeichnis 9
6 Anforderungsspezifikation 13
6.1 Ist-Aufnahme 13
6.1.1 Kennzahlen (Jahr 2011) 13
6.1.2 Eingesetzte Softwarelösung 13
6.1.3 Planung und Disposition 14
6.1.4 Produktionsplan 15
6.1.5 Monatsplan 15
6.1.6 Tagesplan 16
6.1.7 Fahrt 19
6.1.8 Auswertungen 20
6.2 Spezifische Anforderungen 20
6.2.1 Nicht funktionale Anforderungen 20
6.2.2 Funktionale Anforderungsliste 22
6.3 Use Cases 22
6.3.1 Use-Case Diagramm 22
6.3.2 Aktoren und Stakeholders 23
6.3.3 Use Cases – Stammdaten CRUD 23
6.3.4 Use Cases – Disposition 25
6.4 Business Prozesse 31
6.4.1 Monatsplanung 31
6.4.2 Tagesdisposition 33
7 Grundlagen 36
7.1 Vehicle Routing Problem 36
7.1.1 DARP 37
7.1.2 Simulated Annealing 40

7.2 Routing-Machine 41
7.2.1 Datenmenge der Fahrtenabfrage 41
7.2.2 OSRM 42
7.3 Google Maps 42
8 Umsetzungskonzept 43
8.1 Technologien 43
8.1.1 LAMP 43
8.1.2 Frameworks und Libraries 43
8.2 Entwicklungsumgebung 44
8.2.1 Werkzeuge und Tools 44
8.2.2 Coding-Standards 45
8.3 Qualitätssicherung 45
8.3.1 Teammeetings / Sitzungsprotokolle 45
8.3.2 Reviews 45
8.3.3 Continous Integration 45
8.3.4 Unit-Tests 46
9 Design 47
9.1 Domain 47
9.1.1 Architekturentscheid Domain Model / Domain-Driven-Design 47
9.1.2 Entscheid OR-Mapper 48
9.1.3 Domain Model 49
10 Implementierung 54
10.1 Systemübersicht 54
10.1.1 Gesamtsicht 54
10.1.2 ApiBundle 54
10.1.3 App und AppBundle 61
10.1.4 CoreDomain 63
10.1.5 CoreDomainBundle 64
10.2 iTIXI als Informationssystem 65
10.2.1 Menükonzept / Sitemap 65
10.2.2 ID-Konzept 68
10.2.3 Controller-API-Funktionen 70
10.2.4 Data Grid 71
10.2.5 Adressverwaltung 71
10.3 Disposition 76
10.3.1 Produktionsplan verwalten 76
10.3.2 Monatsplan verwalten 79
10.4 Routen Abfrage 80

10.4.1 RouteManagement ... 80
10.4.2 RoutingMachine ... 83
10.5 Fahrwegsoptimierung ... 86
10.5.1 Konfigurationserzeugung ... 86
10.5.2 Konfigurationsanalyse ... 89
10.5.3 Annealing ... 90
10.6 Security ... 92
10.6.1 Roles ... 93
10.6.2 OAuth 2.0 ... 93
11 Deployment ... 94
11.1 Build ... 94
11.1.1 Umgebungsparameter ... 94
11.1.2 Applikations Parameter ... 95
11.1.3 Symfony + Composer ... 96
11.1.4 Doctrine + Database ... 96
11.2 Technologie-Stack ... 97
11.2.1 OSRM Update ... 98
11.3 Commands ... 98
11.3.1 Adresskoordinaten abrufen ... 98
11.3.2 Routen aktualisieren ... 98
11.3.3 Fahrtoptimierungen vorausberechnen (angedacht) ... 99
12 Projektmanagement ... 100
12.1 Arbeitsaufwand ... 100
12.2 Gantt-Diagramm ... 101
12.3 Meilensteine und Arbeitspakete ... 102
12.3.1 MS0.0 Projekteinarbeitung ... 102
12.3.2 MS0.1 Code- und Anforderungsanalyse abgeschlossen ... 102
12.3.3 MS0.2 Neues Architekturkonzept erarbeitet ... 102
12.3.4 MS0.3 Domain Model+REST API bis Iteration 3 umgesetzt ... 103
12.3.5 MS0.4 CRUD Architektur Client bis Iteration 3 abgeschlossen ... 103
12.3.6 MS0.5 Dispositionsprozess definiert ... 103
12.3.7 MS0.6 Detailspezifikation Disposition abgeschlossen ... 104
12.3.8 MS0.7 Domain Model bis Iteration 4 implementiert ... 104
12.3.9 MS0.8 Implementationen Disposition abgeschlossen ... 104
12.3.10 MS1.0 Fahrwegsoptimierung implementiert ... 105
12.3.11 MS1.1 Testings und Refactorings abgeschlossen ... 105
12.3.12 MS1.2 Projektabgabe ... 105
12.4 Risikomanagement ... 106

12.4.1 Risikobewertung 106
13 Projektstand 107
14 Erfahrungsbericht 108
14.1 Reto Schelbert 108
14.2 Carmelo Schumacher 109
15 Glossar 110
15.1 Domain 110
15.2 Weitere Begriffe 111
16 Literaturverzeichnis 113
17 Abbildungsverzeichnis 113
18 Inhalt der CD 115
19 Appendix 117
19.1 Sitzungsprotokolle 117
19.2 System-Testdokumente 129
19.3 Domain Model (Grossformat) 149
19.4 Annealing Tests 150
19.5 Benutzeranleitungen 153
19.5.1 Build on Windows 153
19.5.2 Technologie-Deployment 164

6 Anforderungsspezifikation

Um die bestehende Software im TIXI Betrieb ab zu lösen, bedarf es einer genaueren Analyse über die Arbeitsabläufe und Belangen der Mitarbeitenden. Viele Anforderungen wurden bereits durch unseren Projektpartner spezifiziert und vorgeschlagen, dank eines ausführlichen und begleiteten Besichtigungstag im Betrieb vor Ort konnten wir uns aber selbst ein Bild der jetzigen Situation machen. Aus diesen Impressionen ergab sich eine Anforderungsspezifikation mit besonderem Fokus auf die Disposition.

6.1 Ist-Aufnahme

6.1.1 Kennzahlen (Jahr 2011)

Einige Kennzahlen zum Betrieb helfen uns den Umfang und die Grösse der Business Domain zu erkennen:

- 18 Fahrzeuge
- 240 Fahrer,
- 2'778 Fahrgäste,
- 46'158 Fahraufträge/ Jahr
- 638'288 gefahrene Kilometer/ Jahr
- Orte von Interesse (ca. 300): Arztpraxen (187), Spitäler (1), Kliniken (34) und Therapien (80)

Laut Projektpartner wurden im Jahr 2012 an die 48'896 Fahraufträge[1] durchgeführt, was im Durschnitt 134 Fahraufträge pro Tag bedeutet. Gegenüber dem Jahr 2011 bedeutet dies bereits eine Zunahme von gut 5.9%. Der Verein hat vor zu wachsen und wird sich in absehbarer Zeit neue Fahrzeuge beschaffen. Die Anzahl Fahraufträge, welche man definitiv durchführen kann, wird mit Bestimmtheit noch einmal zunehmen.

6.1.2 Eingesetzte Softwarelösung

Die momentan eingesetzt Lösung basiert auf einer Microsoft Access Datenbank, welche seit 1997 im Einsatz steht. Diese wurde fortlaufend weiterentwickelt und nahm stets an Funktionsumfang zu. Mehrere Access Dateien werden per SMB (Server Message Block) über einen Windows Server zentral für die Clients zur Verfügung gestellt.

Zu den Clients zählen 3-4 Windows Stationen die von Geschäftsleitung und Mitarbeitenden bedient werden. Der Aufbau dieser MS Access Datenbank ist laut Aussagen der Projektpartner ungenügend oder gar nicht dokumentiert, eine Wartung durch Dritte wird praktisch verunmöglicht.

Der Funktionsumfang der MS Access Lösung umfasst eine Verwaltung von Kundendaten, das Erfassen von Fahraufträgen und die Tagesplanung für die Disposition dieser Fahraufträge zu Fahrzeugen und Fahrern. Hinzu kommen eine mögliche Ressourcenplanung von benötigten Fahrzeugen und Bereitstellung verschiedener Dokumente, sowie das Versenden von E-Mails, wobei dies an eine E-Mailanwendung auf den Client Geräten ausgelagert wird.

[1] gemäss Kennzahlen im Dokument *iTIXI_Google_Challgeges.ppt*: *Google Maps Daily Load* auf Sourceforge

6.1.3 Planung und Disposition

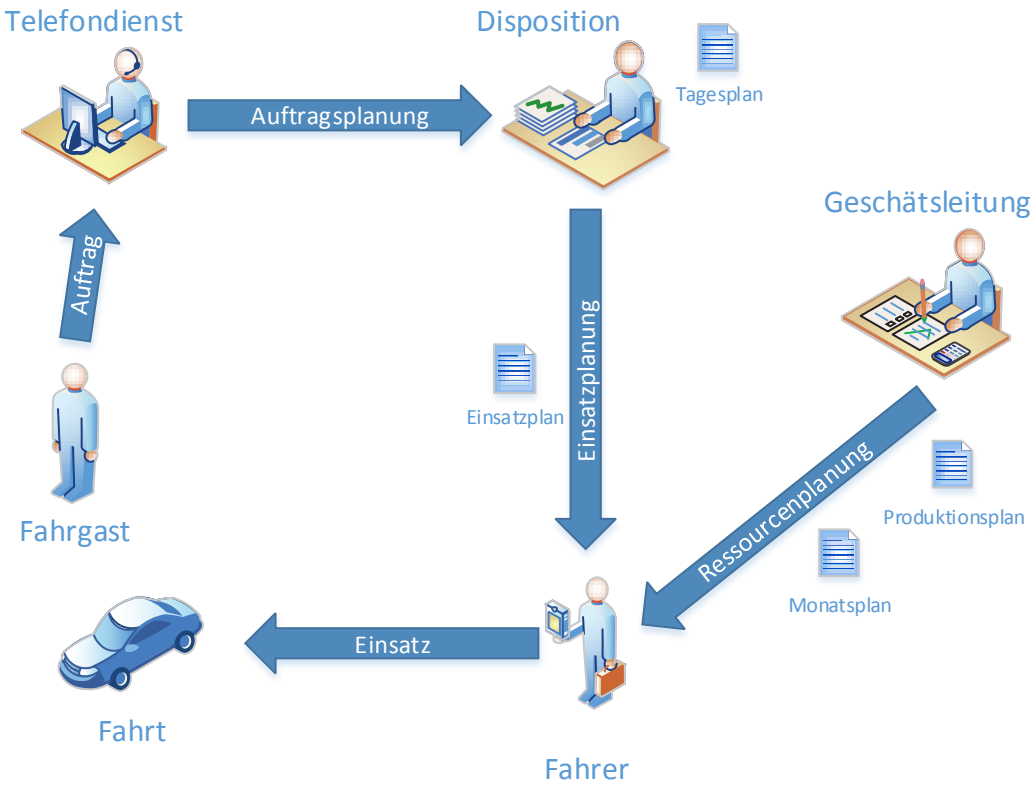


Abbildung 1: Grober Arbeitsablauf TIXI Zentrale

Wie in Abbildung 1 dargestellt, kann man aus den Arbeitsabläufen in der Zentrale der TIXI Zug im Groben die Tätigkeitsgebiete der Mitarbeitenden erkennen.

Geschäftsleitung

- Plant Ressourcen für die kommenden Monate ein um eine ungefähre Anzahl an Fahrer und Fahrzeugen bereit zu halten.

Fahrgast

- Gibt eine Fahrt in Auftrag

Mitarbeiter im Telefondienst

- Nehmen Telefonate entgegen um neue Fahraufträge zu erfassen von Fahrgästen, welche später zur Auftragsplanung den Mitarbeitern in der Disposition weitergereicht werden.

Mitarbeiter in der Disposition

- Erstellen aus vorhandenen Fahraufträgen und Ressourcen Tagespläne für die kommenden Tage.
- Stellen Einsatzpläne für Fahrer bereit.

Fahrer

- Führen Fahrtaufträge aus gemäss Einsatzplan

6.1.4 Produktionsplan

Anhand Erfahrungen, Daueraufträgen, Fernfahrten, Feiertagen und saisonalen Bedingungen wird ein Produktionsplan von der Geschäftsleitung erstellt, welcher die Anzahl benötigter Fahrzeuge für einen Tag definiert. Dazu werden in einer Maske der MS Access Lösung Zahlenwerte für die Anzahl Fahrzeuge zu einem Monatstag und Arbeitsschicht eingetragen.

6.1.5 Monatsplan

In einem Monatsplan werden angeknüpft am Produktionsplan genügend Fahrer für die Anzahl benötigter Fahrzeuge gesucht. Die Informationsbeschaffung über die Fahrtbereitschaft von Fahrern ist hierbei sehr unstrukturiert. Das können einzelne ausgedruckte E-Mails von Fahrern sein, die Ihre Einsatzwünsche oder Absenzen mitteilen in einer Form wie *„kann am 17.02 nicht, dafür gerne Schicht 1 am 18.02…“*.
Es können aber auch schriftliche sowie mündliche Mitteilungen von Fahrern sein, die auf Notizblättern oder in Formularfeldern wie „Details“ oder „Einsatzwunsch“ innerhalb der Fahrerverwaltung der MS Access Lösung festgehalten werden. Dieser werden allesamt für die Monatsplanung zusammengesucht. Dadurch, dass diese Informationen nicht in einer geeigneten und strukturierten Form vorliegen und aus allen möglichen Quellen nach verfügbaren Fahrer gesucht werden muss, ist dieser Vorgang zur Monatsplanung für die Geschäftsleitung sehr aufwändig. Es werden deshalb oft keine geeigneten Fahrer für bestimmte Arbeitsschichten im vornherein gefunden. Diese werden dann in einem Dokument für alle Fahrereinsätze eines Monates mit einem Fragezeichen als fehlende Fahrer dargestellt (siehe Abbildung 2).

Fahrereinsätze März 2014

	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo
Wer / Tag	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
?	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2	.	.	.	.	.	.	2
?	2	.	2	.	.	2	1	.	.	2	.	1	2	4	.	.	2	2	3	2	4	.	.	2	.	3	2	1	2	2	2
?	.	.	.	.	.	2	.	.	.	.	.	.	.	.	.	.	.	.	.	2	.	.	.	2	.	.	2	.	.	.	2
?	.	.	2	.	.	2	.	.	.	2	.	1	2	4	.	.	2	2	.	2	1	.	.	2	.	.	2	1	.	.	2
?	4	2	2	2	2	4	1	2	7	2	1	1	4	4	4	7	2	7	6	4	4	7	7	2	7	1	4	4	7	4	2
?	4	.	2	2	.	4	1	.	.	2	3	1	4	4	2	.	2	2	6	4	4	4	4	2	4	1	2	1	4	4	2
?	7	7	2	4	7	7	4	2	7	2	4	1	4	4	4	7	6	7	7	7	4	7	7	2	7	1	7	4	7	3	6
Achermann Hermann	.	.	.	.	.	2	.	.	.	.	.	.	2	.	.	.	.	.	.	2	.	.	.	.	.	.	2	.	.	.	.
Aeberli Walter	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	1	.	.	.	.
Alisic Hamdija	.	.	2	.	2	.	2	.	.	2	.	2	.	2	.	.	2	.	2	.	2	.	.	2	.	2	.	2	.	.	2

4=Schicht 1+2; 5=Schicht 1+3; 6=Schicht 2+3; 7=Schicht 1+2+3 Seite 1 von 4

Abbildung 2: Fahrereinsätze Monatsplanung

Alle Fahrer erhalten für den nächsten Monat einen Einsatzplan per E-Mail als PDF-Dokument oder als Brief in ausgedruckter Form. Die Fahrer melden sich dann wiederum für eine Arbeitsschicht an, bei der noch kein Fahrer gefunden wurde. Diese Meldung erfolgt telefonisch oder per Mail und die Mitarbeiter oder die Geschäftsleitung tragen die fehlenden Fahrer von Hand ein (siehe Abbildung 3).

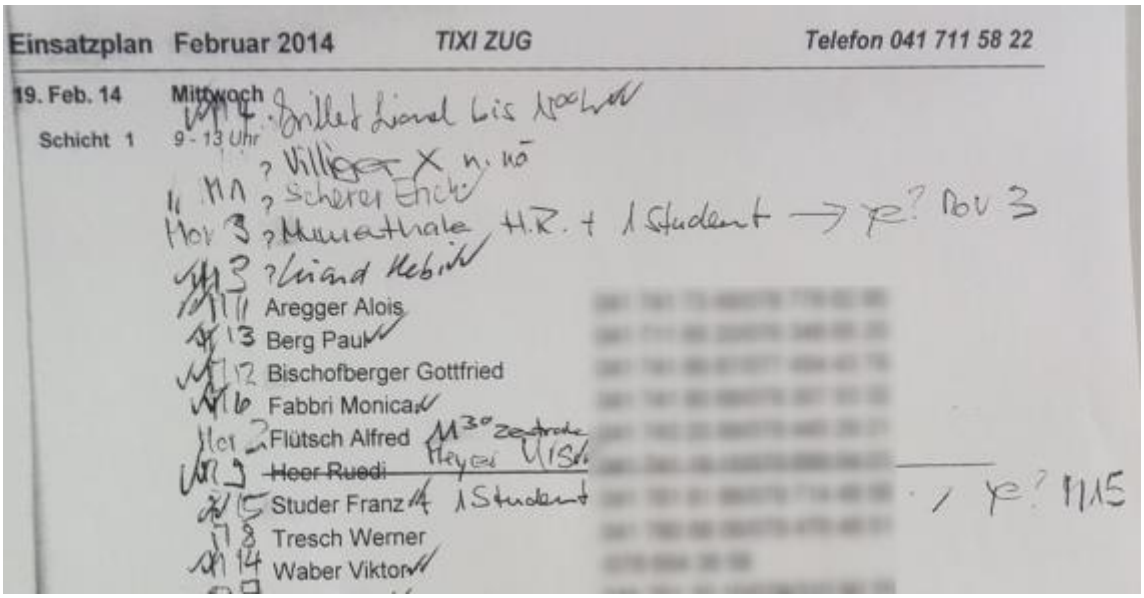


Abbildung 3: Einsatzplan Ergänzungen und Änderungen von Hand

Werden noch Fahrer für eine Arbeitsschicht benötigt, wird ein Rundtelefon durch die Mitarbeiter durchgeführt und Kontakt mit freiwilligen Fahrern aufgenommen um alle Lücken zu schliessen.

Alle nachträglichen Änderungen werden schriftlich auf einem ausgedruckten Einsatzplan ergänzt und anschliessend für die Tagesplanung verwendet, bei der es um die Disposition der Fahrer zu Fahrzeugen und Fahraufträgen geht.

6.1.6 Tagesplan

In der TIXI Zug arbeiten meist zwei Mitarbeiterinnen an der Tagesplanung, eine im Telefondienst und eine weitere Mitarbeiterin in der Disposition.

6.1.6.1 Telefondienst

Die Mitarbeiterin im Telefondienst nimmt neue Fahraufträge und Änderungen an bestehenden Aufträgen entgegen. Bei der TIXI Zug ist eine Fahrtbestellung nur telefonisch möglich. Dabei ist Ihre Vorgehensweise für einen neuen Auftrag wie folgt:

1. Telefonat wird entgegen genommen
2. Fahrgast wird aus Dropdownlist ausgewählt oder neu erfasst
3. Fahrauftrag für diesen Fahrgast wird neu erstellt oder von einem bestehenden kopiert
4. Datum für den Fahrauftrag wird aus einem Kalender von Hand gesucht um den Wochentag zu bestätigen
5. Wenn möglich wird eine Grafik der Tagesplanung (siehe Abbildung 5) aufgerufen oder die Mitarbeiterin in der Disposition gefragt ob dieser Auftrag zeitlich machbar ist
6. Aus dem gleichen Auftrag heraus wird eine Rückfahrt erstellt falls gewünscht
7. Der Auftrag wird erstellt und steht zur Disposition bereit

Unterstützt mit Informationen zu Fahrgast und Fahrt (siehe Abbildung 4) wird ein neuer Auftrag erfasst und steht anschliessend für die Disposition bereit um Fahrer und Fahrzeug zu zuordnen.

Fahraufträge werden meist Tage im Vorherein bestellt. Laut Projektpartner entstehen ca. 80% der Fahrten durch Daueraufträge. Spontanfahrten werden nicht durchgeführt.

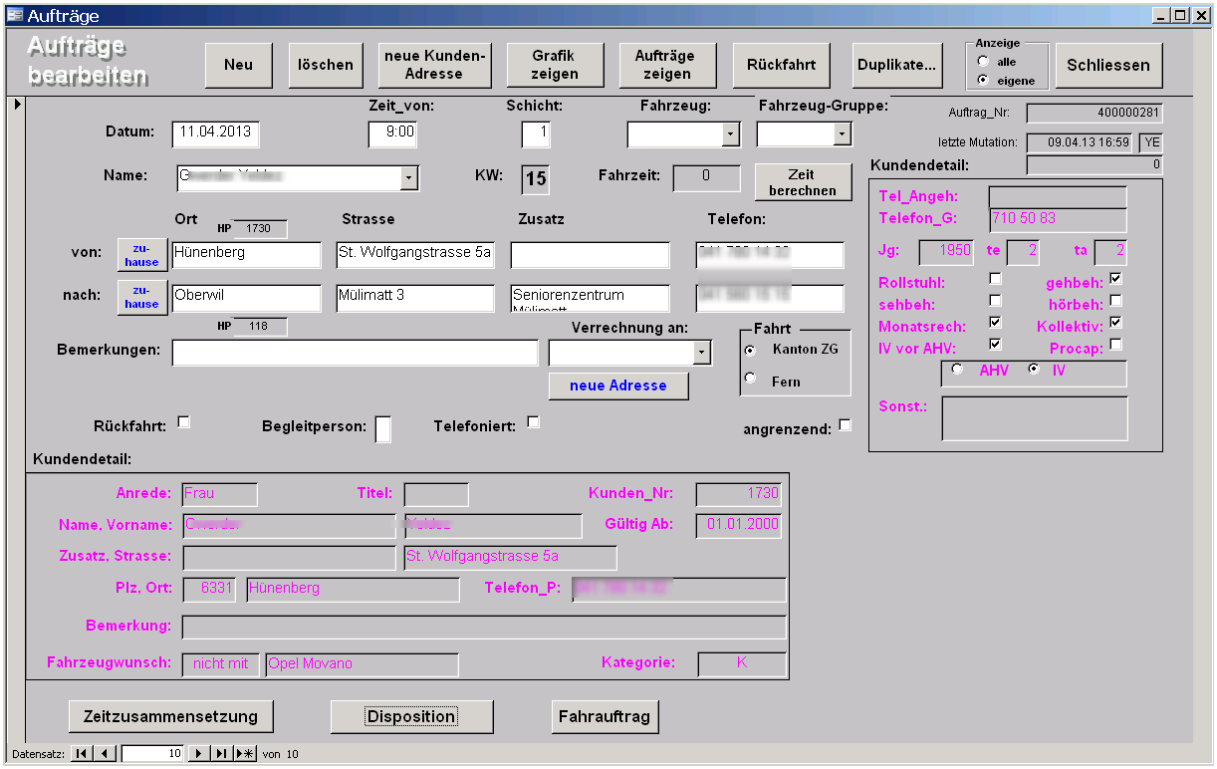


Abbildung 4: Fahrauftragserfassung (Quelle: iTIXI_GUI.ppt auf Sourceforge)

Es ist möglich eine Fahrzeit zu berechnen. Wie diese Berechnung zustande kommt, möchten wir im nächsten Abschnitt kurz erläutern.

6.1.6.2 Berechnung Fahrtdauer

Eine Fahrtdauer für erfasste Fahraufträge steht den Mitarbeitern der MS Access Lösung zur Verfügung. Diese wird nicht exakt durch Abruf eines Routing-Services bestimmt, aber recht genau angenähert dank eines Bushalteplans des Zugers Verkehrsverband ZVB. Da für die TIXI Zug ein Grossteil der durchgeführten Fahrten im oder um den Kanton Zug durchgeführt wird, können bereits viele Ortschaften durch ein Netz von Bushaltestellen abgebildet werden. Die Fahrtdauer kommt zustande in dem man für Fahrtadressen die nächste Bushaltestelle angibt mit einer zusätzlichen Dauer angibt, wie lange man etwa brauchen wird diese Adresse von der Bushaltestelle aus anzufahren. Zusammen mit den Zeiten im Busfahrplan, die uns angibt wie lange der ZVB rechnet um von einer Haltestelle zur nächsten zu fahren, ergibt sich die errechnete Fahrtdauer. Dadurch konnte man in der TIXI Zug bereits mit einer recht hohen Genauigkeit die Dauer von Fahrten berechnen lassen um damit ein Zeitfenster für die Tagesplanung zu erhalten. Es kommen aber immer Umstände für eine Fahrt hinzu, die nicht berechnet werden können, sei es ein Stau in der Innenstadt, eine schlecht ausgebaute Strasse oder weitere Faktoren die eine effektive Fahrtdauer beeinflussen.

Findet ein Auftrag aber ausserhalb der Busfahrzonen statt, wird die Fahrtdauer manuell von den Mitarbeiterinnen über Online-Dienste wie Google Maps abgefragt und eingetragen.

6.1.6.3 Disposition

Die Mitarbeiterin in der Disposition arbeitet an einer Arbeitsstation mit zwei Bildschirmen. Auf einem Bildschirm werden noch nicht disponierte Fahraufträge angezeigt, auf dem anderen Bildschirm wird eine DataGrid[2] ähnliche Anzeige von disponierten Fahraufträgen zu Fahrzeug und Fahrer dargestellt (siehe Abbildung 5).

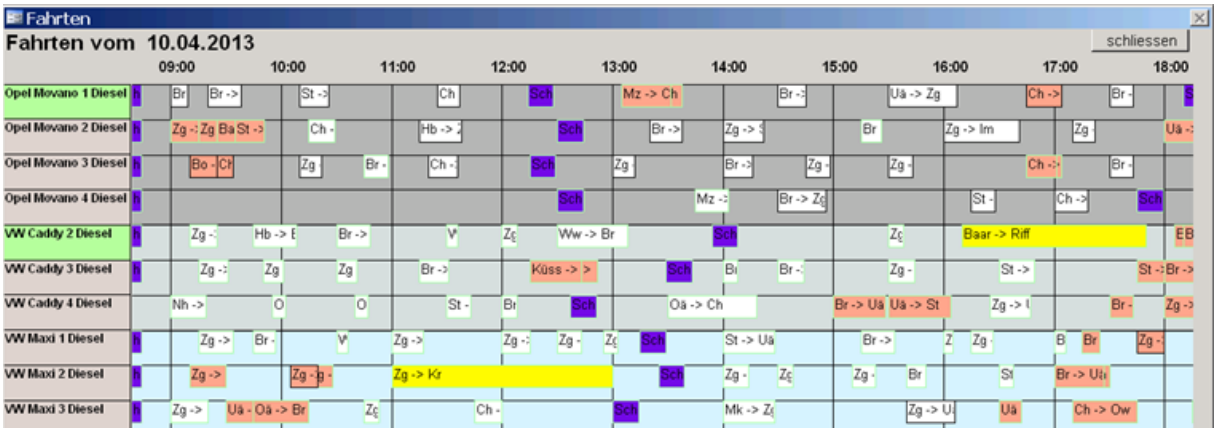


Abbildung 5: Tagesansicht Disposition (Quelle: iTIXI_GUI.ppt auf Sourceforge)

Die Vorgehensweise für die Disposition von Fahraufträgen ist für die Mitarbeitende wie folgt:

1. Ein zu disponierender Tag (Datum) wird ausgewählt
2. Alle nicht disponierten Fahraufträge erscheinen chronologisch sortiert auf einer Bildschirmmaske und stehen zur weiteren Bearbeitung bereit
3. Ein Fahrauftrag wird einem zur Verfügung stehenden Fahrer zugeteilt.
   Dabei werden einige Muss- und Soll-Kriterien beachtet:
   a. Muss-Kriterien:
      - Fahrer kann vollumfänglich mit dem Kunden umgehen
      - Fahrer ist bereit Rollstuhlfahrten durchzuführen (auch physisch im Stande dazu) falls der Fahrauftrag Rollstuhlfahrer beinhaltet

   b. Soll-Kriterien:
      - Fahrer bedient nur gewisse Fahrzeuge (z.B. möchte nur kleine Fahrzeuge fahren)
4. Der Fahrauftrag wird dann in der Tagesansicht einem zu dieser Zeit verfügbaren Fahrzeug zugeordnet.
   Dabei werden wieder einige Muss- und Soll-Kriterien beachtet:
   a. Muss-Kriterien:
      - Fahrzeug muss rollstuhltauglich sein falls der Fahrauftrag Rollstuhlfahrer beinhaltet
      - Es sind genügend Sitzplätze vorhanden für Fahrgast und Begleitpersonen, besonders bei einem Sammelauftrag

   b. Soll-Kriterien:
      - Fahrgast möchte nur mit gewissen Fahrzeugen fahren (z.B. keine grossen Fahrzeuge)

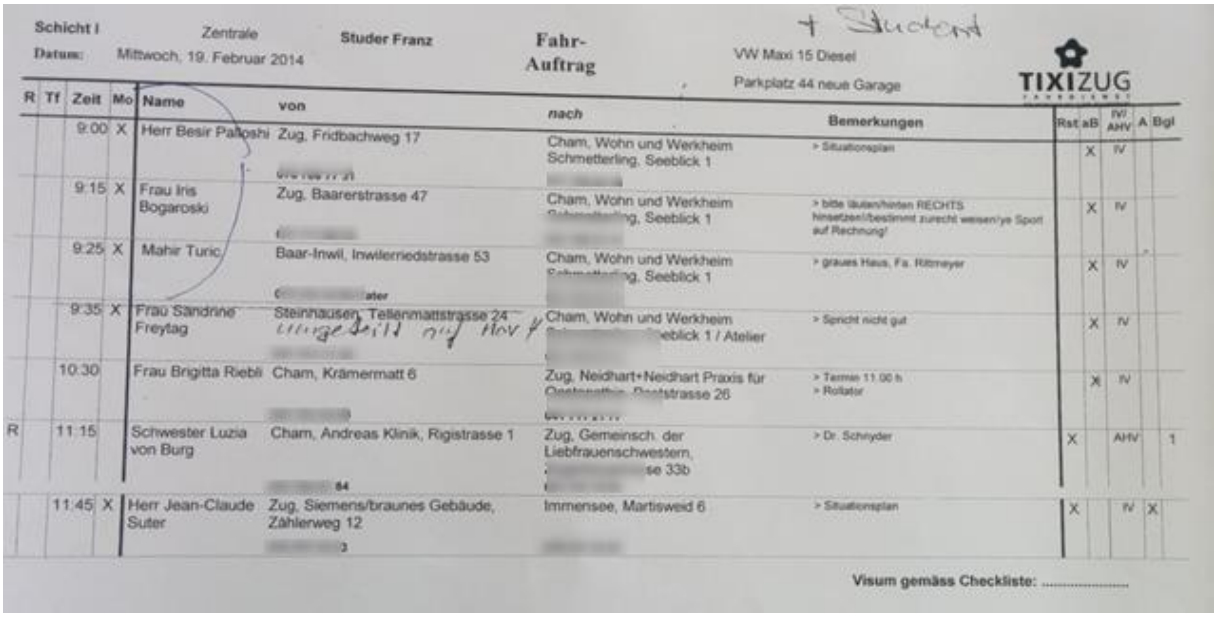
Innerhalb der Tagesansicht können die Fahraufträge per Drag-and-Drop einem Fahrzeug zu einem bestimmten Zeit Slot zugeordnet werden. Durch Farbkodierung werden Einzelaufträge, Sammelaufträge und Fernfahrten unterschieden und mit Abkürzungen die Start- und Ziel-Orte der jeweiligen Aufträge bezeichnet. Anfang und Ende der jeweiligen Arbeitsschichten können angepasst werden je nach Zuteilung von Fahrer und Fahrauftrag.

Alle Zuteilungen werden soweit von Hand durch die Disponentin durchgeführt und es fliessen viele Erfahrungswerte mit ein bei der Entscheidung ob gewisse Fahraufträge nacheinander durchgeführt werden können. Das System unterstützt immerhin mit Warnungen, wenn zwei Fahraufträge in zu kurzer Zeit aufeinanderfolgen. Diese Berechnung stützt sich wiederum auf die Dauer- und Distanz-Angaben im Bushalteplan.

[2] DataGrid: Ein Steuerelement, welches Daten in einem definierten Raster anzeigt.

Sind alle offenen Positionen disponiert und kamen in der Zwischenzeit keine neuen Fahraufträge mehr herein, wird die Tagesplanung abgeschlossen und für weitere Bearbeitungen gesperrt. Dies ist meist gegen Abend um 17:00 Uhr der Fall, ab hier wird dann anhand der disponierten Positionen für alle Fahrer ein Informationsblatt (siehe Abbildung 6) zu allen Fahraufträgen pro Schicht zugestellt per E-Mail oder Fax.

Jedes Informationsblatt wird ausgedruckt und am nächsten Tag den Fahrern vor Schichtbeginn ausgehändigt. Falls nach der Sperrung noch Änderungen eingehen, werden diese per Handnotiz auf den ausgedruckten Exemplaren ergänzt. Nachträgliche Änderungen an der Tagesplanung werden erst am nächsten Tag von der Disponentin eingetragen, da auch im Verlaufe des Tages Absagen von Fahraufträgen vorkommen können oder gar nicht durchgeführt werden, wenn z.B. ein Fahrgast nicht erschienen ist.



Schicht I Zentrale Studer Franz **Fahr-Auftrag** VW Maxi 15 Diesel

Datum: Mittwoch, 19. Februar 2014 Parkplatz 44 neue Garage

TIXIZUG

R	Tf	Zeit	Mo	Name	von	nach	Bemerkungen	Rst	aB	IV/ AHV	A	Bgl
		9:00	X	Herr Besir Palloshi	Zug, Fridbachweg 17	Cham, Wohn und Werkheim Schmetterling, Seeblick 1	> Situationsplan		X	IV		
		9:15	X	Frau Iris Bogaroski	Zug, Baarerstrasse 47	Cham, Wohn und Werkheim [illegible], Seeblick 1	> bitte lauten/hinten RECHTS hinsetzen!/bestimmt zurecht weisen/ye Sport auf Rechnung!		X	IV		
		9:25	X	Mahir Turic	Baar-Inwil, Inwilerriedstrasse 53	Cham, Wohn und Werkheim [illegible], Seeblick 1	> graues Haus, Fa. Rittmeyer		X	IV		
		9:35	X	Frau Sandrine Freytag	Steinhausen, Tellenmattstrasse 24	Cham, Wohn und Werkheim [illegible] eblick 1 / Atelier	> Spricht nicht gut		X	IV		
		10:30		Frau Brigitta Riebli	Cham, Krämermatt 6	Zug, Neidhart+Neidhart Praxis für [illegible] Poststrasse 26	> Termin 11.00 h > Rollator		X	IV		
R		11:15		Schwester Luzia von Burg	Cham, Andreas Klinik, Rigistrasse 1	Zug, Gemeinsch. der Liebfrauenschwestern, [illegible] se 33b	> Dr. Schnyder	X		AHV		1
		11:45	X	Herr Jean-Claude Suter	Zug, Siemens/braunes Gebäude, Zählerweg 12	Immensee, Martisweid 6	> Situationsplan	X		IV	X	

Visum gemäss Checkliste: .....

Abbildung 6: Informationsblatt Fahrauftrag

6.1.7 Fahrt

Die Fahrer erhalten wie erwähnt ein Informationsblatt am Vorabend ihres Einsatzes. Die darin enthaltenen Fahrten schauen sich die Fahrer an und ziehen sich allenfalls die Hilfe eines Online-Dienstes bei um die genaue Fahrtroute nachzuschauen. Die exakte Fahrtroute wird nicht durch die TIXI Zug bestimmt und es werden auch keine Pläne vorgegeben, hier stehen die Fahrer frei in der Auswahl, welche Strassen sie wählen um ans Ziel zu kommen. Am Einsatztag erhält der Fahrer dann vor Schichtbeginn sein ausgedrucktes Exemplar vom Informationsblatt inklusive Änderungen durch die Disponentin.

Die Disponentin kopiert vorhandene ausgedruckte Routenpläne oder Satellitenbilder von Standorten, falls diese Informationen vom Fahrer noch benötigt werden. Es gibt immer Adressen, welche auch bei Online-Diensten noch nicht erfasst wurden oder über gar keine Strassen-Nummer verfügen. Diese Standorte sind nicht ohne weiteres auffindbar und solche Routen- oder Situationspläne sind dabei eine grosse Hilfe.

Für den Fahrer befinden sich auf dem Informationsblatt zudem weitere nützliche Hinweise und Bemerkungen zum Fahrgast oder dem Standort. Dies sind einerseits Hinweise über Verhaltensmerkmale der Fahrgäste, z.B. jemand der bei der Fahrt eventuell dazu neigt ins Steuerrad zu fassen sollte nicht als Beifahrer vorne hocken, andererseits Hinweise über die physische Verfassung oder weitere Merkmale die man beachten sollte im Umgang mit dem Fahrgast. Auch genaue Standortdaten werden hierbei mitgegeben, z.B. die Zimmer Nummer falls sich der Fahrgast in einem Altersheim aufhält oder eine Ansprechperson an die sich der Fahrer wenden kann falls der Fahrgast nicht auffindbar ist. Diese Einträge werden von verschiedensten Feldern in der MS Access Lösung zusammengezogen.

Sind für den Fahrer alle Details geklärt, tritt er die Fahrt an und bezieht sein zugewiesenes Fahrzeug im Fahrzeugdepot. Die TIXI Zug verfügt momentan nur über ein Depot im Hauptsitz Zug für alle Fahrzeuge.

Der Fahrer überprüft vor Fahrtbeginn das Fahrzeug auf Mängel und führt dementsprechend ein Logbuch in dem auch der Kilometerstand für statistische Auswertungen nachgeführt wird.

Wie erwähnt wählt der Fahrer seine eigene beste Fahrtroute aus um zu den Abhol- oder Zieladressen zu fahren. Die ideale Route kann je nach Erfahrung oder Kenntnis über das Strassennetz von Fahrer zu Fahrer variieren.

6.1.8 Auswertungen

Unterstützt durch die MS Access Lösung wird jährlich von der Geschäftsleitung eine Auswertung erstellt mit den Eckdaten:

- Anzahl Aufträge
- Anzahl Fahrten, was den effektiv durchgeführten Aufträgen entspricht
- Gefahrene Kilometer, welche aus dem Logbuch der Fahrzeuge entnommen werden

Die TIXI Zug möchte vor allem die total gefahrenen Kilometer auf ein Minimum zu reduzieren.

Da sie als Verein mit freiwilligen Fahrern keine weiteren Kosten haben wenn sie zusätzliche Fahrer einstellen für mehr Fahraufträge pro Tag, kommen die Kosten mehrheitlich nur durch die gefahrenen Kilometer zusammen.

6.2 Spezifische Anforderungen

Viele Anforderungen an die iTIXI Applikation sind bereits durch die Projektpartner vorgegeben[3]. Diese haben wir hier noch einmal zusammengestellt und mit unserer Analyse aus dem Besichtigungstag ergänzt.

6.2.1 Nicht funktionale Anforderungen

6.2.1.1 Zuverlässigkeit

Eine hohe Verfügbarkeit ist massgebend um den Betrieb in der TIXI Zentrale aufrecht zu erhalten. Bei Ausfall sollte eine Wieder-Instandstellung innerhalb eines Tages möglich sein.

Ein Backup-Server soll innerhalb der TIXI-Organisation bereitgestellt sein um über ein Backup Daten Stand den Betrieb wieder auf zu nehmen. Ein Backup muss täglich durchgeführt werden.

Falls Abhängigkeiten zu Internet-Diensten bestehen wie etwas Google Maps API, muss eine Fallback Lösung her, sollte diese einmal nicht erreichbar sein.

6.2.1.2 Leistung

Die iTIXI Applikation soll beim Aufrufen innerhalb von 5 Sekunden reagieren.

Funktionalitäten die für die Disposition zwingend sind sollen innerhalb 1 Sekunde reagieren.

Bereitstellungen von Auswertungen oder Rapporten können länger dauern, sollen aber über eine Fortschrittsanzeige den Benutzer auf dem Laufenden halten.

6.2.1.3 Erweiterbarkeit

Weiterentwicklungen an der Applikation sollten ohne grössere Änderungen möglich sein damit in zukünftigen Entwicklungsschritten einfach weitere Funktionalitäten implementiert werden können.

[3] Gemässe Anforderungskatalog *iTIXI_SRS_Liste.xlsm* und dem System Requirements Specification Dokument *iTIXI_SRS.docx* auf Sourceforge

6.2.1.4 Sicherheitsanforderungen

Nur berechtigte Benutzer dürfen Zugriff auf die Applikation und Datenbestand erhalten. Ein Login-System muss ausreichend sicher sein.

Benutzereingaben dürfen zu keiner Zeit zu einer Verletzung der Datenintegrität führen und dürfen die Zuverlässigkeit des Systems nicht beeinträchtigen. Fehler sollen zudem in einer Log-Datei mit Benutzername, Zeitstempel und Fehlerdaten aufgezeigt werden.

6.2.1.5 Lizenzanforderungen

Die komplette iTIXI Applikation soll auf rein Open-Source basierten Technologien aufbauen und keine weiteren Lizenzen benötigen. Die Nutzungsrechte der an der Softwarelösung sollen durch die GNU Public Library License (GPL) frei verfügbar sein.

6.2.1.6 Schnittstellen

Benutzerschnittstelle	Die Applikation soll auf gängigen Browsern wie Safari, Firefox, Internet Explorer lauffähig sein und auf grossen Arbeitsflächen (grosse Bildschirme) skalieren.
Hardwareschnittstellen	Die fertigentwickelte Lösung wird bei einem Provider gehostet. Die Applikation soll auf 64-Bit Betriebssysteme ausgerichtet sein. Bereitgestelle Dokumente werden Client-seitig ausgedruckt.
Softwareschnittstellen	Ein Export an ein Buchhaltungssystem ist vorgesehen.
Kommunikationsschnittstellen	Die Applikation kommuniziert über TCP/IP mit den Clients

6.2.2 Funktionale Anforderungsliste

Alle Funktionalen Anforderungen sind aus der Liste iTIXI_SRS.xlsm zu entnehmen.

6.3 Use Cases

6.3.1 Use-Case Diagramm

Farblich kodiert wurde nach Anwendungsfällen der Stammdatenverwaltung (CRUD[4] Funktionalitäten) und nach Anwendungsfällen die für die Disposition relevant sind.

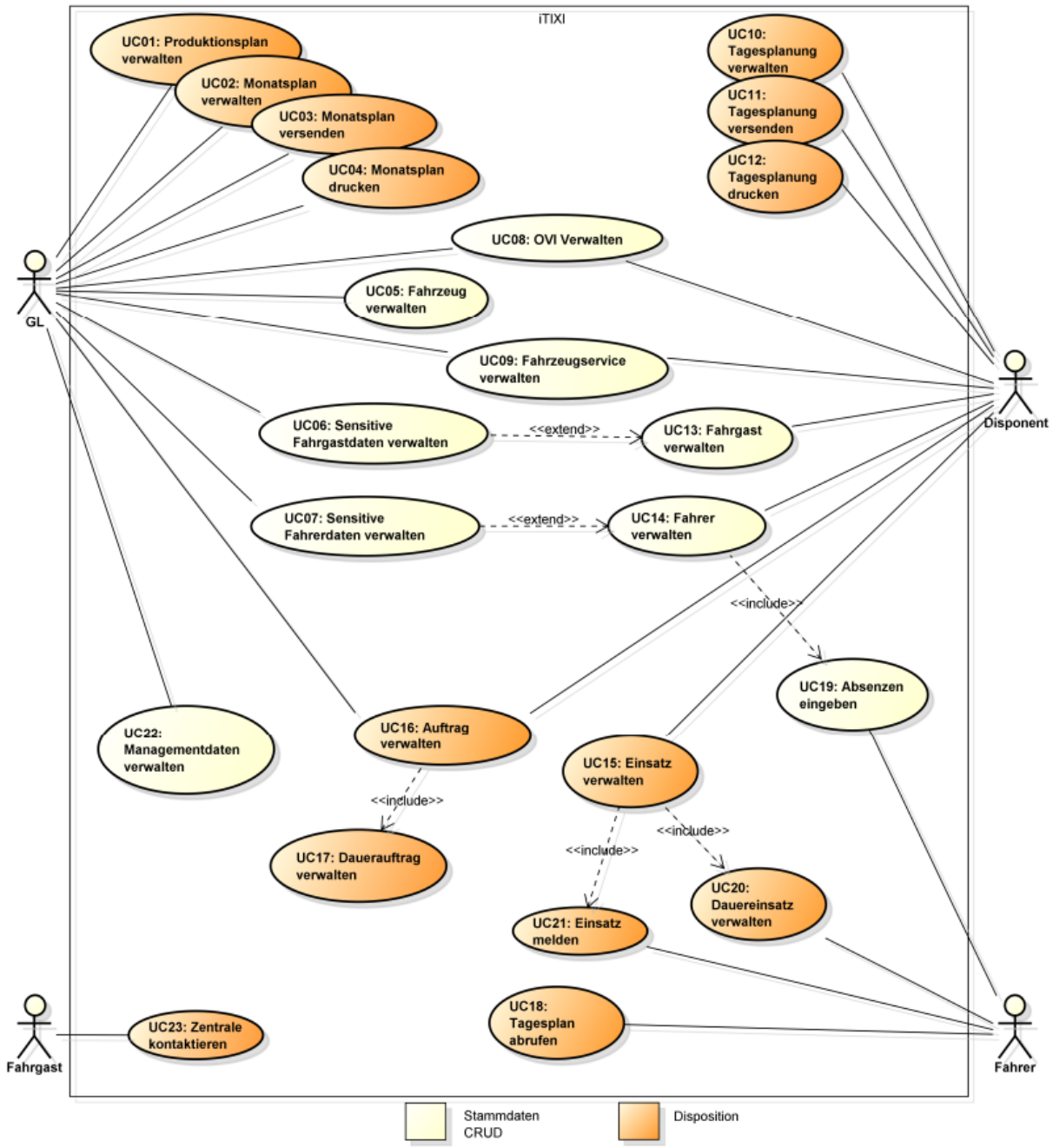


Abbildung 7: Vollständiges Use-Case Diagramm

[4] CRUD: Akronym für die grundlegenden Operationen **C**reate, **R**ead, **U**pdate, **D**elete

6.3.2 Aktoren und Stakeholders

GL	Geschäftsleitung, benötigt in erster Linie Instrumente zur Ressourcenplanung (Fahrerschichten, Fahrzeuge) und erstellt für die Mitarbeitenden Einsatzpläne. Pflegt zudem sensitive Daten.
Disponent	Mitarbeitende im Telefondienst oder Disposition. Möchte Fahraufträge und Einsätze für die Tagesplanung sinnvoll einteilen und zur Verfügung stellen. Pflegt zudem die Fahrer und Fahrgastdaten.
Fahrer	Will Arbeitszeiten definieren können und wissen, wann Einsätze geleistet werden müssen.
Fahrgast	Möchte Fahrten in Auftrag geben indem die TIXI Zentrale kontaktiert wird

6.3.3 Use Cases – Stammdaten CRUD

Die Beschreibung der CRUD Use Cases erfolgt im *brief format*.

6.3.3.1 UC05: Fahrzeug verwalten

UC05	Fahrzeug verwalten
	Die GL verwaltet die Daten über alle verfügbaren Fahrzeuge. Bei Neuanschaffungen werden neue Fahrzeuge eingetragen. Die Fahrzeuge müssen über Angaben verfügen wie Name, Fahrschildnummer, Anzahl Sitzplätze und Rollstuhlplätze. Den Fahrzeugen wird zudem ein Betreuer zugeordnet der wöchentliche Wartungen am Fahrzeug durchführt. Alle einsatzfähigen Fahrzeuge dienen dann als Ressource bei der Produktionsplanung und werden dementsprechend für Fahreinsätze eingeteilt und den Fahrern zugeordnet.

6.3.3.2 UC09: Fahrzeugservice verwalten

UC09	Fahrzeugservice verwalten
	Fahrzeuge können zu bestimmten Tagen im Service zur Wartung oder Instandhaltung sein. GL oder Disponent verwalten diese Servicepläne zu den entsprechenden Fahrzeugen, welche nachfolgend nicht zur Verfügung stehen bei der Ressourcenplanung und während diesen Tagen auch nicht einsatzfähig sind.

6.3.3.3 UC08: OVI verwalten

UC08	OVI verwalten
	GL und Disponent erfassen und pflegen die Orte von Interesse, sprich Adressen die häufig gebraucht werden und mit zusätzlichen Informationen ausgestattet sind. Dies können z.B. Pflegeheime sein, Arztpraxen oder Physiotherapien. OVI‘s werden mit diesen Begriffen gekennzeichnet was später für eine Statistikauswertung (nicht in Aufgabenstellung dieser Arbeit) dienen wird, ob z.B. ein Grossteil von Fahrten in gewissen Bereichen stattfindet.

6.3.3.4 UC13: Fahrgast verwalten

UC13	Fahrgast verwalten
	Fahrgastdaten werden gepflegt durch den Disponent. Hierzu gehören die komplette Personen und Adressdaten des Fahrgasts sowie weitere wichtige Informationen zu Versicherung, Behinderung, Rollstuhlbenutzung und Verrechnungsart. Weitere Angaben werden zum Fahrgast gemacht, die auch einen Einfluss auf die Disposition haben. Beispielsweise möchten gewisse Fahrgäste nicht mit grossen Fahrzeugen mitfahren – meiden also eine bestimmte Fahrzeugkategorie. Es kommen zudem Bemerkungen zum Fahrgast hinzu die auf dem Informationsblatt für die Fahrer aufgeführt werden.

6.3.3.5 UC06: Sensitive Fahrgastdaten verwalten

UC06	Sensitive Fahrgastdaten verwalten
	Die GL erhält zusätzlich zu der Möglichkeit den Fahrgast zu verwalten eine Ansicht um sensitive Fahrgastdaten zu verwalten, welche von den anderen Benutzern nicht eingesehen werden dürfen. Dies können Informationen über Verhalten oder Bezahlungen sein, welche geschäftsintern eine gewisse Brisanz haben.

6.3.3.6 UC14: Fahrer verwalten

UC14	Fahrer verwalten
	Fahrerdaten werden, ähnlich wie bei UC13 zum Fahrgast, gepflegt durch den Disponent. Hierzu gehören die kompletten Personendaten mit E-Mail-Adresse oder Fax-Nummer um dem Fahrer schnell Informationen zu zuschicken wie z.B. Einsatzpläne. Dem Fahrer werden zudem Fahrzeugkategorien zugeordnet welche er nicht fahren möchte und ob er bereit ist z.B. Rollstuhlfahrten durchzuführen. Diese Angaben fliessen wiederum in die Disposition mit ein und spielen bei der Fahrzeugzuteilung eine Rolle.

6.3.3.7 UC07: Sensitive Fahrerdaten verwalten

UC07	Sensitive Fahrerdaten verwalten
	Die GL erhält, ähnlich wie bei UC06, zusätzlich zu der Möglichkeit den Fahrer zu verwalten eine Ansicht um sensitive Fahrerdaten zu verwalten, welche von den anderen Benutzern nicht eingesehen werden dürfen. Dies können Informationen über Fahrverhalten oder Unfälle sein.

6.3.3.8 UC19: Absenzen verwalten

UC19	Absenzen verwalten
	Zum *Fahrer verwalten* (UC14) können zusätzlich Absenzen erfasst werden. Diese werden von den Fahrern mitgeteilt und von Disponent oder GL zusammen mit den Fahrerdaten eingetragen. Die Absenzen - d.h. Tage an denen der Fahrer nicht anwesend oder fahrbereit sein wird weil er z.B. in den Ferien ist – spielen bei der Ressourcenplanung eine Rolle, bei der schon mögliche Fahrer an Arbeitsschichten zugeteilt werden.

6.3.3.9 UC22: Managementdaten verwalten

UC22	Managementdaten verwalten
	Zu den Stammdaten zählen noch viele weitere Angaben: Daten für das innerbetriebliche Management und für die Verwendung der Applikation. Wir haben diese Vorgänge unter Managementdaten zusammengefasst. Die GL tätigt hierbei Anpassungen an Daten die Einfluss haben auf den Gebrauch der Applikation und nicht für die normalen Mitarbeiter zugänglich sind.

6.3.4 Use Cases – Disposition

Die Beschreibung der Use Cases für die Disposition erfolgt im *fully dressed* Format.

6.3.4.1 UC01: Produktionsplan verwalten

UC01	Produktionsplan verwalten
Ziel	GL hat einen monatlichen Produktionsplan vor sich mit Anzahl Fahrzeugen die zu einem bestimmten Datum und Schicht benötigt werden und kann diesen bearbeiten. Benötigte Fahrzeuge beruhen einerseits auf Schätzungen von GL und Disponent und andererseits auf Vorjahresdaten und Daueraufträgen.
Primärakteure	GL
Stakeholder	GL
Vorbedingungen	Definierte Arbeitsschichten Fahrzeuge eingetragen Daueraufträge von Fahrgästen (optional) Vorjahresdaten (optional)
Nachbedingungen	Produktionsplan für Monat XY
Fehlerfall	Es konnte kein Produktionsplan erstellt werden aufgrund fehlender Daten, d.h. sind keine Arbeitsschichten oder Fahrzeuge erfasst worden.
Standard Ablauf	1. Erstellt/ändert Produktionsplan für Monat XY 2. Trägt Anzahl benötigter Ressourcen pro Tag und pro Arbeitsschicht ein 3. Verwendet Produktionsplan für Monatsplanung
Alternativer Ablauf	
Spezialanforderungen	
Auftrittshäufigkeit	Monatlich

6.3.4.2 UC02: Monatsplan verwalten

UC02	Monatsplan verwalten
Ziel	GL hat einen Monatsplan vor sich, der als Einsatzplan von Fahrer und Fahrzeuge dient. Dabei erscheinen alle aktiven Fahrer (auch Zivilschützer) in einer Fahrer-pro-Schicht-und-Tag Maske und enthält dabei die Anzahl benötigter Fahrer pro Tag und Arbeitsschicht. Fehlende Fahrer werden mit einem Platzhalter eingetragen und werden später von Fahrer oder Disponent ergänzt. Dauereinsätze und Absenzen von Fahrer werden dabei berücksichtigt.
Primärakteure	GL Disponent
Stakeholder	GL Disponent Fahrer
Vorbedingungen	UC01: Produktionsplan Dauereinsätze vorhanden (optional)
Nachbedingungen	Monatsplan für Monat XY erstellt Einsätze für Fahrer eingetragen
Fehlerfall	Es kann kein Monatsplan für Monat XY erstellt werden da noch kein Produktionsplan von Monat XY vorhanden ist.
Standard Ablauf	1. GL erstellt/ändert Monatsplan für Monat XY aus Produktionsplan 2. System trägt mögliche Fahrer in Tag und Schicht ein für Monat XY a. Beachtet Dauereinsätze b. Beachtet Absenzen c. Trägt Platzhalter bei Tag und Schicht ein, bei der kein passender Fahrer gefunden wurde 3. GL oder Disponent trägt fehlende Fahrer im Monatsplan ein

Alternativer Ablauf	
Spezialanforderungen	Fahrer verfügen über Dauereinsätze, damit überhaupt schon ein Monatsplan mit Fahrern vorbesetzt werden kann vom System.
Auftrittshäufigkeit	Erstellung Monatlich Eintrag fehlender Fahrer bei Bedarf mehrmals täglich

6.3.4.3 UC03: Monatsplan versenden

UC03	**Monatsplan versenden**
Ziel	Ein erstellter Monatsplan für Monat XY soll von der GL aus an alle Fahrer versendet werden, die dann das Dokument „Fahreinsätze“ bzw. „Einsatzplan“ für den Monat XY erhalten und dementsprechend ihre nächsten Einsätze terminieren können oder sich noch für weitere Einsätze melden können die noch einen Fahrer benötigen. Die Versendung erfolgt per E-Mail an die Fahrer, welche E-Mailadressen hinterlegt haben.
Primärakteure	GL
Stakeholder	GL Fahrer
Vorbedingungen	UC02: Monatsplan Fahrer mit E-Mailadresse vorhanden
Nachbedingungen	Monatsplan für Monat XY versendet an Fahrer YZ
Fehlerfall	Es kann kein Monatsplan für Monat XY versendet werden wenn: • Kein Monatsplan für Monat XY vorhanden • Kein Fahrer mit E-Mailadresse hinterlegt
Standard Ablauf	1. GL wählt Monatsplan für Monat XY aus 2. GL klickt auf versenden 3. GL erhält Status über versendete E-Mails an Fahrer
Alternativer Ablauf	Fahrer hat keine E-Mailadresse -> UC04: Monatsplan drucken
Spezialanforderungen	
Auftrittshäufigkeit	Monatlich

6.3.4.4 UC04: Monatsplan drucken

UC04	**Monatsplan drucken**
Ziel	Für Fahrer die keine E-Mailadresse haben, sowie auch für Vereinsmitglieder die noch interessiert am Vereinswesen sind, wird ein Monatsplan für den Monat XY ausgedruckt und per Post zugeschickt.
Primärakteure	GL
Stakeholder	GL Fahrer Vereinsmitglieder
Vorbedingungen	UC02: Monatsplan
Nachbedingungen	Monatsplan für Monat XY ausgedruckt
Fehlerfall	Es kann kein Monatsplan für Monat XY gedruckt werden wenn: • Kein Monatsplan für Monat XY vorhanden
Standard Ablauf	1. GL wählt Monatsplan für Monat XY aus 2. GL klickt auf drucken
Alternativer Ablauf	
Spezialanforderungen	Client mit Drucker vorhanden
Auftrittshäufigkeit	Monatlich

6.3.4.5 UC15: Einsatz verwalten

UC15	Einsatz verwalten
Ziel	Nachdem ein Monatsplan für Monat XY erstellt wurde, werden Einsätze an Fahrern vom System aus oder von Disponent aus zugeordnet. Ein Einsatz wird aus einem Tag und einer bestimmten Arbeitsschicht bestehen an dem der Fahrer bereit sein muss Fahrten durchzuführen. Änderungen an Einsätzen werden von Disponent nachgetragen.
Primärakteure	Disponent
Stakeholder	Disponent Fahrer
Vorbedingungen	Monatsplan für Monat XY vorhanden und versendet
Nachbedingungen	Einsatz wurde für Fahrer erstellt/geändert Einsatzplan kann für Fahrer bestimmt werden
Fehlerfall	
Standard Ablauf	1. Disponent ordnet Einsatz einem Fahrer zu oder entfernt diesen
Alternativer Ablauf	
Spezialanforderungen	Fahrer verfügbar an diesem Tag
Auftrittshäufigkeit	Mehrmals täglich

6.3.4.6 UC21: Einsatz melden

UC21	Einsatz melden
Ziel	Im Einsatzplan sind fehlende Fahrer vermerkt. Fahrer melden sich für entsprechende Einsätze. Diese werden durch den Disponent eingetragen. Für weitere Entwicklungen ist denkbar, dass der Fahrer dies über einen Online-Dienst gleich direkt einträgt.
Primärakteure	Disponent
Stakeholder	Disponent Fahrer
Vorbedingungen	Monatsplan für Monat XY vorhanden und versendet
Nachbedingungen	Offener Einsatz dem Fahrer zugeordnet
Fehlerfall	
Standard Ablauf	1. Disponent ordnet Einsatz dem meldenden Fahrer zu
Alternativer Ablauf	
Spezialanforderungen	
Auftrittshäufigkeit	Mehrmals täglich

6.3.4.7 UC20: Dauereinsatz verwalten

UC20	Dauereinsatz verwalten
Ziel	Fahrer kann sich für wiederkehrende Einsätze ein- und austragen. Diese sind z.B. gedacht für Einträge in der Form von „Jeden 2. Dienstag Schicht 1 und 2“. Diese werden durch den Disponent eingetragen. Für weitere Entwicklungen ist denkbar, dass der Fahrer dies über einen Online-Zugang gleich direkt einträgt.
Primärakteure	Disponent Fahrer
Stakeholder	GL Disponent Fahrer
Vorbedingungen	Fahrer im System vorhanden
Nachbedingungen	Dauereinsatz für Fahrer wurde erstellt/geändert und es werden mögliche Einsätze für den Fahrer zugeordnet.

Fehlerfall	
Standard Ablauf	1. Disponent erstellt/ändert Schema für Dauereinsatz beim Fahrer
Alternativer Ablauf	
Spezialanforderungen	Änderungen an Dauereinsatz lösen Change Management aus für bereits zugeteilte Einsätze des Fahrers oder um neue Einsätze für offenen Monatsplan dem Fahrer zuzuordnen.
Auftrittshäufigkeit	Eher Wöchentlich

6.3.4.8 UC16: Auftrag verwalten

UC16	**Auftrag verwalten**
Ziel	Disponent (oder auch GL) erfassen und verwalten Aufträge auf Anfrage von Fahrgästen.
Primärakteure	Disponent
Stakeholder	Disponent Fahrgast
Vorbedingungen	Fahrgast vorhanden, wenn nicht -> UC13 um neuen Fahrgast zu erstellen
Nachbedingungen	Auftrag wurde erstellt
Fehlerfall	Es konnte kein Auftrag erstellt werden wenn: • Nicht genügend Kapazitäten vorhanden um Auftrag zu diesem Datum/Zeit durchzuführen
Standard Ablauf	1. Disponent erstellt/ändert Auftrag für Fahrgast 2. Trägt Datum, Zeit und Standort ein 3. System überprüft Machtbarkeit des Auftrages 4. Auftrag wird vom System erstellt und für zur Disposition frei gegeben
Alternativer Ablauf	Kein Fahrgast vorhanden: 1. Disponent erstellt Fahrgast und fährt mit Standard Ablauf fort Fahrgast möchte Auftrag mehrfach durchführen: 1. Disponent erstellt aus Auftrag einen Mehrfachauftrag -> UC17 Es besteht Möglichkeit für Sammelauftrag: 5. Disponent erstellt aus dem erstellten Auftrag einen Sammelauftrag mit mehreren Fahrgästen von weiteren ähnlichen Aufträgen
Spezialanforderungen	Machbarkeit für Auftrag muss gewährleistet werden Neuer Auftrag -> Neue Route: • Adresse abfragen/persistieren • Fahrtroute abfragen/persistieren
Auftrittshäufigkeit	Mehrmals täglich

6.3.4.9 UC17: Mehrfachauftrag verwalten

UC17	**Mehrfachauftrag verwalten**
Ziel	Disponent möchte einen wiederkehrenden Auftrag für einen Fahrgast erfassen/ändern. Aufträge können Mehrfachaufträge sein die wiederkehrend oder einige male nacheinander an verschiedenen Daten durchgeführt werden müssen.
Primärakteure	Disponent
Stakeholder	Disponent Fahrgast
Vorbedingungen	Fahrgast vorhanden UC16: Auftrag wird erstellt
Nachbedingungen	Mehrfachauftrag erstellt aus dem einzelne Aufträge herauswachsen.
Fehlerfall	Es konnte kein Mehrfachauftrag erstellt werden wenn: • Nicht genügend Kapazitäten vorhanden um Auftrag zu diesem Datum/Zeit durchzuführen • Zu viele ähnliche Mehrfachaufträge zu dieser Zeit/Datum durchgeführt

	werden und gar nicht genügend Fahrzeuge dafür vorhanden sind
Standard Ablauf	1. Disponent erfasst/ändert nach Bedürfnis ein Mehrfachauftrag aus einem eröffnetem Auftrag heraus 2. Trägt wiederkehrende Datum/Wochentag/Zeit ein 3. System überprüft Machbarkeit 4. Mehrfachauftrag wird vom System erfasst, Aufträge werden anhand des Mehrfachauftrags erstellt und für Disposition frei gegeben
Alternativer Ablauf	
Spezialanforderungen	Änderungen an Mehrfachauftrag muss Change Management für die aus diesem Mehrfachauftrag erstellen Einzelaufträgen auslösen.
Auftrittshäufigkeit	Täglich

6.3.4.10 UC10: Tagesplanung verwalten

UC10	**Tagesplanung verwalten**
Ziel	Disponent bearbeitet Tagesplan für die Einteilung von Aufträgen Pro Schicht zu Fahrer und Fahrzeugen. Durch die Fahrtoptimierung kann bereits eine Konfiguration/Zuteilung als Vorschlag vorhanden sein pro Schicht. Disponent kann soweit aber manuelle Änderungen vornehmen um Aufträge anderen Fahrern oder Fahrzeugen zuzuordnen. Nicht zugeordnete Aufträge sollen an Fahrer und Fahrzeug zugeordnet werden können. Aus Aufträgen mit gleichem Start oder Ziel kann wenn gewünscht ein Sammelauftrag entstehen. Die Tagesplanung dient soweit als ein wichtiges Werkzeug für die Disposition damit schlussendlich Fahrten zustande kommen.
Primärakteure	Disponent
Stakeholder	Disponent Fahrer
Vorbedingungen	• Monatsplan für Monat XY in dem sich der Tagesplan für Tag XY befindet • Erfasste Aufträge für diesen Tag • Vorhandene Fahrer durch erstellte Einsätze
Nachbedingungen	Tagesplan vorhanden
Fehlerfall	Es sind gewisse Konfiguration nicht möglich: • Zu viele Aufträge vorhanden um einer Schicht zuzuordnen -> Change Management muss eintreten um z.B. mehr Fahrzeuge anzufordern
Standard Ablauf	1. Disponent teilt sequentiell offene Aufträge an Fahrern zu 2. Teilt Fahrer und Auftrag einem Fahrzeug zu 3. Kontrolliert Konfiguration 4. Tagesplan wird gesperrt und ist bereit für Versenden UC11
Alternativer Ablauf	Disponent führt Fahrtoptimierung für Schicht aus: 1. Kontrolliert erstellte Konfiguration für Schicht 2. Trägt Änderungen ein, Schicht wird gesperrt 3. Tagesplan wird gesperrt und ist bereit für Versenden UC11
Spezialanforderungen	Disponent kann nach Fahrwegoptimierung eingreifen. Fahrwegoptimierung kann manuell ausgeführt werden um eine Konfiguration für eine Schicht automatisch zu erstellen. Tagesplan kann nach lock bis zum Folgetag nicht mehr bearbeitet werden. Ab Folgetagen können Änderungen an Fahrten und Aufträgen eingetragen werden.
Auftrittshäufigkeit	Mehrmals täglich

6.3.4.11 UC11: Tagesplanung versenden

UC11	Tagesplanung versenden
Ziel	Nach dem fertig stellen einer Tagesplans wird diese durch den Disponent versendet als „Informationsblatt“ über Fahrtaufträge zu den korrespondieren Fahrern.
Primärakteure	Disponent
Stakeholder	Disponent Fahrer
Vorbedingungen	• Tagesplan für Tag YZ fertig erstellt und gesperrt • Fahrer zugeteilt an Auftrag im Tagesplan YZ + verfügt über E-Mail-Adresse
Nachbedingungen	Informationsblatt für Tagesplan YZ an Fahrer versendet
Fehlerfall	E-Mail konnte nicht gesendet werden
Standard Ablauf	1. Disponent wählt Tagesplan aus 2. Klickt auf versenden und erhält Status über versendete Mails
Alternativer Ablauf	Disponent druckt Informationsblatt für Tagesplan YZ aus um dieses per Fax oder über andere Wege einem Fahrer ohne E-Mail-Adresse zu zuschicken.
Spezialanforderungen	
Auftrittshäufigkeit	Täglich

6.3.4.12 UC12: Tagesplanung drucken

UC12	Tagesplanung drucken
Ziel	Disponent möchte nach dem fertig stellen eines Tagesplans und nachdem diese versendet wurde (kein Muss-Kriterium) zusätzlich eine Kopie ausdrucken um ein physisches Exemplar den Fahrern zu überreichen am Folgetag bevor sie die Schicht beginnen. Änderungen an Fahraufträgen und weitere Informationen werden soweit von Hand eingetragen.
Primärakteure	Disponent
Stakeholder	Disponent Fahrer
Vorbedingungen	• Tagesplan für Tag YZ fertig erstellt und gesperrt • Fahrer zugeteilt an Auftrag im Tagesplan YZ
Nachbedingungen	Informationsblatt für Tagesplan YZ und Fahrer YX ausgedruckt
Fehlerfall	E-Mail konnte nicht gesendet werden
Standard Ablauf	1. Disponent wählt Tagesplan aus 2. Klickt auf drucken
Alternativer Ablauf	Disponent druckt Informationsblatt für Tagesplan YZ aus um dieses per Fax oder über andere Wege einem Fahrer ohne E-Mail-Adresse zu zuschicken.
Spezialanforderungen	Client mit Drucker vorhanden
Auftrittshäufigkeit	Täglich

6.4 Business Prozesse

Einige Abläufe im Betrieb lassen sich gut modellieren und werden für unsere weitere Entwicklungen von grossem Nutzen sein. Die sogenannte Monatsplanung und die Tagesdisposition sind nachfolgend per BPMN (Business Process Model and Notation) abgebildet und werden im Detail betrachtet.

6.4.1 Monatsplanung

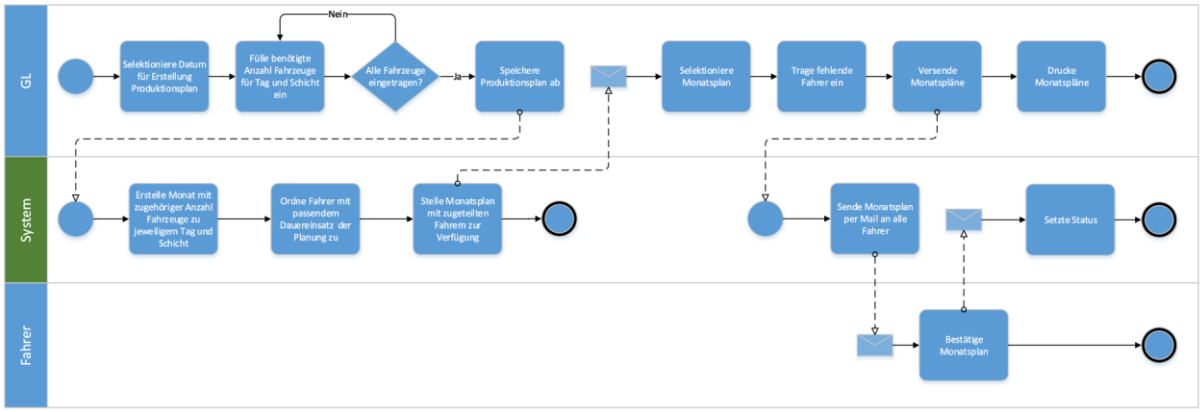


Abbildung 8: BPM Monatsplan Übersicht

Ein Produktionsplan wird von der Geschäftsleitung für den nächsten Monat erstellt. Dieser ist soweit bindend für Fahrer und Fahrzeug die zum Einsatz bereit gehalten werden an einem Tag und Schicht.

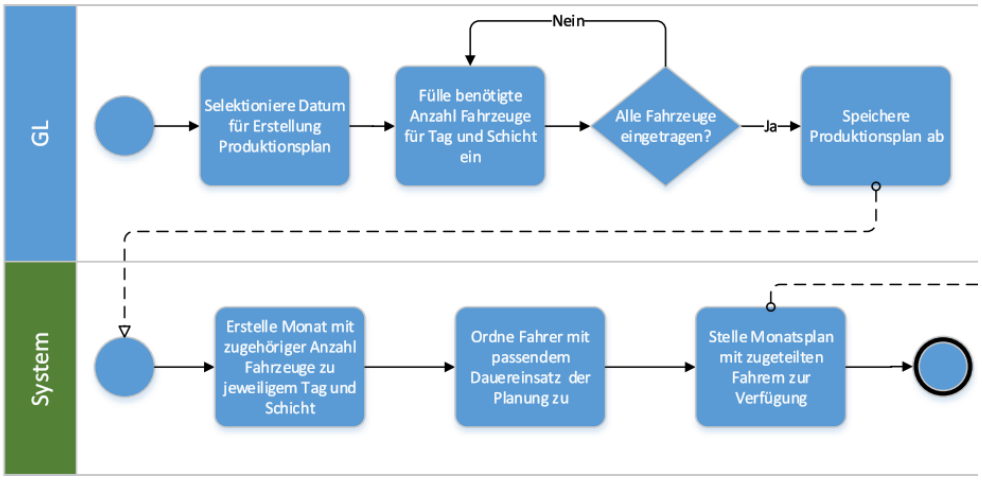


Abbildung 9: BPM Monatsplan Teil 1

Abbildung 9: Wird ein Produktionsplan erstellt, muss für jeden Tag und Schicht die Anzahl benötigter Fahrzeuge eingetragen werden, wobei die Anzahl Fahrzeuge auch der Anzahl Fahrer entspricht. Ausgehend davon erstellt iTIXI einen Monatsplan und versucht bereits vorhandene Fahrer, die über einen Dauereinsatz verfügen, den bereitgestellten Schichten einzuteilen.

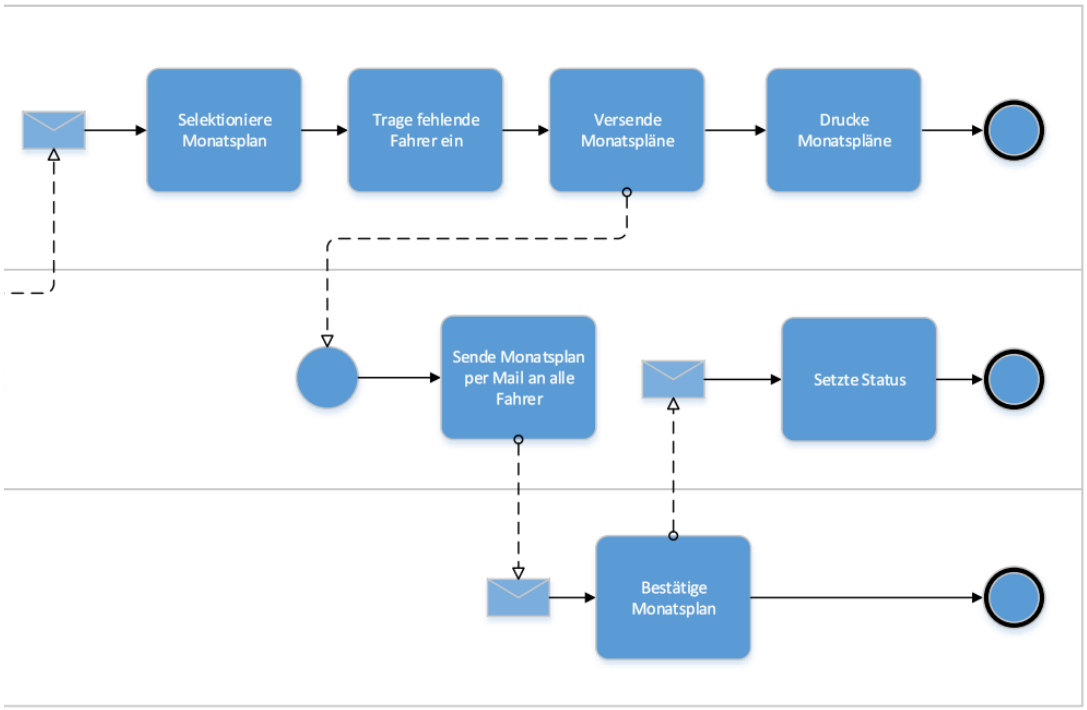


Abbildung 10: BPM Monatsplan Teil2

Abbildung 10: Die Geschäftsleitung wird den Monatsplan vor sich haben und sieht, zu welchen Schichten noch keine Fahrer zugeteilt wurden. Die Aufgabe „Trage fehlende Fahrer ein“ sieht dabei so aus, dass man einzelne Fahrer telefonisch sucht. Dies sind meist bekannte Fahrer, die gerne noch freiwillig an weiteren Schichten arbeiten. Beim Fahrer wird oft noch eine Notiz vermerkt, wann er zusätzlich zum Dauereinsatz noch bereit ist Einsätze zu leisten. Vom iTIXI System aus wird dann der Monatsplan per Mail an alle Fahrer mit Mailadressen versendet. Der Monatsplan wird zusätzlich noch ausgedruckt und per Brief an alle Fahrer gesendet, insbesondere an Fahrer ohne Mailadressen. Erhält der Fahrer den Monatsplan, bestätigt er dies. Es ist vorstellbar, dies über ein Bestätigungssystem zu realisieren, indem der Fahrer z.B. auf einen generischen Hyperlink im Mail klickt und den Erhalt somit bestätigt.

6.4.2 Tagesdisposition

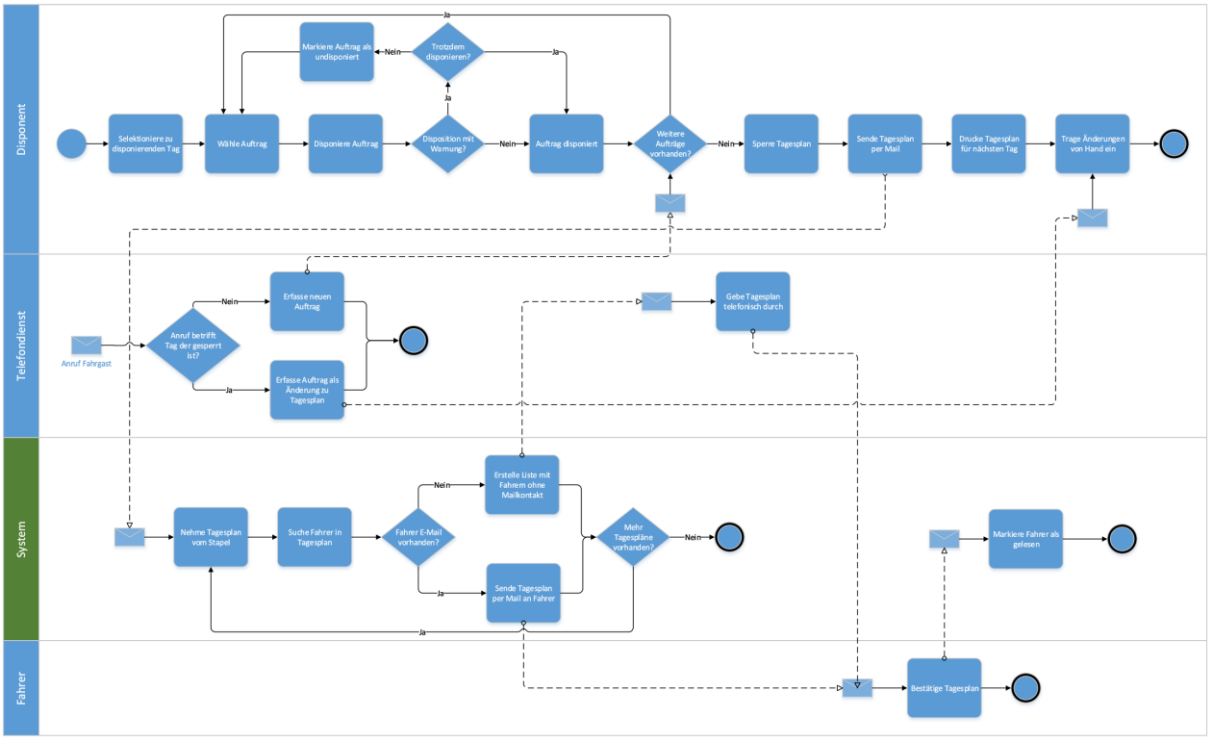


Abbildung 11: BPM Tagesdisposition Übersicht

Bei der Tagesdisposition kommt es zu intensiveren Abläufen zwischen den Mitarbeitern im Telefondienst und Disposition, sowie dem iTIXI System. Es geht darum, dass Fahraufträge in den nachfolgenden Tagen auf Fahrer und Fahrzeuge eingeteilt werden können.

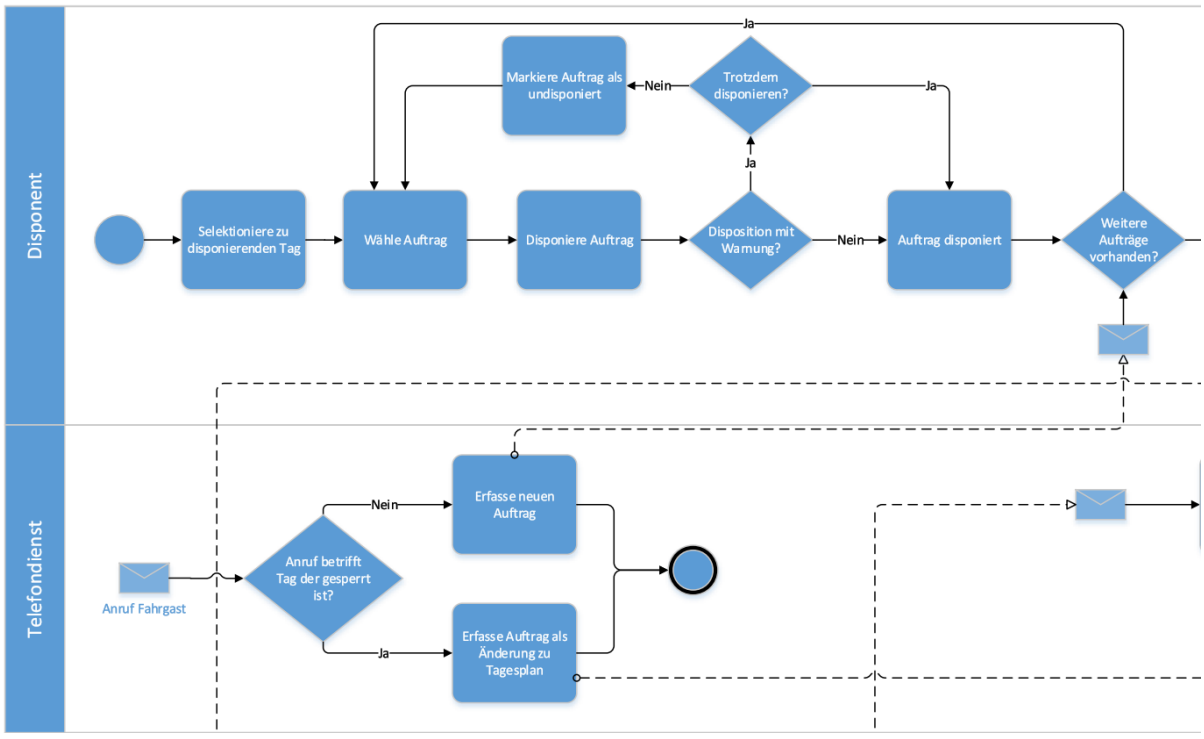


Abbildung 12: BPM Tagesdisposition Teil 1

Ein Disponent wählt den zu disponierenden Tag aus und wählt Aufträge aus, die noch nicht eingeteilt wurden. Diese Aufträge werden disponiert, d.h. an einem Fahrer und Fahrzeug zugewiesen. Erfolgt dies mit einer Warnung, dann ist ein Fahrer oder ein Fahrzeug nicht vereinbar mit diesem Auftrag oder es wird zeitlich knapp diesen noch irgendwo unterzuordnen. Grundsätzlich werden alle Fahraufträge von Gästen angenommen. Ist ein Auftrag nun nicht machbar, wird versucht ein zusätzliches Fahrzeug aufzutreiben. Sind alle Fahrzeuge im Betrieb bereits eingeteilt, werden je nach Möglichkeit private Fahrzeuge von Fahrern verwendet. Nur ganz selten muss ein Auftrag abgelehnt werden, der Fahrgast wird dann informiert. Soweit aber wird die Disposition entweder mit oder ohne Warnung fortgesetzt und ein Auftrag zugeteilt, solange bis keine mehr vorhanden sind. Die wäre der manuelle Prozess der Disposition, der soweit durch eine Fahroptimierung ersetzt wird. Trotzdem, die Hoheit muss für die Disponenten gewährleistet alle Aufträge von Hand zu ändern und einzuteilen. Sind alle Aufträge zugeteilt und ist die Tagesplanung für den nächsten Tag fertig, wird der Tagesplan gesperrt. Wie in der Ist-Aufnahme beschrieben, werden nun die Informationsblätter vom Tagesplan den Fahrern zugesendet und zusätzlich ausgedruckt. Änderungen, die jetzt noch eintreffen, werden nur noch per Hand von den Disponenten eingetragen (Abbildung 13), der Tagesplan im System ist bis zum nächsten Tag gesperrt. Kommen also beim Telefondienst neue Aufträge oder Änderungen zu bestehende Aufträgen herein, gibt es zwei Fälle zu beachten (Abbildung 12):

Ist der dazugehörige Tagesplan bereits gesperrt, wird der Auftrag als Änderung zum Tagesplan eingetragen, die Disponentin trägt die Änderungen von Hand ein.

Ist der Tagesplan nicht gesperrt, wird ein Auftrag normal erfasst und steht dann als weiterer Auftrag zur Disposition frei.

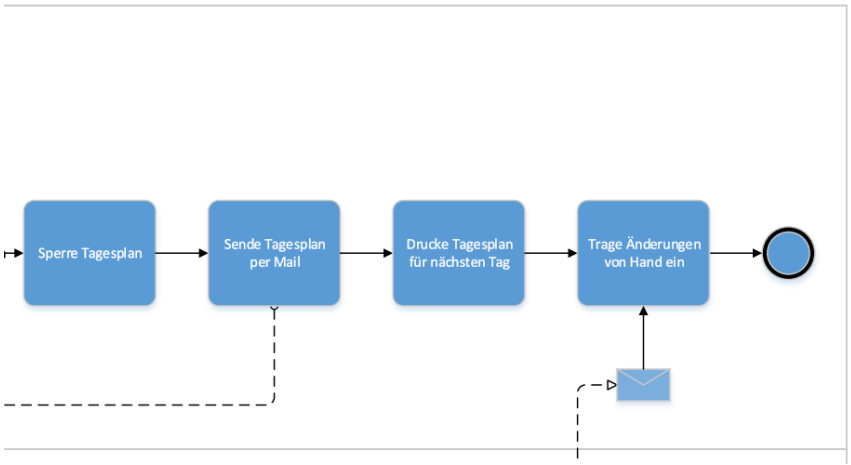


Abbildung 13: BPM Tagesdisposition Teil 2

Wir ein Tagesplan versendet, sucht das iTIXI System alle betroffenen Fahrer und dessen E-Mailadressen heraus und sendet den Tagesplan per Mail (Abbildung 14). Sind Fahrer darunter, die über keine E-Mailadresse verfügen, wird eine Liste mit diesen Fahrern erstellt, die dann von den Disponenten telefonisch über die Aufträge am nächsten Tag informiert werden. Oft werden die Informationen auch per Fax übermittelt.

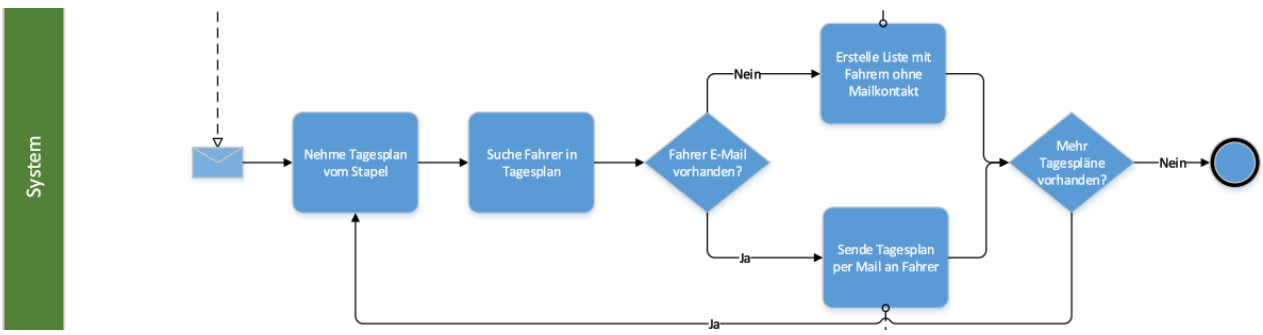


Abbildung 14: BPM Tagesdisposition Teil 3

Die Fahrer bestätigen dann den Erhalt (Abbildung 15), ähnlich wie beim Monatsplan.

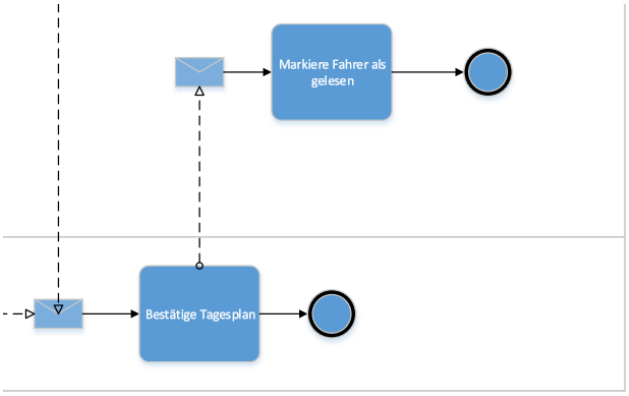


Abbildung 15: BPM Tagesdisposition Teil 4

7 Grundlagen

Nachfolgend einige Theorie zur Fahrtoptimierung und der Verwendung einer Routenberechnung.

In den Kapiteln zur Implementierung wird auf diese Themen verwiesen.

7.1 Vehicle Routing Problem

Fahrtoptimierung

Für die Dispositionssoftware soll eine algorithmische Fahrtoptimierung ermöglichen optimale Fahrten nach vorgegebenen Kriterien zu finden. Fahrten sind hierbei Fahraufträge, bei dem ein Fahrgast zu einer bestimmten Zeit an einem bestimmten Standort mit einem Fahrzeug abgeholt und zu einer (meist vorgegebenen) Zeit an einem anderen Standort abgeladen wird. Je nach Standort, an dem eine Fahrt endet, ermöglichen sich weitere Fahrten die zeitlich durchführbar sind. Diese Folgefahrten beginnen an Standorten, die nun vom letzten Standort aus innert kurzer Zeit oder Distanz erreicht werden können. Um von einem Fahrauftrag zum nächsten Fahrauftrag zu gelangen ist eine Leerfahrt ohne Fahrgast nötig. Aus all diesen Fahrten, Folgefahrten und Leerfahrten resultieren endlich viele Kombinationen die ein globales Optimum erreichen können indem z.B. die Anzahl gefahrener Kilometer pro Leerfahrt auf ein Minimum reduziert wird. Eine Fahrtoptimierung für die TIXI Betriebe bedeutet also, alle Aufträge für ein Fahrzeug so aneinander zu reihen, dass am Schluss möglichst wenig Leerfahrt-Kilometer gefahren werden. Eine andere Möglichkeit wäre, Aufträge zeitlich möglichst kurz aneinander zu reihen damit ein Fahrzeug möglichst viele Aufträge abarbeiten kann und insgesamt die Anzahl benötigter Fahrer und Fahrzeuge reduziert wird. Da bei TIXI-Betrieben aber in erster Linie nicht die Anzahl Fahrer Kosten verursachen, lassen sich mehr Einsparrungen vornehmen in dem möglichst kurze Distanzen gefahren werden.

Permutationen

Damit ein globales Optimum erreicht werden kann, müssten also alle möglichen Kombinationen gefunden werden, um daraus die beste Konfiguration zu wählen. Schnell wird klar, dass die Anzahl möglicher Kombinationen mit der Anzahl Fahraufträgen rasant zunehmen kann. Da die Fahrten in einer bestimmten Reihenfolge durchgeführt werden, entspricht dies einem gerichteten Graphen in dem die Fahraufträge als Knoten und die Leerfahrten zwischen den Fahraufträgen durch Kanten repräsentiert sind. Um eine optimale Konfiguration zu bestimmen würde ein naiver Algorithmus alle möglichen Permutationen von Fahrten ausprobieren und die Beste auswählen. Die Zahl der möglichen Fahrten nimmt hierbei faktoriell mit der Anzahl Fahraufträgen zu, ein Problem welches nicht in polynomieller Zeit lösbar ist.

VRP

Dieses Problem der Fahrwegsoptimierung ist soweit bekannt und gehört zur Gruppe der *Vehicle Routing Problems* VRP welches 1959 (durch [1]) schon auf das NP-vollständige *Traveling Salesman Problem* (TSP) abgebildet wurde und als NP-Schwer gilt. Eine noch genauere Definition für die Fahrtoptimierung mit zeitlichen Abhängigkeiten zwischen den Knoten und dem Einsatz mehrerer Fahrzeuge wird bei *Pick-up and Delivery Vehicle Routing Problem* und *Vehicle Routing Problem with Time Windows* beschrieben, einer Untergruppe der Vehicle Routing Problems welche als *One-To-One Pickup and Delivery Problems*[5] (durch [2]) beschrieben werden.

DARP

Gerade bei den TIXI Organisationen kommen noch weitere Faktoren zu einer möglichen Fahrt hinzu, nämlich unterschiedliche Fahrzeuge die z.B. rollstuhltauglich sind und für bestimmte Fahrgäste ausgelegt wurden. Zusätzlich können aus Fahraufträgen mit gleichem Start oder Ziel sogenannte Sammelaufträge entstehen, bei dem mehrere Fahrgäste auf einmal transportiert und entweder von einem gemeinsamen Start die Ziel-Standorte nacheinander erreicht werden oder alle Fahrgäste bei ihrem Start-Standort abgeholt und gemeinsam zu einem Ziel-Standort gebracht werden.
Man möchte aus einer beschränkten Anzahl von Fahrzeugen möglichst viele Fahrgäste bedienen, dabei aber möglichst wenige Unannehmlichkeiten für die Mitfahrer verursachen. Exemplarische Problemstellungen zum Transport von älteren und

[5] PDP: one-to-one Pickup and Delivery Problems

behinderten Personen im öffentlichen Wesen wurden allgemein unter dem Dial-a-Ride Problem (DARP[6]) veröffentlicht (durch [3]) und spezifiziert unser Problem für die Fahrtoptimierung der TIXI Betriebe recht genau.

7.1.1 DARP

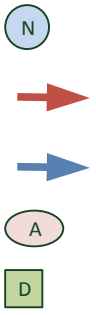
Exemplarisch wird das Dial-a-Ride Problem für viele Systeme im öffentlichen Transportbereich verwendet. Anbei gehen wir kurz auf die Einzelheiten des Problems ein, welche uns für die algorithmische Lösung der Fahrtoptimierung Grundlagen gaben. Hierbei sei erwähnt, dass wir aus zeitlichen Gründen keine exakte Optimierung vornehmen konnten und es bei der Implementation nur zu einer angenäherten Lösung kam.

Eine Lösung mit Sammelaufträgen musste völlig ausgelassen werden, da hier die Komplexität und der Umfang für unsere Arbeit zu gross war und Sammelaufträge in der jetzigen Form sehr auf menschliche Faktoren beruhen – die Disponenten weisen Fahrgäste mit anderen Fahrgästen zu die sich gut kennen und auch gerne miteinander fahren etc. Eine manuelle Zuweisung von Sammelaufträgen durch die Disponenten wird in jedem Fall notwendig sein und kann in dieser Form nicht wie gewünscht von einem Optimierungsalgorithmus gelöst werden.

7.1.1.1 Begriffe und Elemente

Einige Begrifflichkeiten werden in dieser Arbeit eingeführt.

Diese entstammen nicht direkt aus Literatur oder offiziellen Problembeschreibungen, sondern dienen lediglich dazu, die Problemlösung aus unserer Sicht zu repräsentieren.



Node = Ein Standort

Ride = Ein Fahrauftrag, bzw. eine Fahrt zwischen zwei Nodes mit einem Fahrgast

EmptyRide = Leerfahrt

RideNode = Ein Fahrauftrag (vereinfacht zusammengesetzt)

Depot = Fahrzeugdepot

RideConfiguration = Eine mögliche Abfolge von Fahraufträgen

Vehicles = Set von Fahrzeugen die zur Verfügung stehen

Drivers = Set von Fahrern die zur Verfügung stehen

Pickup = Start Standort für Fahrauftrag

Delivery = Ziel Standort für Fahrauftrag

RideTime = Zeit die für die Fahrt eines Auftrags benötigt wird

PickupTime = Zeit wann Fahrgast abgeholt wird

DeliveryTime = Zeit wann Fahrgast am Ziel ist

EmptyRideTime = Zeit die für eine Leerfahrt benötigt wird

EmptyRideDistance = LeerfahrtKilometer

BoardingTime = Zusätzliche Zeit die benötigt wird für Auf- Abladen der Fahrgäste.*

*für TIXI Zug wird mit 3min Auf- und 3min Ab-Laden der Fahrgäste gerechnet – Die BoardingTime wäre somit 6min

[6] DARP: Dial-a-Ride Problem

7.1.1.2 TIXI Dial a Ride Problem

Folgende Bedingungen kommen auf für eine Fahrt

- Tür zu Tür Transport älterer und invalider Fahrgäste
- Fahrgäste bestimmen die Pickup- oder DeliveryTime
- Fahrgäste können Begleiter haben welche ebenfalls mitfahren
- Fahrer arbeiten während einer oder mehrerer Schichten am Tag
- Fahrzeuge haben gewisse Kapazitäten für Personen und Rollstühle -> Fahrzeugkategorie
- Jedes Fahrzeug ist einem Depot zugeordnet von wo aus die Schicht beginnt

Ziel: Fahrtkosten Minimierung durch eine möglichst günstige RideConfiguration

Hinzu kommen noch einige Einschränkungen (Constraints) welche erfüllt werden müssen für eine Fahrt:

- Fahrer ist bereit Rollstuhlfahrten durchzuführen
- Fahrer ist bereit eine gewisse Fahrzeugkategorie zu bedienen
- Fahrgast ist bereit eine gewisse Fahrzeugkategorie zu nehmen
- Fahrzeugkategorie bietet genug Platz für Fahrgäste / Rollstühle und Begleiter

7.1.1.3 Konfigurationsbildung

Eine mögliche RideConfiguration könnte nun so zustande kommen:

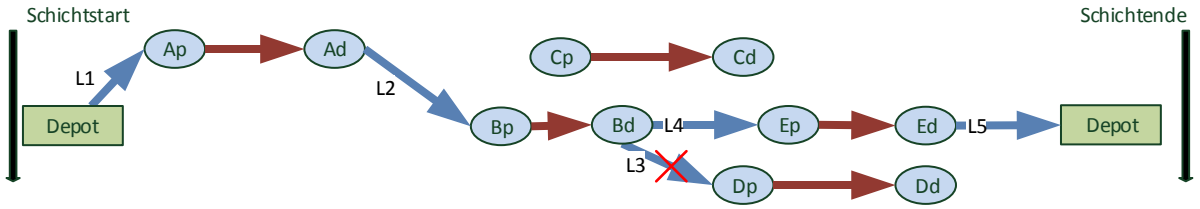


Abbildung 16: Mögliche RideConfiguration für ein Fahrzeug

Fahrzeug aus Vehicles={} fährt vom Depot aus zu einem möglichen Node bei Auftrag **A** (Ap = A pickup)

Diese führt dann den Auftrag **A** aus und fährt den Gast von **Ap** nach **Ad**. Von Node **Ad** aus wird der nächste Auftrag angesteuert, in diesem Fall zum Node **Bp** bei Auftrag **B** - die Leerfahrt **L2** kommt zustande.

Nachdem Auftrag **B** ausgeführt wurde, wären zeitlich gesehen die Aufträge **E** und **D** möglich, **C** scheidet aus.

Da die Leerfahrt **L3** zum Node **Dp** aber zu lange dauert und es nicht möglich ist rechtzeitig zur PickupTime am Standort zu sein, wird Auftrag **E** durchgeführt und Leerfahrt **L4** kommt zustande. Ist kein weiterer Auftrag mehr möglich in der Schichtzeit des Fahrers, wird das Fahrzeug zurück ins Depot gefahren.

Da die Aufträge immer fix vorgegeben sind mit Pickup + Ride + Delivery werden diese als Konstant angesehen – d.h. man benötigt eine gewisse Zeit um den Auftrag zu erfüllen bevor man zum nächsten gehen kann - und nur die EmptyRides zwischen den RideNodes ändern sich für die Konfigurationsmöglichkeiten.

Die Durchführungszeit die für eine RideNode in Anspruch genommen wird entsteht aus der benötigten Zeit für die Fahrt von Pickup nach Delivery + benötigter Zeit für Ein- und Aus-laden der Fahrgäste.

Hinzu kommen Bestimmungen der TIXI Betriebe, die z.B. wünschen, dass der Fahrer 5min vor Auftragsbeginn am Pickup sein sollte um bereit zu sein für den Fahrgast.

Um von einem Delivery die nächsten durchführbaren Aufträge/Pickups zu bestimmen muss folgende Bedingung erfüllt sein:

$$DeliveryTime + EmptyRideTime + 5\,\text{min} < PickupTime$$

Beispiel: Auftrag endet um 14:15 Uhr (DeliveryTime), möglicher nächster Auftrag ist um 15:00 Uhr (PickupTime), die Leerfahrt dorthin benötigt 25min, dann ergibt sich:

14:15 + 25min + 5min = 14:45 => Der Auftrag um 15:00 Uhr wäre durchführbar.

Die Zeit die effektiv für die Durchführung einer RideNode benötigt wird errechnet sich wie folgt:

$$PickupTime + RideTime + BoardingTime$$

Beginnt ein Auftrag um 15:00 Uhr und die dazugehörige Fahrt dauert 30min, so ist die DeliveryTime bei diesem Auftrag um 15:36 Uhr.

Durch die Berechnung, ob zwei RideNodes nacheinander durchgeführt werden können, entsteht eine Adjazenzmatrix der Grösse **N x N** die alle durchführbaren Beziehungen zwischen **N** RideNodes aufzeigt:

	A	B	C	D
A	x	13	35	21
B	x	x	27	19
C	x	x	x	x
D	x	x	x	x

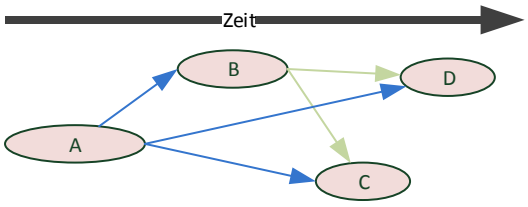


Abbildung 17: Adjazenzmatrix durchführbarer RideNodes

Dank Adjazenzmatrix lassen sich schnell mögliche aufeinanderfolgende RideNodes bestimmen und der Aufbau dieser Matrix aus einem gerichteten Graphen heraus geschieht innerhalb $\boldsymbol{O(n)}$.
Bei durchführbaren Beziehungen wird der zur Optimierung benötigte Wert abgefüllt, in unserem Fall die Kilometeranzahl durch die Leeerfahrt, die z.B. zwischen A -> B = 13km entsteht.
Bei einem Set von 3 möglichen RideNodes = {A,B,C} wären die möglichen Konfigurationen mit mindestens 3 verfügbaren Fahrzeugen bereits wie folgt:

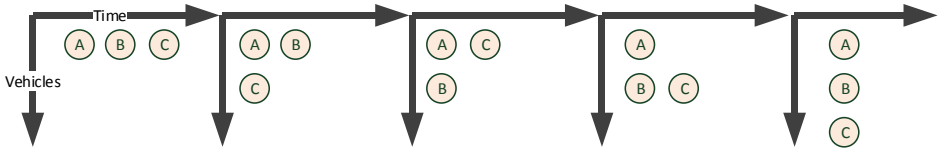


Abbildung 18: Konfigurationsmöglichkeiten bei 3 RideNodes

Bei 4 möglichen RideNodes = {A,B,C,D} und mindestens 4 Fahrzeugen:

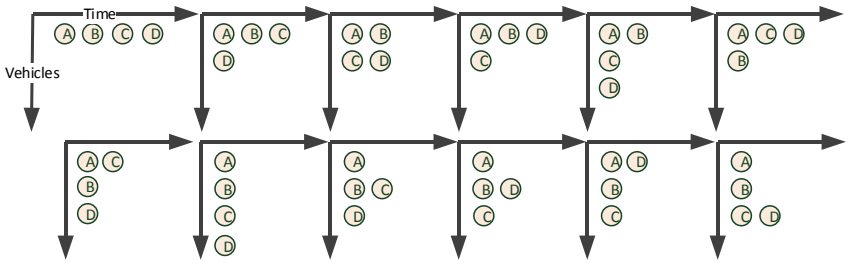


Abbildung 19: Konfigurationsmöglichkeiten bei 4 RideNodes

Mit diesem Preprocessing verringert sich die Anzahl der zu vergleichenden RideNodes für die Konfigurationsbildung. Um aber für ein Optimum alle Permutationen an möglichen Konfigurationen durchzurechnen wird dennoch $\boldsymbol{O}(\boldsymbol{n^2 3^n})$ benötigt wie bei single-vehicle DARP [3] beschrieben.

Es gibt verschiedene Verfahren und exakte Algorithmen für die Lösung von DARP, darunter wird auch unterschieden zwischen static und dynamic DARP (D-DARP), wobei letzteres gerade für einen Abrufbetrieb gedacht wäre. Dies würde sich insofern auszahlen, wenn der TIXI Betrieb in Zukunft auch Spontanfahrten durchführen möchte. Für unsere Arbeit reichte die Zeit nicht aus um sich mit einem exakten Algorithmus zu beschäftigen. Dabei würde man z.B. nur schon zusätzliche Frameworks für Lineare Optimierung benötigen oder Constraint Programming Libraries (CPL) für Constraintprogrammierung, welche meist nur in den Programmiersprachen C/C++ existieren und somit zusätzliche Wrapper Classes für C/C++ auf PHP notwendig wären.

Um dennoch eine machbare Lösung zur Fahrtoptimierung zu erhalten, entwickelten wir vorerst einen einfachen Algorithmus, der möglichst simpel eine machbare Konfiguration erstellt:

- AdjazenzMatrix für alle RideNodes einer Schicht wird erstellt
- Pro verfügbare Fahrzeuge wird eine RideConfiguration erstellt
- Ein Set wird mit allen RideNodes gefüllt und nach PickupTime sortiert
- Eine RideNode aus dem Set wird einer RideConfiguration zugewiesen
- Sequentiell werden alle nacheinander durchführbaren RideNodes der RideConfiguration hinzugefügt, wobei immer die nächste RideNode mit der kürzesten EmptyRideDistance genommen wird

Danach wird zu der entstandenen Konfiguration für jede RideNodeList ein Fahrzeug zugeteilt, falls dieses alle Constraints zu den entsprechenden Fahraufträgen erfüllt. In einem zweiten Schritt werden verfügbare Fahrer einer RideNodeList hinzugefügt wenn ebenfalls alle Constraints zu den Fahrzeugen und Fahraufträgen erfüllt sind. So entsteht eine Konfiguration die zwar machbar ist und einigermassen auf die benötigten Leerfahrtkilometer achtet, aber noch lange kein Optimum an Leerfahrtkilometer hervorbringt.

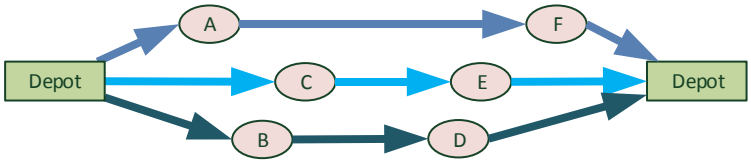


Abbildung 20: Machbare Konfiguration mit 3 Fahrzeugen aus gleichem Depot

In einem weiteren Schritt wurde versucht mehrere verschiedene Konfigurationen zu erzeugen, in dem das ursprüngliche Set von RideNodes geändert wurde. Die ersten RideNodes, nach Anzahl verfügbarer Fahrzeuge, werden untereinander ausgetauscht, damit von allen möglichen Start-RideNodes ausgegangen. Weitere Konfigurationen werden erzeugt, indem das Set von RideNodes beliebige Male zufällig durcheinandergemischt wird bevor daraus RideNodes genommen werden. Je nachdem entstehen nicht durchführbare Konfigurationen, da durch eine schlechte Aneinanderreihung von RideNodes eventuell nicht alle einer RideConfiguration zugeordnet werden können. Von allen durchführbaren Konfigurationen wird dann die Beste gewählt.

In einer Weiterentwicklung aus dem einfachen Algorithmus heraus wurde dann ein Verfahren zum Auffinden von approximativen Lösungen angewendet: Simulated Annealing

7.1.2 Simulated Annealing

Bei der *simulierten Abkühlung* handelt es sich um ein heuristisches Optimierungsverfahren zur Findung eines globalen Minimums. Der Begriff stammt aus der Thermodynamik und lehnt an den Abkühlungsprozess nach dem Erhitzen eines Metalls an. Die langsame Abkühlung nach dem Erhitzen sorgt dafür, dass die Atome genügend Zeit haben sich zu ordnen um stabile Kristalle zu bilden – dieser Zustand wird als Optimal betrachtet und gilt als besonders energiearm. Bei hohen Temperaturen bewegen sich Atome frei und zufällig und ordnen sich entweder in Kristallgebilde ein oder bleiben weiter in Bewegung. Die Wahrscheinlichkeit bei hoher Temperatur ist dabei wesentlicher kleiner, dass sich Atome in stabile Gebilde einordnen und es werden Zwischenergebnisse erreicht die kein Optimum begünstigen.

Wenn die Temperatur hoch und die mittlere Geschwindigkeit der Teilchen entsprechend groß ist, lösen sich viele Teilchen heraus, bis hin zur vollständigen Auflösung des Kristalls. Wenn die Temperatur niedriger ist, wechseln viele Teilchen in einen energetisch vorteilhaften Kristall, aber nur wenige haben die Energie um ihn wieder zu verlassen: der Kristall fängt an zu wachsen (aus [4]).

Bildet man dieses Verfahren auf das Traveling-Salesman-Problem (TSP) ab, bei dem ein globales Minimum die kürzeste Reiseroute ist, dann wäre die zufällige Bewegung von Atomen gleichzusetzen mit der zufälligen Änderung einer Reiseroute. Führt man beliebig viele Änderungen durch, kann sich die Reiseroute willkürlich ändern und die gesamte Strecke der Rundreise kann sich verkürzen oder verlängern. Eine Verkürzung ist immer willkommen, eine Verlängerung wird nicht immer zugelassen, ausser dann, wenn bei hoher Temperatur die „Teilchen“ noch genügend Energie haben das Gebilde wieder zu verlassen. Die Wahrscheinlichkeit, ob die Teilchen genügen Energie haben, wird durch die Boltzmann-Verteilung angegeben:

$$p = e^{-\frac{\Delta E}{T}}$$

Je grösser die Energiedifferenz (beim TSP die Streckenverlängerung) und je kleiner die Temperatur, desto unwahrscheinlicher wird es, dass sich die Änderung günstig auswirkt.

Die Vorgehensweise um eine für uns auf die Leerfahrten optimierte Lösung zum Dial-a-Ride Problem zu erhalten über erwähntes Optimierungsverfahren ist wie folgt.

Es wird zuerst eine machbare Lösung erstellt, ähnlich anhand eines einfachen Algorithmus.

Von dieser machbaren Lösung werden zufällig immer zwei RideNodes miteinander vertauscht.

Falls das Vertauschen möglich und durchführbar ist, wird die neue Gesamtdistanz der entstandenen Konfiguration errechnet. Falls eine bessere Lösung erzielt wurde oder gemäss Boltzmann-Verteilung die Möglichkeit besteht einen günstigen Zustand zu erreichen, wird diese Konfiguration als weitere Ausgangslage verwendet. Dies wird solange durchgeführt bis die Temperatur abgekühlt ist.

7.2 Routing-Machine

7.2.1 Datenmenge der Fahrtenabfrage

Ursprünglich war die Verwendung der Googles Distance Matrix API geplant um Dauer und Distanz von Fahrten zu erhalten. Gerade für die Fahrtoptimierung ist es notwendig diese Daten abzurufen für alle Fahrten und Leerfahrten. Geht man von ca. 300 Fahrten pro Tag und 80 Fahrten pro Schicht aus und man möchte für eine Schicht alle Leerfahrten, bzw. Fahrten zwischen den Aufträgen **N** erhalten, so sind dies:

$$N(N-1) = 6320\ Leerfahrten$$

6320 Leerfahrten die im Schlimmsten Fall berechnet werden müssen um die Fahrten einer Schicht zu Optimieren. Da wir von einer zeitlichen Abfolge ausgehen können und nur die machbaren Leerfahrten zwischen zwei Aufträgen relevant sind, nimmt die Anzahl möglicher Leerfahrten noch einmal ab.

Möchte man diese Mengen mit Googles API nutzen, ist man bereits am Limit:

Googles API Limitations für die freie Nutzung:

2‘500 Abfragen täglich – bei 300 Fahrten am Tag könnten es gut 89‘700 Abfragen für Leerfahrten sein.

100 Abfragen innert 10 Sekunden – würde die Optimierung für eine Schicht um fast 64s verzögern nur für Leerfahrtenabfragen.

Eine Idee durch den Projektpartner war anfänglich, dass man alle vorhandenen Adressen von Fahrgästen und Orte von Interesse zusammen nimmt und daraus alle möglichen Fahrten abfragt über einen längeren Zeitraum hinweg und diese Abfragen dementsprechend persistiert, somit also vorhandene Fahrten von der eigenen Routen-Datenbank bezieht. Für TIXI Zug seien es von allen Aufträgen ca. 80% Daueraufträge die immer an den gleichen Ort gehen. Damit würde man bereits eine grosse Menge vorberechneter Fahrten abdecken.

Bei 2800 Fahrgästen und ca. 300 Orte von Interesse käme man für die Hin- und Rückfahrt auf 1‘680‘000 verschiedene Fahrten. Bei einer Limitierung von 2‘500 Abfragen am Tag käme man zu den gewünschten Informationen erst nach 672 Tagen. Bei Google Maps API für Unternehmen liegen die Limitierungen bei 100‘000 Abfragen täglich und 1000 Abfragen innert 10s.

Die TIXI Zug möchte jedoch nicht in eine Lizenz zur Google Maps API für Unternehmen investieren, die jährlichen Kosten dafür übersteigen die Budgetvorstellungen. Für die Fahrtenabfrage musste eine Alternative her.

7.2.2 OSRM

Mit der Open Source Routing Machine (OSRM)[7] wurde eine Möglichkeit gefunden, die Menge an Abfragen effizient durchzuführen und ohne weitere Kosten zu betreiben. Diese C++ Implementierung berechnet den Shortest Path zwischen zwei Koordinaten, basierend auf den Strassennetzwerk Daten des OpenStreetMap (OSM)[8] Projekts. Für unsere Zwecke wurde eine stabile Version von OSRM auf einem Linux Server kompiliert und mit den OSM Daten der Schweiz betrieben. Da gemäss Anforderungen ein Fallback vorhanden sein muss, wenn u.U. keine Internet-Kartendienst erreichbar sind vom iTIXI Server aus, konnten wir mit einer eigenen Instanz eines Routing-Dienstes durch OSRM gleich zwei Anforderungen erfüllen.

Die Routen Berechnung kommt zustande, indem OSRM zuerst die OSM Files extrahiert um daraus alle möglichen Strasseninformationen zu entnehmen und sogenannte contraction hierarchies[9] bildet um daraus bestmöglich die kürzesten Routen zu berechnen. Die OSM Informationen über einzelne Wege werden durch ein sogenanntes Routing-Profil ersetzt, welche die Geschwindigkeit auf diesen Wegen angibt für die Routen-Berechnung. Dabei kann das Profil angepasst werden mit durchschnittlichen Geschwindigkeiten In- und Ausserorts für Fahrzeuge etc.

Um eine Route zwischen zwei Koordinaten zu finden, wird zuerst die naheliegende Koordinate

(Nearest Node Point) zu einem vorhandenen Strassensegment gesucht, welche für die Berechnung verwendet werden kann.

Um die Dienste der Routing-Machine zu nutzen, stellt OSRM eine HTTP Schnittstelle zur Verfügung welche Abfragen über Nearest Waypoints und Routen ermöglicht. Die Response kann per JSON-Datei erfolgen und ermöglicht eine einfache Implementation für die Routen-Abfrage in unserem Projekt.

7.3 Google Maps

Auch wenn nun OpenStreetMap bereits genutzt wird, für unser Adress-Verwaltungs-Konzept wurde auf Google Maps gesetzt. Die Menge an verfügbaren Adressen ist bei Googles Dienst ungemein grösser und es wird fast immer ein Resultat geliefert. Im Vorfeld wurden bereits Tests zur Adressabfrage durch Martin Jonasse durchgeführt und Googles Service für die Koordinaten-Bestimmung von Kundenadressen für qualitativ besser empfunden.

[7] Open Source Routing Machine OSRM: http://project-osrm.org/
[8] Open Street Map: http://www.openstreetmap.org/
[9] Contraction Hierarchies: Vorberechnete Version eines verbundenen Graphen um die Shortest Path Routing Berechnung zu beschleunigen.

8 Umsetzungskonzept

8.1 Technologien

8.1.1 LAMP

Im Vorfeld dieser Arbeit wurde bereits über die Verwendung eines LAMP[10]-Stacks entschieden.

8.1.1.1 Apache

Apache dient uns als Open Source HTTP Web Server.

Weitere Entscheide zu einer bestimmten Version gibt es nicht.

8.1.1.2 PHP

PHP wird in der Mindestversion 5.5 verwendet. Dies wurde von uns so entschieden, da ab Version 5.5 die kryptografische Hashfunktion BCrypt ein Bestandteil der SPL ist und zusätzlich eine Password Hashing API zur Verfügung gestellt wird, welche wir in unserer Security Implementierung nutzen. Zudem ist OPCache als fester Bestandteil integriert und beschleunigt die Verwendung von PHP über vorkompilierten Bytecode. Mit einer möglichst aktuellen Version von PHP möchte man auch zukünftige Entwicklungen begünstigen.

8.1.1.3 MySQL

MySQL wird in der Mindestversion 5.6 genutzt. Dieser Entscheid kam daher, dass wir für unsere Implementierung eine Volltextsuche innerhalb der Datenbank benötigen. Ein Volltextindex im Zusammenhang mit InnoDB (Doctrine verwendet InnoDB als Subsystem für MySQL) lässt sich erst ab Version 5.6 nutzen[11]. Wiederum möchten wir auch zukünftige Entwicklungen mit einer möglichst aktuellen Version begünstigen.

8.1.2 Frameworks und Libraries

8.1.2.1 Symfony

Wie bereits vor den Arbeiten bestimmt, wurde Symfony als Basis für die Entwicklungen in PHP gewählt. Das Framework legt vieles vor in den Bereichen:

- Konfiguration
- ORM
- Templating
- Security
- Session-Management und Request-Handling
- Internationalisierung I18N
- Validation
- Form-Building
- Caching

Mit Symfony 2.3 wurde eine Long Term Support (LTS) Version herausgegeben. Für den Projektpartner sind die Beliebtheit und Langlebigkeit dieses Frameworks ein Kriterium und sind zukunftsgerichtet um spätere Entwicklungen an diesem Projekt zu ermöglichen und um weitere Entwickler zu finden.

[10] LAMP: Programmkombination aus **L**inux **A**pache **M**ySQL **P**HP/**P**ython
[11] MySQL Fulltext Searches: http://dev.mysql.com/doc/refman/5.1/en/fulltext-search.html

Über sogenannte Bundles lassen sich einfach Erweiterungen und Libraries in Symfony einspielen, wie z.B. Swift Mailer als Mailing Library für PHP.

8.1.2.2 Doctrine2

Doctrine2 erlaubt die objektrelationale Abbildung (engl. ORM). Durch Doctrine werden die einzelnen Klassen und dessen Beziehungen in der Datenbank abgebildet. Doctrine2 ist integriert in Symfony und unterstütz das ORM von Haus aus. Es gilt als eines der meist verbreiteten ORM für PHP.

8.1.2.3 PHPUnit

Mit PHPUnit lassen sich Unit Tests für die PHP-Klassen schreiben. Durch die Tests lässt sich der Code auf seine Funktionstüchtigkeit überprüfen. Automatisierte Tests überprüfen die Resultate zu bestimmten Übergabeparametern. Entsprechen die Resultate nicht den erwarteten Werten, wird in den Tests mit Assertions darauf hingewiesen.

8.1.2.4 phpDocumentor

PHPDoc ermöglicht die generische Dokumentationsbildung aus PHP Source Code heraus. Mit Kommentaren im Source-Code werden für Klassen, Methoden und Attribute weitere Informationen hinzugefügt für die Dokumentation.

8.2 Entwicklungsumgebung

Bereits vor Beginn der Arbeit wurden Entwicklungsumgebungen evaluiert um dem Projektpartner für die vorgängigen Projekt-Iterationen eine Empfehlung abzugeben. PHPStorm[12] von JetBrains ist eine Entwicklungsumgebung, die speziell für Webentwicklung konzipiert wurde und Code-Completion für viele Sprachen bietet wie HTML, CSS, JavaScript. Zudem kommen Erweiterungen hinzu für Symfony, die es erleichtern z.B. Service-Container anzulegen und zu benutzen.

8.2.1 Werkzeuge und Tools

8.2.1.1 Git

Für den Source Code Management und Versionsverwaltung wurde Git verwendet, ein Repository auf GitHub[13] diente während den Entwicklungsarbeiten als Basis für Versionierung und Tracking des Quellcodes. Eine Anbindung in PHPStorm vereinfachte uns das Handling mit Git.

8.2.1.2 Redmine

Für die Zeiterfassung und Planung der Arbeitspakete wurde Redmine eingesetzt. Dieses wurde auf unserem Buildserver bereits vorinstalliert und konnte ohne weiteres genutzt werden. Fortlaufend wurden während einzelnen Iterationen Arbeitspakete erfasst und direkt Arbeitsstunden darauf verbucht.

8.2.1.3 MySQL Workbench

Arbeiten an der MySQL Datenbank, gerade Import und Export von Daten, wurden durch Workbench 6.0 sehr vereinfacht. Einzelne SQL Commands konnten damit schnell getestet werden.

[12] JetBrains PHPStorm IDE für PHP: http://www.jetbrains.com/phpstorm/
[13] GitHub Respositories: https://github.com/

8.2.2 Coding-Standards

8.2.2.1 Namens-Konventionen

Die Sprache der Implementierung ist Englisch. Alle Klassen, Funktionen und Attribute werden sinngemäss Bezeichnet. Mehre Wörter hintereinander erfolgen durch die Camel-Case Notation.

Jede Klasse und Funktion wird mit Kommentarblöcken beschrieben, dies hilft bei der Orientierung und wird für die PHP-Dokumentation verwendet.

8.2.2.2 Code Style

Der Coding-Style wir dem *Zend Coding Standard for PHP* angelehnt. Durch die PHPStorm IDE werden diese Schemas bereit angewendet.

8.3 Qualitätssicherung

8.3.1 Teammeetings / Sitzungsprotokolle

Es wurde versucht wöchentlich ein Team-Meeting durchzuführen mit dem Betreuer, dazu wurden jeweils die erreichten Ziele, Hindernisse und das weitere Vorgehen besprochen. Meetings mit den Projektpartnern wurden je nach Belangen einberufen um wichtige Entscheide durchzuführen. Die Kommunikation per Mail und Telefon wurde stets aufrechterhalten um bei Fragen und weiteren Vorgehen keine Zeit zu verlieren. Sitzungsprotokolle wurden bei allen Meetings geschrieben (enthalten im Appendix unter **Fehler! Verweisquelle onnte nicht gefunden werden.**) um die wichtigsten besprochenen Massnahmen festzuhalten.

8.3.2 Reviews

Dem Projektpartner wurden nach erreichten Funktionen sogenannte Baselines zur Verfügung gestellt. Auf diesen Baselines wurde aus Benutzersicht getestet. Dazu gehörten anfänglich GAP-Analysen, welche die Gaps (Lücken) zwischen den Anforderungen der TIXI Zug und der Implementierungen berücksichtigte, was vor allem die benötigten Datenfelder und Informationen betrifft, welche der Betrieb verwaltet.

Weiter wurden durch den Projektpartner Smoke-Tests durchgeführt, indem alle Funktionen, die zu einer Baseline hinzugekommen sind, mit verschiedenen Ausgangswerten benutzt wurden.

Fehler und Ergänzungen zu den Baselines wurden über das umfangreiche Issue-Tracking-System auf Github erfasst und fortlaufend durch unsere Arbeiten abgedeckt. Änderungen und Abweichungen sind damit gut nachvollziehbar. Alle Issues werden kommentiert und geschlossen, wenn diese behoben sind. Es wurde eine sehr gute Kommunikation zum Projektpartner hergestellt und es konnten viele Probleme sofort bei Auftreten über das Internet behandelt werden und nicht erst bei anstehenden Meetings.

8.3.3 Continous Integration

Unsere Arbeit wurde kontinuierlich ausgeliefert auf einem eigens betriebenen Buildserver (http://tixi.seiout.ch/). Der Buildserver läuft auf einem Debian 7 Linux mit PHP 5.5 und MySQL 5.6 und entspricht einem Stand, der auch beim Deployment anzutreffen sein wird. Dieser überprüft das Repository auf GitHub stetig nach Veränderungen und gleicht diese auf sich selbst ab per Git. Das ganze Projekt in Symfony wird erstellt und für den Produktiv-Betrieb eingerichtet. Zum Build werden alle Units-Tests kontrolliert und die PHP-Dokumentation wird laufend erstellt. Zusätzlich zum eigenen Buildserver, werden die Builds auf Travis-CI erstellt. Dieses bietet für uns eine komfortable Benachrichtigung, falls ein Build oder Unit-Test fehlschlagen sollte. Ausserdem wird somit die Webanwendung auf mehreren Umgebungen erstellt und getestet.

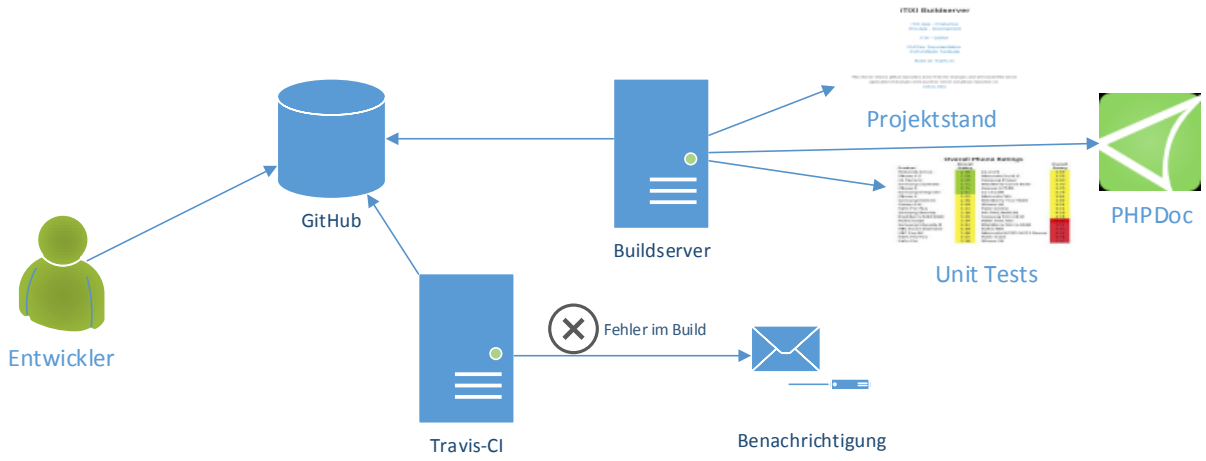


Abbildung 21: Continous Integration Build-Prozess

Wir erhalten somit immer eine aktuelle Version der Web-Anwendung auf dem Buildserver, die wir dem Projektpartner vorlegen können für Tests und Kontrollen. Zudem sehen wir das Laufverhalten auf unterschiedlichen Umgebungen. Diese Vorgehensweise hat sich im Laufe der Entwicklungen als sehr wertvoll erwiesen. Fehler, die auch beim Deployment auftreten können, wurden somit bereits erkannt und behoben.

8.3.4 Unit-Tests

Zu neuen Funktionen innerhalb der Anwendung werden Unit-Tests geschrieben. Für die CRUD-Funktionen werden die Haupt-Entitäten genommen und für dazugehörige Entitäten eine Unit-Test Klasse geschrieben (z.B. Driver zusammen für DriverCategory etc.)

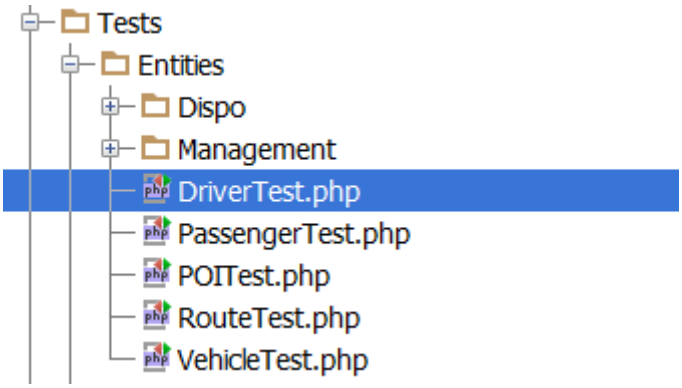


Abbildung 22: Aufbau Testklassen

Besonderen Wert wird auf die Unit-Tests kritischer Funktionen in der Business-Logik gelegt. Durchgängige Tests über alle Schichten sind durch WebTestCase von PHPUnit möglich, diese stellt Request und Response Hilfsklassen bereit um einen Client zu simulieren. Hier werden besonders die API-Funktionen angesprochen und getestet. Tests und Auswertungen liegen im Appendix bei.

9 Design

9.1 Domain

Das Resultat einer der fundamentalsten und zugleich wichtigsten Prozesse in der Konzeption einer Unternehmensanwendung ist der Entscheid, wie die Geschäftslogik auf die Software abgebildet werden soll. Dazu können je nach Komplexität der Geschäftslogik unterschiedliche Ansätze verfolgt werden. In diesem Kapitel sollen generelle Architekturentscheide und weitere Top-Level-Strukturen beschrieben werden. Informationen zur konkreten Implementation finden sich im Kapitel 10 Implementierung.

9.1.1 Architekturentscheid Domain Model / Domain-Driven-Design

Nach Fowler [5] können grundsätzlich die folgenden drei Pattern unterschieden werden, nach denen Geschäftslogik innerhalb einer Software organisiert werden kann: Transaction Script, Domain Model und Table Module (es wird im Rahmen dieser Dokumentation davon ausgegangen, dass die genannten Pattern bekannt sind). Da die Wahl eines dieser Pattern massgeblichen Einfluss auf die Gesamtarchitektur des Systems hat, haben wir dieser Entscheidung einen hohen Stellenwert beigemessen.

Fowler hat den Versuch unternommen, die Pattern als Funktion der Beziehung zwischen der Komplexität der Geschäftslogik und dem Aufwand, der geleistet werden muss, wenn sich die Geschäftslogik ändert oder erweitert, aufzuzeichnen. Die dahinterstehenden Überlegungen hatten einen grossen Einfluss auf unsere Entscheidung.

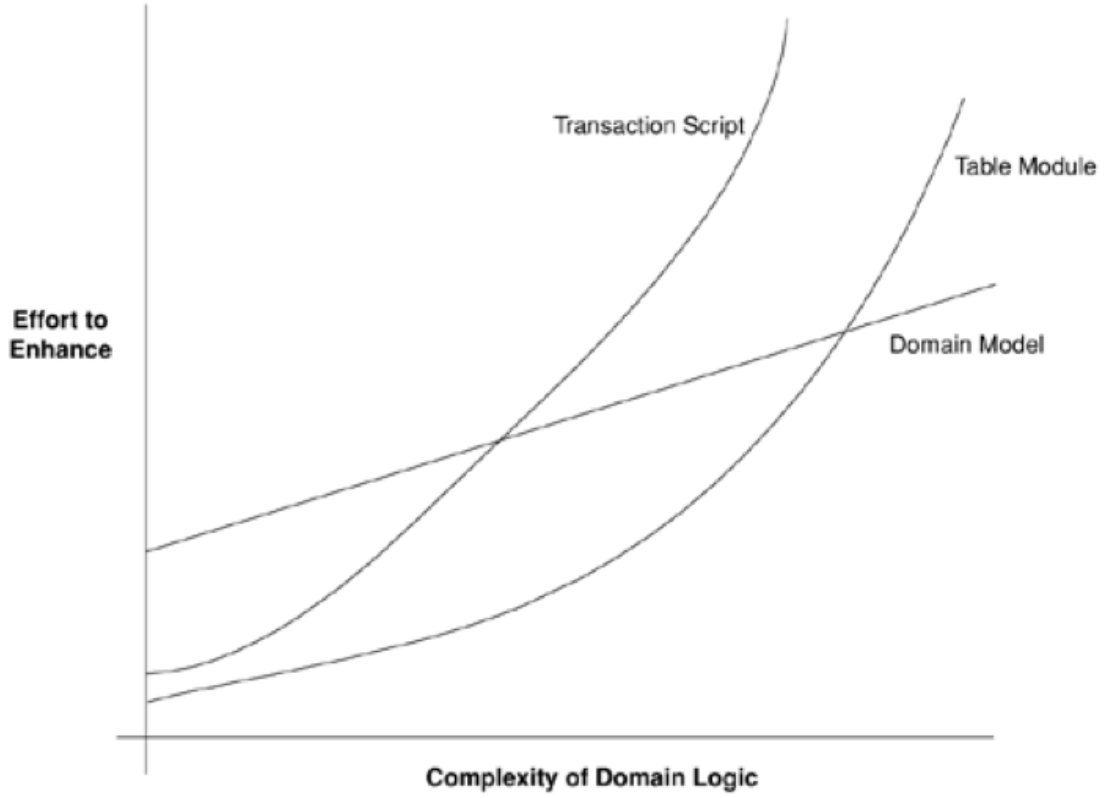


Abbildung 23: Beziehung Komplexität und Entwicklungsaufwand (Fowler [5], S.42)

Aufgrund der in Kapitel 6 beschriebenen funktionalen Anforderungen an das System, sind wir von einer relativ hohen Komplexität ausgegangen, die wir möglichst modular und gleichmässig an die Komponenten verteilen wollten. Zudem sollte das System die Möglichkeit bereithalten, zukünftige Schnittstellen zu ergänzenden

Komponenten mit möglichst geringem Entwicklungsaufwand bereitstellen zu können. Aus diesen Überlegungen und dem Umstand, dass wir als Projektteam sehr stark mit den objektorientierten Ansätzen verwurzelt sind, haben wir uns entscheiden, die Geschäftslogik nach einem Domain Model und die Applikation nach dem Konzept des Domain-Driven-Design's zu organisieren. Nachfolgend soll der Entscheid als Y-Statement zusammengefasst werden:

> *Im Kontext der Architekturentscheidung zur Organisation der Geschäftslogik innerhalb der iTIXI-Gesamtapplikation, konfrontiert mit vielen Anforderungen aus unterschiedlichen Funktionsbereichen, der Forderung nach einem möglichst generischen Ansatz um zukünftigen Anpassung und Erweiterung begegnen zu können und unseren Vorkenntnissen in objektorientierten Programmierparadigmen, haben wir uns für das Domain Model Pattern und gegen das Transaction Script oder Table Module Pattern entschieden, um zu erreichen, dass die Applikation trotz ihrer hohen Komplexität wartbar, änderbar und erweiterbar ist; wir nehmen dafür in Kauf, dass die Entwicklung nicht nach der Rapid-Application-Development (RAD) Methode erfolgen kann und bedingt durch den hohen Abstraktionsgrad einen höheren Einarbeitungsaufwand für zukünftige Entwickler mit sich bringt. Auch nehmen wir in Kauf, dass dadurch zu Beginn einen erheblichen Mehraufwand an Entwicklungsarbeit geleistet werden muss, da bereits bestehende Teile der Applikation neu programmiert werden müssen und dadurch die ursprüngliche Aufgabenstellung des Projektes angepasst werden muss.*

Y-Statement Architekturentscheid Domain Model

9.1.2 Entscheid OR-Mapper

Eine Grundproblematik bleibt die Abbildung des Domain Models auf die Persistenzschicht in einem relationalen Umfeld. Fowler [5] nennt dabei zwei grundlegende Strategien: Active Record und Data Mapper. Da wir es mit einem reichen Domain Model zu tun haben und gängige Techniken wie Vererbung etc. einsetzen, kam für uns der selbstverwaltende Ansatz des Active Record Pattern nicht in Frage und wir entscheiden uns, das Data Mapper Pattern einzusetzen. Da die Implementation dieses Patterns in Eigenregie aufgrund der fehlenden Zeit nicht möglich war und innerhalb der gewählten Technologie und des Frameworks eine gute Lösung existierte, entschieden wir uns, Doctrine als Data Mapper einzusetzen.

> *Im Kontext des Strategientscheids, wie das Domain Model auf die Datenbankschicht in einem relationalen Umfeld abgebildet werden soll, konfrontiert mit einem reichen und komplexen Domain Model, haben wir uns für das Data Mapper Pattern und im speziellen für Doctrine als Implementationslösung eines Drittanbieters und gegen das Active Record Pattern entschieden, um als machbare und performante Lösung das Domain Model Pattern innerhalb eines objektorientierten Umfeldes umsetzen zu können; wir nehmen dafür in Kauf, dass Doctrine eine Abhängigkeit gegenüber eines Drittanbieters darstellt.*

Y-Statement Entscheid OR-Mapper

9.1.3 Domain Model

Nachfolgend lässt sich das durch iterative Prozesse fortan weiterentwickelte Domain Model in einer geeigneten Notation abbilden. Durch die Wahl eines OR-Mappers beruht die Notation auf einer Mischform von UML 2.0 und Elementen aus der Datenbanksicht. Es werden Teilansichten aus dem ganzen Model entnommen, die Gesamtsicht ist für die Dokumentation zu gross, ein hochauflösendes Dokument ist der CD beigelegt (02-Abbildungen/Domain Model.pdf) und taucht als A4 Ausdruck im Appendix 19.3 auf.

Bei der Modellierung wurden aus Platzgründen nicht alle Attribute und Funktionen aufgelistet, die wichtigsten Assoziationen und Attribute sind aufgeführt.

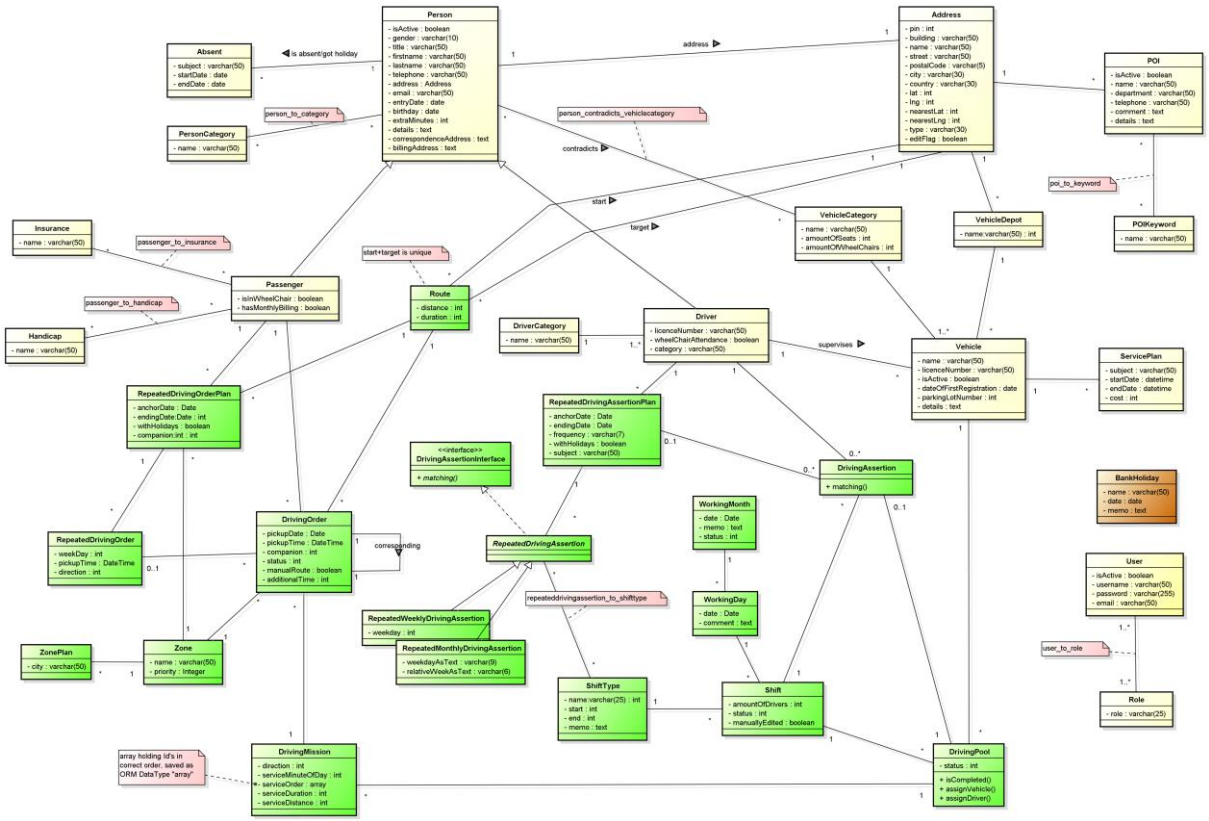


Abbildung 24: Domain Model Gesamtübersicht

Die wichtigsten Aspekte werden nachfolgend in den Teilansichten nur in einer hohen Abstraktion und nicht im Detail erläutert.

9.1.3.1 Elemente zur Stammdatenverwaltung

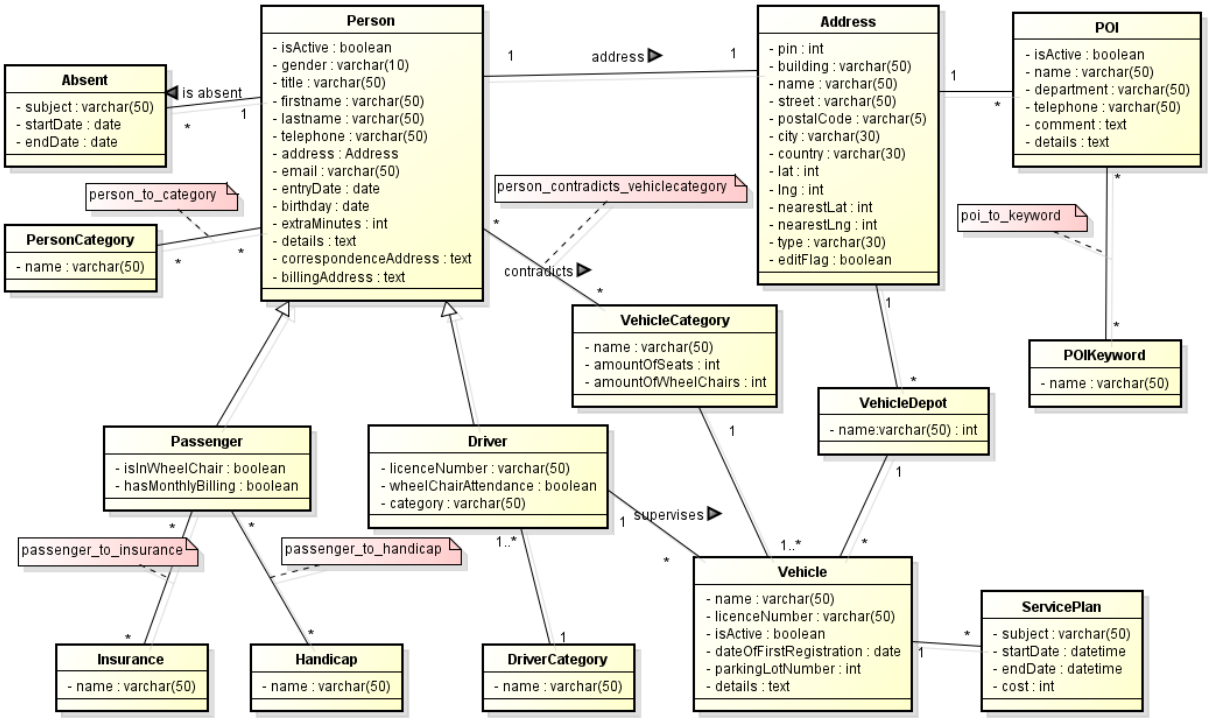


Abbildung 25: Domain Model Stammdatenverwaltung

Um Datenredundanz zu vermeiden, erben Driver und Passenger gemeinsame Attribute der Klasse Person. Drei verschiedene Strategien ermöglichen dabei das Abbilden einer Vererbung per OR-Mapper auf eine relationale Datenbank (Inheritance Mapping):

- Mapped Superclass
- Single Table Inheritance[14]
- Class Table Inheritance[15]

Für die Abbildung von Passenger und Driver wurde ein Single Table Inheritance Mapping gewählt, dies ermöglicht performante Abfragen durch eine Minimierung von JOIN‘s, da alle Daten in einer Tabelle gehalten werden und über ein Discriminator Attribut unterschieden werden.

Auch bei der Address werden alle Attribute aus Performance-Gründen in derselben Klasse gehalten, da wiederum auf Datenbank-Ebene schnelle Abfragen möglich sind und zudem die Volltextsuche einen Index über die Kolumnen der Tabelle erstellen kann. Hier wird bewusst Redundanz eingegangen.

[14] Infos gemäss Fowler auf http://martinfowler.com/eaaCatalog/singleTableInheritance.html
[15] Infos gemäss Fowler auf http://martinfowler.com/eaaCatalog/classTableInheritance.html

9.1.3.2 Elemente zum Fahrauftrag

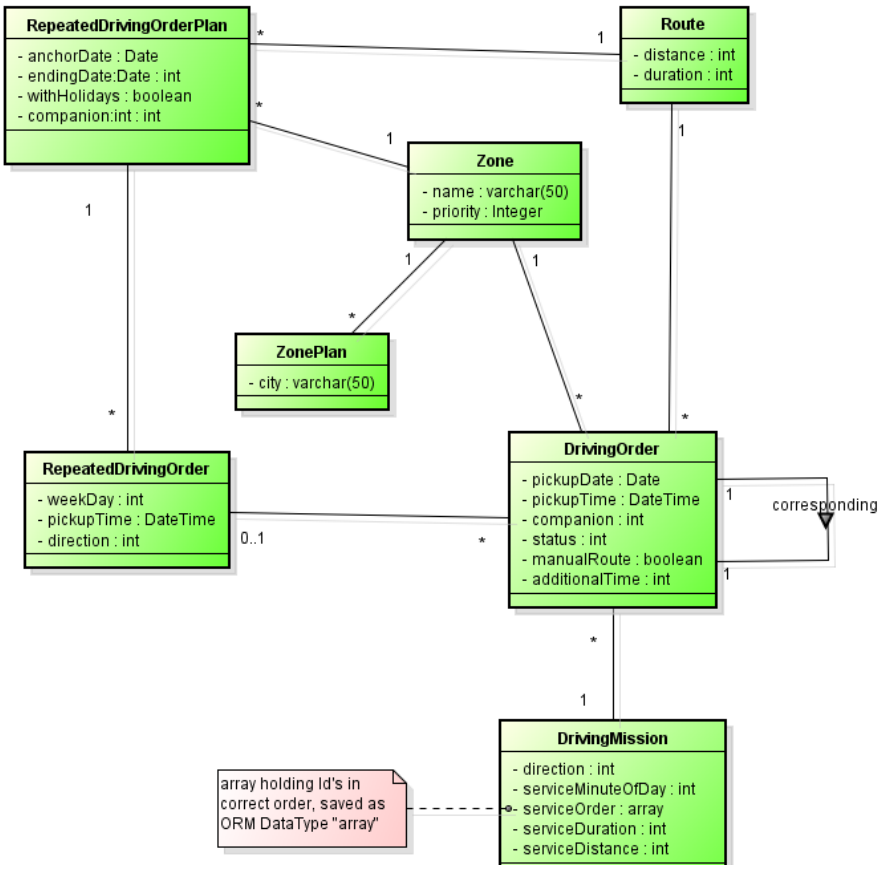


Abbildung 26:Domain Model Fahrauftrag

Die Fahrauftragserfassung beinhaltet Einzel- (DrivingOrder) und wiederkehrende Aufträge/Daueraufträge (RepeatedDrivingOrder). Dieser werden dabei zu einem Fahrgast/Passenger assoziiert (nicht ersichtlich in Abbildung 26). Beim Erstellungsprozess werden aus RepeatedDrivingOrders einzelne DrivingOrders erstellt, die Information aus welchem Dauerauftrag diese stammen wird durch die Assoziation erhalten. Damit eine gemeinsame Bearbeitung von Hin- und Rückfahrten für einen Auftrag möglich ist, wird eine selbstreferenzierte Assoziation auf DrivingOrder erstellt. Kommt ein Fahrauftrag zustande, oder werden mehrere Fahraufträge zu einem Sammelauftrag zugeordnet, erwächst daraus eine DrivingMission.

9.1.3.3 Elemente der Ressourcenplanung

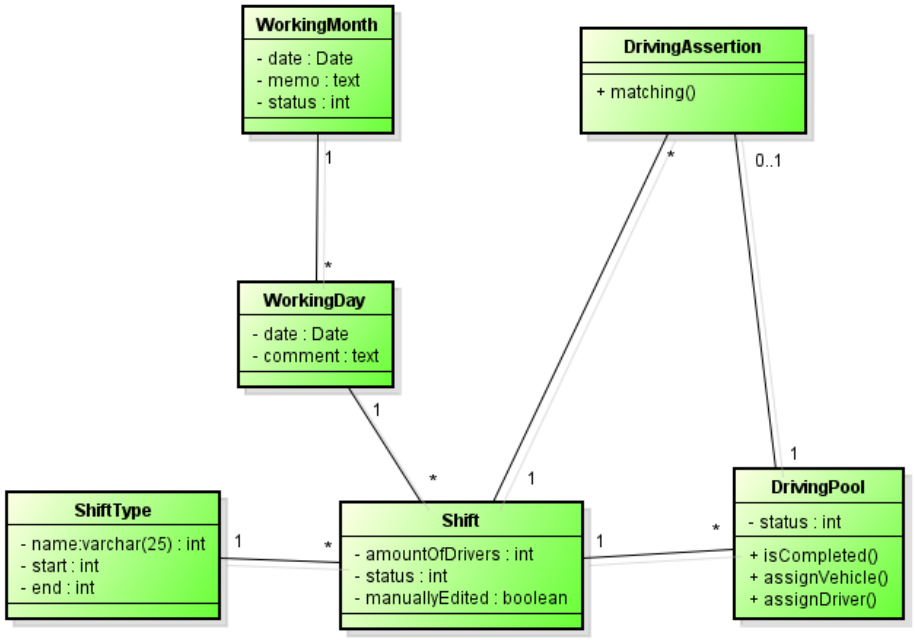


Abbildung 27: Domain Model Ressourcenplanung

Die Abbildung zu einem Produktions- und Monatsplan ist über WorkingMonth, WorkingDay und einzelnen Shifts gegeben, die über einen ShiftType korrespondierende Start- und End-Zeiten zuordnet. Aus benötigten Anzahl Fahrer pro Schicht/Tag/Monat erwächst ein DrivingPool, der für die Zuordnung von bereitgehaltenen Fahrern und benötigten Fahrern dient und beim Zustandekommen einer Fahrt die Assoziation zu einer DrivingMission herstellt.

9.1.3.4 Elemente der Fahrerzuteilung

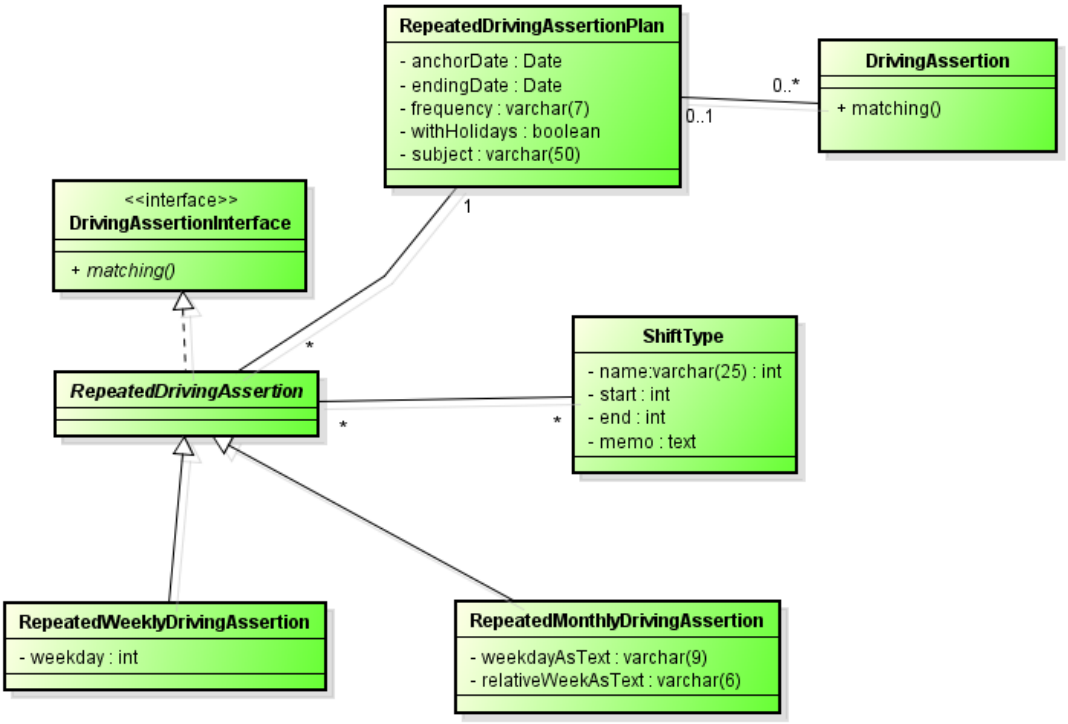


Abbildung 28: Domain Model Fahrerzuteilung

Für die Bereitstellung eines Fahrers wird ein Dauereinsatz (RepeatedDrivingAssertion) erstellt um die möglichen Schichten (ShiftType), Tage (weekday), Wochen (RepeatedWeeklyDrivingAssertion) und Monate (RepeatedMonthrlyDrivingAssertion) abzubilden an denen er Einsätze leisten kann.
Beim Erstellungsprozess werden pro Tag und Schicht DrivingAssertion Objekte erstellt, die für die Zuweisung von Einsätzen im Einsatzplan dient.

10 Implementierung

10.1 Systemübersicht

Die iTIXI-Applikation wurde auf Basis des PHP Frameworks Symfony 2.3 erstellt und nach den Prinzipien des Domain-Driven-Design organisiert. Ziel dieses Abschnittes ist es, einen Überblick über die Komponenten, deren Funktion und Zusammenspiel zu geben und ihre Rolle innerhalb der Domain Driven Design-Konzeptes zu klären. Es wird davon ausgegangen, dass der Leser mit den Symfony-Grundkonzepten vertraut ist.

10.1.1 Gesamtsicht

Um möglichst die Symfony-Standardkonzepte verwenden zu können, haben wir uns entschieden, mit einem sehr dünnen Client zu arbeiten, der vorwiegend Information anzeigt und nur sehr wenig Applikationslogik hält. Die Logik befindet sich demnach überwiegend auf der Serverseite. Das nachfolgende Komponentendiagramm soll einen Überblick über die Struktur der Applikation geben, wobei sich die Bezeichnung der Komponenten (in der Grafik gelb dargestellt) mit der effektiven Dateistruktur des Symfony Source-Ordners deckt. Nicht alle Komponenten sind auch Symfony-Komponenten. Symfony-Komponenten sind daran zu erkennen, dass sie den Bezeichner *Bundle* im Namen tragen. Die blauen Bereiche stellen die logischen Schichten gemäss Domain Driven Design-Terminologie dar. Alles, was nicht als Client bezeichnet ist, ist der Serverseite zuzuordnen.

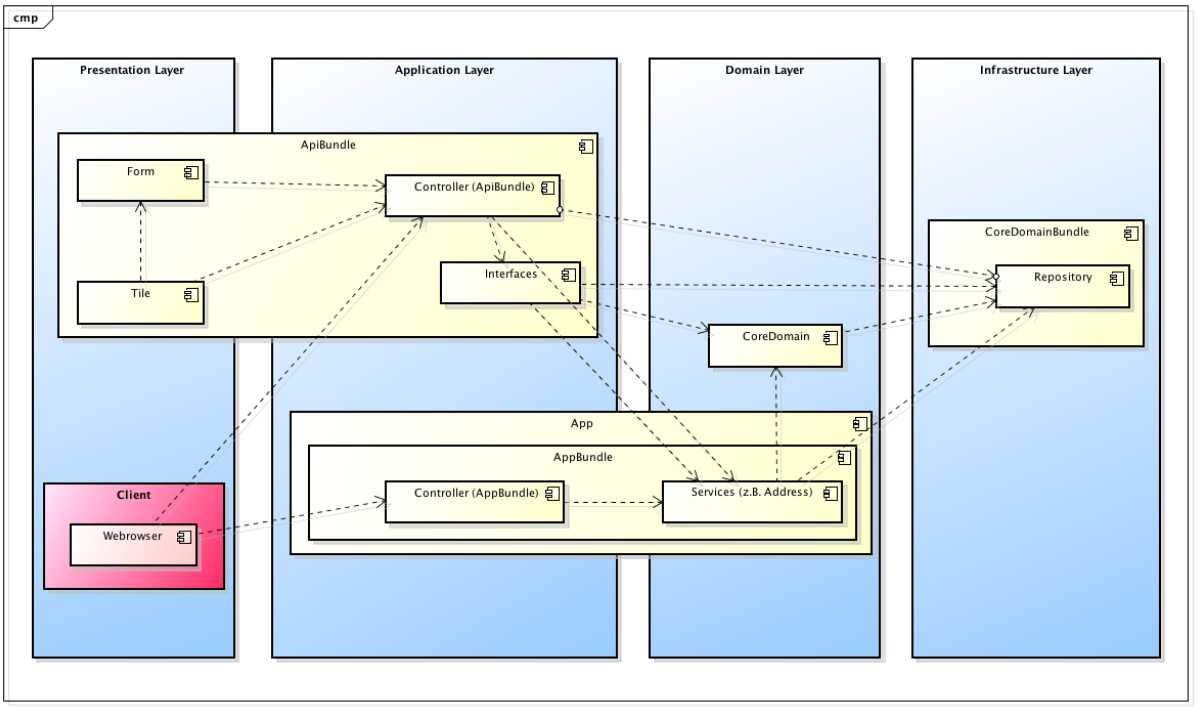


Abbildung 29: Systemgesamtsicht

10.1.2 ApiBundle

Das ApiBundle stellt den Haupteintrittspunkt des Clients in die Applikation dar. Es übernimmt einerseits klassische Frontend-Aufgaben, indem es über die Tile- und Formkomponenten Ein- und Ausgabemasken zur Verfügung stellt und andererseits über die Funktionen der Controller-Klassen, die über HTTP erreichbar sind, das API gegenüber dem Backend bzw. der Domain-Schicht definiert.

10.1.2.1 Controller (ApiBundle)

Paketstruktur:

Controller-Klassen:

`src/Tixi/ApiBundle/Controller`

App-Globale Routing-Konfiguration:

`app/config/routing.yml`

Funktions- und Implementationseckpunkte im Symfony-Konzept:

- Die Controller-Klassen der Controller-Komponente sind klassische Symfony-Controller[16] und erben von `Symfony\Bundle\FrameworkBundle\Controller\Controller`
- Das URL-Routing wird über @Route-Annotations innerhalb der Controller-Klassen gesteuert, was dem System über den folgenden Eintrag in der globalen Routing-Konfiguration bekanntgegeben wird:

```
tixi_api:
      resource: "@TixiApiBundle/Controller/"
      type: annotation
      prefix:   /
```

Beziehung zu anderen Komponenten:

- Client: Die Funktionen der Controller-Klassen sind über HTTP aufrufbar und stellen damit den wichtigsten Eintrittspunkt des Clients (Browser) in die Applikation dar
- ApiBundle\Interfaces: Benutzt die Assembler-Klassen zur Serialisierung und Deserialisierung von Entitäten in Data Transfer Objects und umgekehrt
- ApiBundle\Form: Steuert den Eingabeprozesse über die Form-Komponente
- ApiBundle\Tile: Steuert den Ausgabeprozess über die Tile-Komponente
- CoreDomainBundle\Repository: Lädt und speichert Entitäten aus der Persistenzschicht über die entsprechenden Funktionen der Repository-Klassen

Aufgabe im System:

Die Controller-Komponente ist konzeptionell im Application Layer angesiedelt, der als Mediator zwischen dem Presentation Layer, also der User-Eingabeseite und dem Domain Layer fungiert.

Der Controller hat im Wesentlichen die Aufgabe, entweder Objekte aus dem Domain Layer zu serialisieren und zur Anzeige an eine View, in unserem Konzept einem Tile, zu delegieren, oder Usereingaben, die typischerweise über ein Form übermittelt werden, zu deserialisieren und korrespondierende Entitäten aus dem Domain Layer zu erzeugen oder zu modifizieren.

Das nachfolgende Sequenzdiagramm (Abbildung 30) zeigt das vereinfachte Zusammenspiel der Komponenten, wie sie den für einen Controller typischen Prozess des Anzeigens eines Formulars bedienen:

[16] Symfony Controller: http://symfony.com/doc/current/book/controller.html

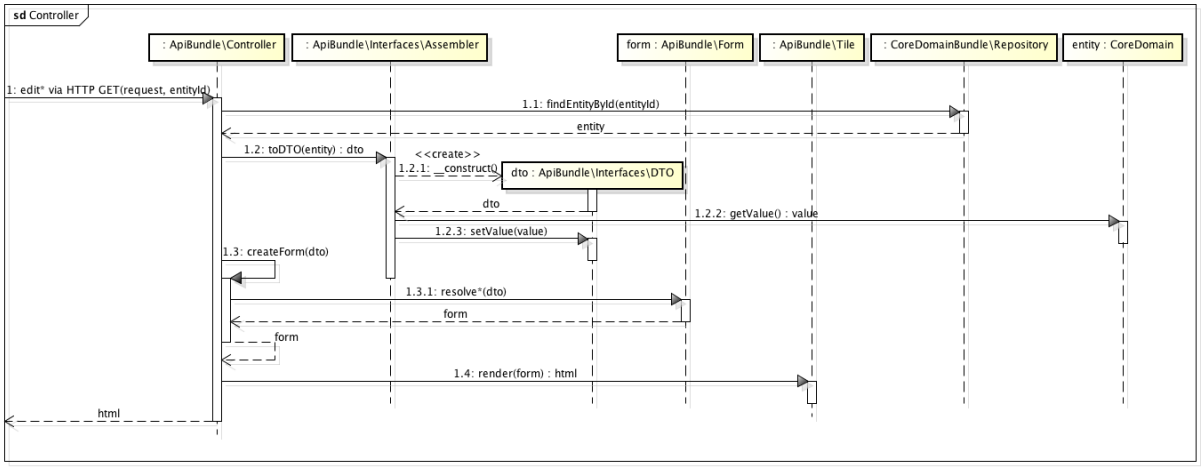


Abbildung 30: Sequenzdiagramm Controllerprozess

Eigenheiten der Implementation:

Der Response, der an den Client übermittelt wird, ist durch die Tile-Komponente generiertes HTML.

10.1.2.2 Interfaces (Assembler und DTO's)

Paketstruktur:

Assembler- und DTO-Klassen:

`src/Tixi/ApiBundle/Interfaces`

Assembler Service-Definitionen:

`src/Tixi/ApiBundle/Resources/config/services.xml`

Funktions- und Implementationseckpunkte im Symfony-Konzept:

- Keine direkte Beziehung zu Elementen im Symfony-Konzept
- Die Assembler-Instanzen werden als Services über den Service-Container[17] zugänglich gemacht

Beziehung zu anderen Komponenten:

- App\AppBundle\div. Services: Assemblers nutzen teilweise Services aus der App-Komponente zur Diensterbringung.
- CoreDomain: Erzeugt oder ändert Entitäten des CoreDomains über entsprechende Factory-Methoden oder Update-Methoden.
- CoreDomainBundle\Repository: Liest oder schreibt Entitäten aus der Persistenzschicht über entsprechende Repository-Funktionen.

Aufgaben im System:

Die serverseitige Kommunikation zwischen dem Presentation Layer und dem Application Layer erfolgt konsequent über sogenannte Data Transfer Objects (DTO's). Die Idee dahinter ist, dass keine direkte Beziehung zwischen den Feldern von Eingabe- bzw. Ausgabemasken der Views und den Eigenschaften der Entitäten des Domain Layers bestehen muss. Vielmehr können z.B. über eine Eingabemaske und die damit verbundene Eingabetransaktion Eigenschaften von verschiedenen Entitäten geändert werden. Jeder Eingabe- und jeder Ausgabetransaktion ist demnach ein entsprechendes DTO zugeordnet.

[17] Symfony Service Container: http://symfony.com/doc/current/book/service_container.html

Das Überführen von Entitäten in DTO's und umgekehrt wird durch sogenannte Assembler übernommen, die als Services über Symfony's Service-Container bereitgestellt werden.
Die Assembler-Klassen und DTO-(pseudo)Klassen werden von der Interface-Komponente bereitgestellt.

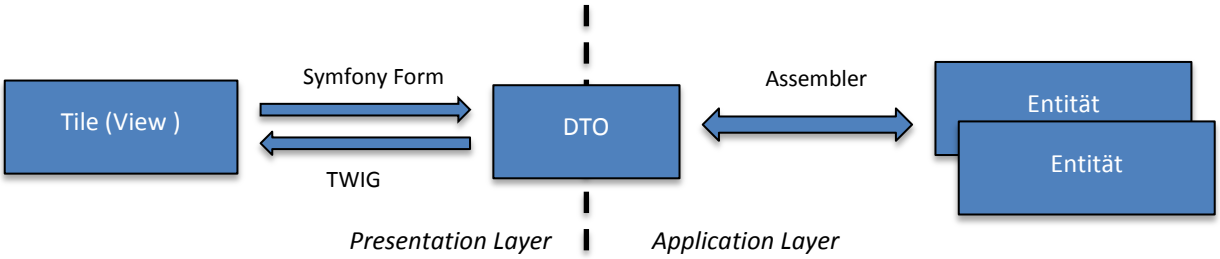


Abbildung 31 Illustration des Data Transfer Object (DTO)-Konzepts

10.1.2.3 Form und Validation

Paketstruktur:

Form-Type-Klassen:

`src/Tixi/ApiBundle/Form/*`

Twitter-Bootstrap Form Template:

`src/Tixi/ApiBundle/Resources/views/Form/fieldsBootstrap.html.twig`

Custom Form Template Config:

`app/config/config.yml`

Funktions- und Implementationseckpunkte im Symfony-Konzept:

- Die FormType-Klassen folgen dem Symfony-Form-Konzept[18] und erben, teilweise indirekt über die CommonAbstractType-Klasse, von `Symfony\Component\Form\AbstractType`
- Die Form-Validators werden in den entsprechenden FormTypes über das constraints-Attribut bekanntgegeben. Zusätzlich wird auch clientseitig validiert (siehe Eigenheiten der Implementation)
- Als Data-Klassen werden DTO's und nicht direkt Entitäten eingesetzt (siehe Eigenheiten der Implementation)
- Das Twitter-Bootstrap Template wird als Default-Template im App-Config eingebunden

Beziehung zu anderen Komponenten:

- ApiBundle\Controller: Die Controller-Komponente steuert den Form-Prozess anhand der Form-Status
- ApiBundle\Tile: Die Form-Komponente wird von der Tile-Komponente zur Darstellung eines Forms innerhalb einer View benutzt.
- ApiBundle\Interfaces: Die Datenklassen der Forms sind DTO's.

Aufgabe im System:

Die Form-Komponente kapselt den Prozess, wie Benutzereingaben in das System gelangen. Diese erfolgen typischerweise über ein HTML-Webformular.

Dazu gehört das Rendern der Formularelemente auf der Clientseiten, das serialisierte Übertragen der Formularwerte zwischen dem Server (Symfony/PHP) und dem Client (Browser) und die Validierung der clientseitigen Eingaben.

[18] Symfony Forms: http://symfony.com/doc/current/book/forms.html

Eigenheiten der Implementation:

DTO's als Data-Klassen:

Um die in den Aufgaben der Interface-Komponente beschriebene Entkoppelung zwischen Entitäten und Eingabemasken umsetzen zu können, werden DTO's als Form-Data-Klassen eingesetzt. Diese werden in den entsprechenden Types durch das Überschreiben der setDefaultOptions()-Funktion bekanntgemacht

(Codebeispiel aus dem DriverType):

```
public function setDefaultOptions(OptionsResolverInterface $resolver) {
      $resolver->setDefaults(array(
          'data_class' => 'Tixi\ApiBundle\Interfaces\DriverRegisterDTO'
      ));
  }
```

Server- und Clientseitige Validierung:

Für die serverseitige Validierung setzen wir die von Symfony angebotenen Form-Validator ein, die wir über das constraints-Attribut direkt über das FormBuilderInterface zum entsprechenden FormType hinzufügen.

Um ein besseres Usability-Verhalten zu erreichen, haben wir uns entschieden, neben der serverseitigen Validierung zusätzlich auch clientseitig zu validieren. Dies erreichen wir, indem wir das HTML5 Validation-Konzept einsetzen.

Symfony setzt bei FormTypes, die als required definiert sind, automatisch das required-Attribut auf dem korrespondierenden HTML-Input-Type auf den Wert "required" (required="required"). Über die HTML5-JavascriptAPI kann dann das Formular über die Methode checkValidity() auf den Status der als required gesetzten Felder abgefragt werden. Standardmässig wird dabei auf "NotBlank" geprüft, also auf nicht leer. Zusätzlich zum Required-Attribut kann noch ein Pattern-Attribut gesetzt werden, das den geforderten Input als RegEx-Pattern definiert.

Die Symfony-Validators setzen meistens auch gleich das Pattern-Attribut auf dem Input Element korrekt. Einzig beim RegEx-Validator musste teilweise das HTML-Pattern manuell überschrieben werden, da sich die RegEx-Formate zwischen den Systemen teilweise unterscheiden.

Die clientseitige Validierung wird per Javascript über das eigens dafür geschriebene FormValidationController-Objekt gemacht. Dieses ist im File *web/bundles/tixiapi/js/FormValidationController.js* spezifiziert und wird standardmässig über das Form-Tile eingebunden.

10.1.2.4 Tile-Konzept

Paketstruktur:

Tile-Klassen:

`src/Tixi/ApiBundle/Tile`

Tile View-Ressourcen:

`src/Tixi/ApiBundle/Resources/views/Tile`

Funktions- und Implementationseckpunkte im Symfony-Konzept:

- Das Tile-Konzept entspricht konzeptionell dem Symfony-Konzept der View
- Der TileRenderer, der für das Rendern der Tile-Instanzen in HTML zuständig ist, ist über den Service *tixi_api.tilerenderer* eingebunden.

Beziehung zu anderen Komponenten:

ApiBundle\Controller: Die Tile-Klassen werden typischerweise in den jeweiligen Controllern instanziiert, verschachtelt und gerendert.

ApiBundle\Form: Die clientseitige HTML-Repräsentation einer Form-Instanz wird über eine entsprechende Tile-Klasse gesteuert.

Aufgabe im System:

Das Tile-System ist konzeptionell dem Presentation Layer zugeordnet und definiert, wie die System-Elemente - meistens Forms - auf der Client-Seite gerendert werden. Die Kernelemente des Tile-Systems sind die Tile-Klassen, die, analog zum Symfony-Form-System, wiederverwendbare und, durch das Prinzip der Selbstreferentialität, verschachtelbare View-Elemente beschreiben.

Eigenheiten der Implementation:

Das Tile-System besteht aus den Tile-Klassen, die alle die abstrakte Klasse AbstractTile erweitern. Zu jeder Instanz einer Tile-Klasse, einem Tile, können über deren add-Methode weitere Tiles hinzugefügt werden. Die zu einem Tile hinzugefügten Tiles werden als Kinder (children) bezeichnet. Aus der Sicht der Kinder wird das Tile, zu dem sie hinzugefügt worden sind, als Eltern-Tile (parent) bezeichnet. Dadurch entsteht eine Baumstruktur von Tiles, wobei ein Tile die Wurzel des Baumes bildet und als root-Element bezeichnet wird.

Jeder Tile-Klasse ist ein Template zugeordnet, dessen Pfad durch das Überschreiben der abstrakten getTemplateName()-Methode des `AbstractTile` bekanntgegeben wird. Das Template kann in einer der von Symfony unterstützen Template-Sprachen verfasst werden. In unserem Fall ist dies TWIG.

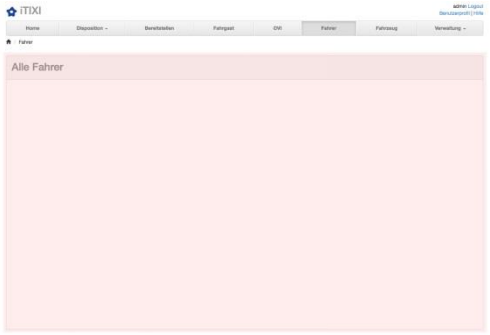
Das root-Element kann dem TileRenderer zum rendern übergeben werden, der den Tile-Baum rekursiv durchläuft und dabei, angefangen mit dem tiefsten, die Elemente gemäss ihrem definierten Template rendert. Es wird mit der Symfony-Template-Engine gerendert, die in der Systemkonfiguration definiert ist (bei iTIXI ist dies wiederum TWIG).

Die View-Parameter, die im jeweiligen Template zur Verfügung stehen sollen, werden durch Überschreiben der getViewParameters()-Methode des `AbstractTile` bekanntgegeben. Die Parameter des Eltern-Tile werden an die Kind-Tiles weitergereicht und stehen dort ebenfalls zur Verfügung. Standardmässig stehen die gerenderten Kind-Elemente dem Eltern-Tile zur Verfügung und können dort entweder über den Tile-Namen, der durch überschrieben der getName-Methode des AbstractTile definiert wird, oder über weitere optionale View-Identifiers, die durch überschreiben der getViewIdentifiers-Methode des AsbtractTile angegeben werden, angesprochen werden.

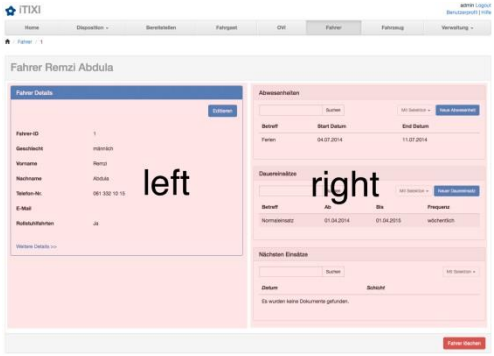
Die Tile-Klassen werden typischerweise in einem Controllern instanziert, verschachtelt und gerendert. Tileweise werden fertig konfigurierte Tiles auch von Services geliefert.

Die wichtigsten Tile's, die im Rahmen dieser Arbeit erstellt wurden sind die folgenden:

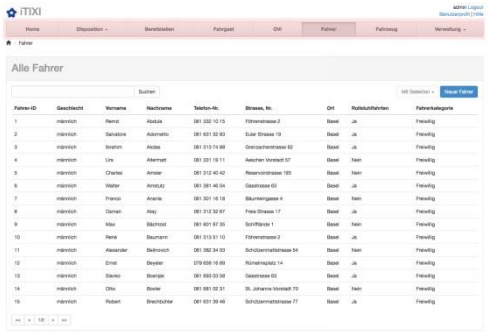
RootPanel



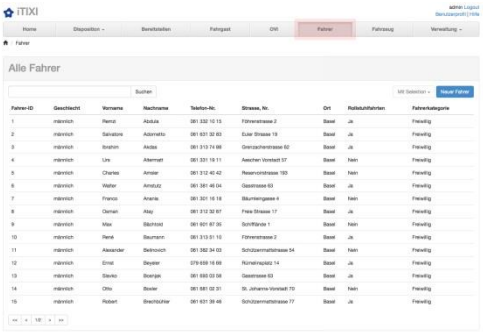
PanelSplitterTile



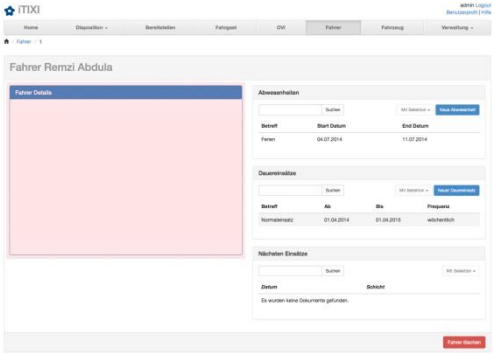
MenuTile



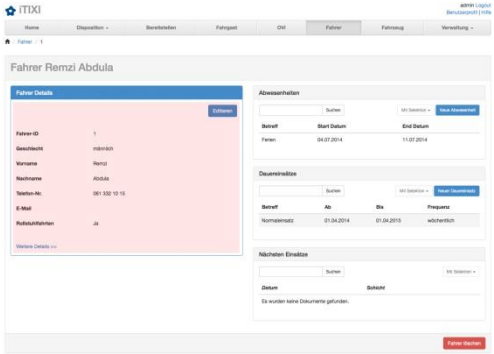
MenuItemTile



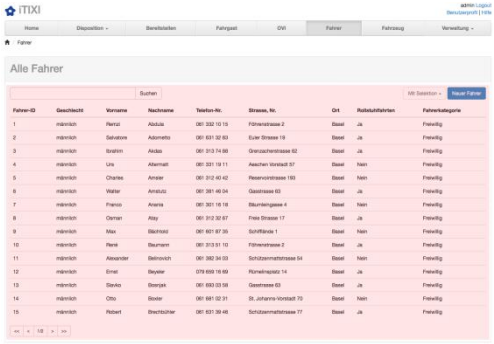
PanelTile



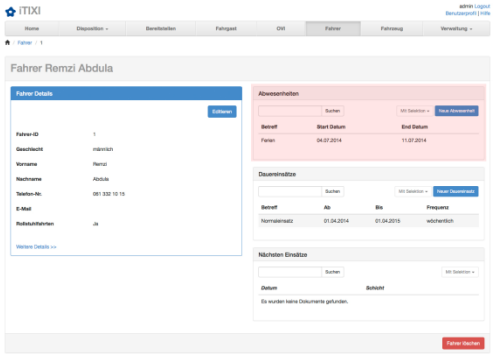
AbstractFormViewTile



DataGridTile



DataGridEmbeddedTile



Für die weiteren Tile-Klassen und die genauen Parametern, die für die Erzeugung der Tile-Instanzen erforderlich sind, sei an dieser Stelle auf den Code bzw. auf die Code-Dokumentation verwiesen.

Das nachfolgende, vereinfachte Code-Beispiel, soll einen Eindruck vermitteln, wie ein typsicherer Tile-Baum innerhalb des Controllers aufgebaut wird:

```
$rootPanel = new RootPanel(...);
$panelSplitter = $rootPanel->add(new PanelSplitterTile(...));
$formPanel = $panelSplitter->addLeft(new PanelTile(...));
$formPanel->add(new DriverRegisterFormViewTile(...));
$absentGridPanel = $panelSplitter->addRight(new PanelTile(...));
$absentGridTile = $dataGridHandler->createEmbeddedDataGridTile(...);
$absentGridPanel->add($absentGridTile);
$rootPanel->add(new PanelDeleteFooterTile(...));
return new Response($tileRenderer->render($rootPanel));
```

10.1.3 App und AppBundle

Grundsätzlich wurde bei der Implementierung darauf geachtet, Businessfunktionalität wo immer möglich direkt auf den zugehörigen Entitäten zu behandeln. Es gibt jedoch auch Operationen, die konzeptionell nicht direkt einer Instanz eines Domain Object's zugeordnet werden können oder für die Diensterbringung noch weitere Entitäten oder Services involviert sind. Diese Operationen wurden zu Services zusammengefasst und stehen über die App-Komponente zur Verfügung. Grundsätzlich werden alle Services durch Interfaces beschrieben, die direkt im Paket *src/Tixi/App* abgelegt sind. Die Implementation der Services unter Symfony findet sich in der AppBundle-Komponente und werden über den Symfony Service-Container zur Verfügung gestellt.

Im Regelfall werden die Services von der ApiBundle-Komponente genutzt, wo sie direkt aus den Controllern oder indirekt über die Assembler-Klassen der Interface-Komponente aufgerufen werden.

Einzelne Service-Funktionen werden jedoch auch direkt vom Client über XMLHttpRequests (AJAX) angesprochen. Die Kommunikation erfolgt dabei über die Controller-Klassen der AppBundle-Komponente.

In diesem Abschnitt sollen die Controller-Komponenten beschrieben werden, sowie die Services aufgelistet werden. Die wichtigsten Services werden in späteren Abschnitten beschrieben.

10.1.3.1 Controller (AppBundle)

Paketstruktur:

Controller-Klassen:

`src/Tixi/App/AppBundle/Controller`

App-Globale Routing-Konfiguration:

`app/config/routing.yml`

Funktions- und Implementationseckpunkte im Symfony-Konzept:

- Die Controller-Klassen sind klassische Symfony-Controller und erben von `Symfony\Bundle\FrameworkBundle\Controller\Controller`
- Das URL-Routing wird über @Route-Annotations innerhalb der Controller-Klassen gesteuert, was dem System über den folgenden Eintrag in der globalen Routing-Konfiguration bekanntgegeben wird:

```
tixi_app:
      resource: "@TixiAppBundle/Controller/"
      type: annotation
      prefix:   /
```

Beziehung zu anderen Komponenten:

- Client: Die Funktionen der Controller-Klassen sind über XMLHttpRequets erreichbar und werden von verschiedenen Javascript-Client-Libraries aufgerufen
- App\AppBundle*diverse Services*: Die Controller-Komponente nutzt zur Verarbeitung der Client-Requests das Funktionsangebot verschiedener Services

Aufgabe im System:

Wie auch die Controller-Komponente des ApiBundles bieten die Controller-Klassen des AppBundles über HTTP ansprechbare Eintrittspunkte in das System an. Sie haben im Wesentlichen die Funktion, Aufrufe an die entsprechenden Services zu dispatchen, das Resultat zu serialisieren und als JSON auszuliefern.

Eigenheiten der Implementation:

Anders als im ApiBundle wird jedoch nicht HTML an den Client zurückgegeben sondern JSON.

10.1.3.2 Übersicht Services

Name	Service-Name	Klasse unter `Tixi\App\AppBundle\*`
Address-Service	tixi_app.addressmanagement	Address\AddressManagementImpl
Dispositions-Service	tixi_app.dispomanagement	Disposition\DispositionManagementImpl
Dokument-Service	tixi_app.documentmanagement	Document\DocumentManagementKnp
Fahrereinsatz-Service	tixi_app.drivingassertionmanagement	Driving\DrivingAssertionManagementImpl
Fahrauftrags-Service	tixi_app.drivingordermanagement	Driving\DrivingOrderManagementImpl
E-Mail-Service	tixi_app.mailservice	Mail\MailServiceSwiftMailer
Fahrtoptimierungs-Service	tixi_app.ridemanagement	Ride\RideManagementImpl
Routenverwaltung-Service	tixi_app.routemanagement	Routing\RouteManagementImpl
Routenabfrage-Service	tixi_app.routingmachine	Routing\RoutingMachineOSRM
Zoneplan-Service	tixi_app.zoneplanmanagement	ZonePlan\ZonePlanManagementImpl

10.1.4 CoreDomain

Die CoreDomain-Komponente stellt das zentrale Element des Domain Layers dar. Darin enthalten sind die Objekte, die im Konzept eines Rich Domain Model die Daten für den Betrieb sowie die eigentliche Businessfunktionalität halten. Die meisten Objekte sind klassische Entities, die neben den Daten auch möglichst viel Logik bereitstellen. Ebenfalls im CoreDomain enthalten sind die als Interface definierten Repositories, die als Implementation des Repository Patterns den Zugriff auf den Entity-Pool kapseln.

Paketstruktur:

Entities und zugehörige Repositoriy-Interfaces:

`src/Tixi/CoreDomain`

Funktions- und Implementationseckpunkte im Symfony-Konzept:

- Die CoreDomain-Komponente hat keine direkte Entsprechung im Symfony-Konzept
- Die Entities sind als Doctrine-Entities konzipiert

Beziehung zu anderen Komponenten:

ApiBundle\Interfaces: Entities werden grundsätzlich im Rahmen von Benutzer-Eingabeprozessen von den Assembler-Klassen der Interface-Komponente anhand der Daten aus den entsprechenden DTO's erzeugt oder geändert oder, ebenfalls von den Assembler Klassen, für Ausgabeprozesse serialisiert.

AppBunle\div. Services: Diverse Services nutzen für die Erbringung ihrer Leistung die Daten und Funktionen der Entities oder Erzeugen oder Ändern Entities anhand von definierten Geschäftsprozessen.

CoreDomainBundle\Repository: Die Repository-Klassen implementieren die im CoreDomain spezifizierten Interfaces in der gewählten Persistenztechnologie, im Rahmen dieser Implementation in Doctrine.

Aufgabe im System:

Wie bereits im Eingangstext besprochen, halten die Objekte der CoreDomain-Komponente alle für den Betrieb relevanten Daten und implementieren soviel der Geschäftslogik, wie möglich. Sie repräsentieren in ihrer Gesamtheit das, was als Rich Domain Model bezeichnet wird.

Die Repositories stellen die nötigen Funktionen zur Verfügung, um Entitäten anhand von bestimmten Eigenschaften aus dem Pool aller Entitäten zu laden und in diesem abzulegen. Dieses Zugriffspattern wird als Repository Pattern bezeichnet und stellt ein sogenanntes "collection-like-interface" für die Entitäten bereit.

Eigenheiten der Implementation:

Es wurde darauf geachtet, die Komponente des CoreDomains soweit als möglich und sinnvoll von Symfony zu entkoppeln und möglichst als Plain Old PHP Objects zu konzipieren, um grundsätzlich das Prinzip der technologieunabhängigen Wiederverwendbarkeit zu gewährleisten. Um das ohnehin schon komplexe Organisationskonzept der Applikation nicht noch weiter zu verkomplizieren, haben wir uns als Kompromiss zwischen formaler DDD-Architektur und Praktikabilität der Lösung darauf geeinigt, die Entitäten über entsprechende Annotations als Doctrine-Entitäten zu konzipieren. Dabei verstehen wir die Information über Annotations als lose Koppelung zu der gewählten Doctrine-Technologie, die im Falle eines Technologiewechsels schnell geändert werden kann. Als einziger Konzeptbruch kann der Einsatz der für Doctrine nötigen Definition von Arrays als Doctrine\Common\Collections\ArrayCollection-Klassen gesehen werden. Wir sind jedoch der Ansicht, dass auch dieser Umstand im Falle eines Technologiewechsels im Persistenz Layer ohne grosse Komplikationen gelöst werden kann, da sich die ArrayCollections genau gleich wie reguläre PHP-Arrays verhalten.

10.1.5 CoreDomainBundle

Das CoreDomainBundle hat einzig die Funktion, die im CoreDomain beschriebenen Repositories in der gewählten Persistenz-Technologie zu implementieren.

Paketstruktur:

Repository-Implementationen:

`src/Tixi/CoreDomainBundle/Repository`

Service-Konfiguration:

`src/Tixi/CoreDomainBundle/Resources/config/services.xml`

Funktions- und Implementationseckpunkte im Symfony-Konzept:

- Die Repositories stehen als Symfony-Services über den Service-Container zur Verfügung

Beziehung zu anderen Komponenten:

CoreDomain: Die Repository-Klassen des CoreDomainBundle implementieren die im CoreDomain spezifizierten Repositories.

ApiBundle\Interfaces, ApiBundle\Controller, AppBundle\div. Services: Die Repositories werden dazu verwendet, um Entitäten anhand bestimmter Eigenschaften aus der Persistenz Schicht zu laden und zu speichern.

Aufgabe im System:

Die konkreten Implementationen der Repositories stellt als Infrastrukturkomponente Zugriff auf die Entitäten sicher, indem sie aus Sicht der Applikation transparent zwischen RAM und MySQL-Datenbank vermittelt.

Eigenheiten der Implementation:

Die konkreten Instanzen der Repository-Implementationen werden im Symfony-Container über die Abstraktion von Aliasen gebunden. Dies ermöglicht theoretisch den Austausch von Doctrine als OR-Mapper rein durch eine Änderung der Systemkonfiguration.

Beispiel dieses Implementationsmerkmal anhand des Vehicle-Repository's:

```
<services>
       <service id="vehicle_repository"
       alias="vehicle_repository.doctrine"></service>


      <service id="vehicle_repository.doctrine"
       class="%vehicle_repository.doctrine.class%" ...>
              ...
      </service>
</services>
```

10.2 iTIXI als Informationssystem

Ein wesentlicher Bestandteil von iTIXI, der im Rahmen dieser Arbeit entwickelt wurde, ist die Implementation der Prozesse und Elemente eines klassischen Informationssystems, die für das Erstellen, Anzeigen, Modifizieren und Löschen (Create, Read, Update, Delete - CRUD) von Daten notwendig sind. Dabei sind in unserem Implementationskonzept grundsätzlich die Komponenten in der Funktion beteiligt, wie sie in der Gesamtsicht beschrieben wurden. Ziel dieses Abschnittes ist es, Besonderheiten der Implementation unter dem Gesichtspunkt von iTIXI als Informationssystem darzulegen.

10.2.1 Menükonzept / Sitemap

Bis auf die Bereiche Disposition, Verwaltung und Bereitstellen, können alle Aktionen klar mit einer Entität verknüpft werden. Die Masterview der jeweiligen Entität bildet den Eintrittspunkt zu allen möglichen Folgeaktionen, die im Zusammenhang mit dieser Entität stehen.

Abbildung 32: Menükonzept

Primärer Funktionspunkt	Sekundärer Funktionspunkt	@Route
Home		/
Disposition		/disposition
	Produktionsplan	/disposition/productionplan
	Monatsplan	disposition/monthlyplan
Bereitstellen		(nicht Teil dieser Arbeit)
Fahrgast		/passengers
OVI		/pois
Fahrer		/drivers
Fahrzeug		/vehicles
Verwaltung		/management
	Benutzer	/management/users
	Zone	/management/zones
	Zonenplan	/management/zoneplans
	Fahrzeugkategorie	/management/vehiclecategories
	Fahrzeugdepot	/management/vehicledepots
	Fahrerkategorie	/management/drivercategories
	Personenkategorie	/management/personcategories
	OVI-Keyword	/management/poikeywords
	Behinderung	/management/handicaps
	Versicherung	/management/insurances
	Schichten	/management/shifttypes
	Feiertage	/management/bankholidays

10.2.1.1 Master-Detail-Konzept

Jeder im Menükonzept aufgeführte Funktionspunkt führt primär auf die Master-View dieser Funktionsgruppe, die bei der Stammdatenverwaltung der jeweiligen Entität entspricht. Ausgehend von dieser Master-View können dann Folgeaktionen auf der Detailstufe der jeweiligen Instanz ausgeführt werden. Dies entweder direkt über das Menü des Master-DataGrids (Abbildung 33) oder indirekt über die Detail-View (Abbildung 34) der Instanz, die durch einen Doppelklick erreicht wird.

Nachfolgend sollen anhand des Beispiels "Fahrer" die wichtigsten Elemente der Views dargestellt werden:

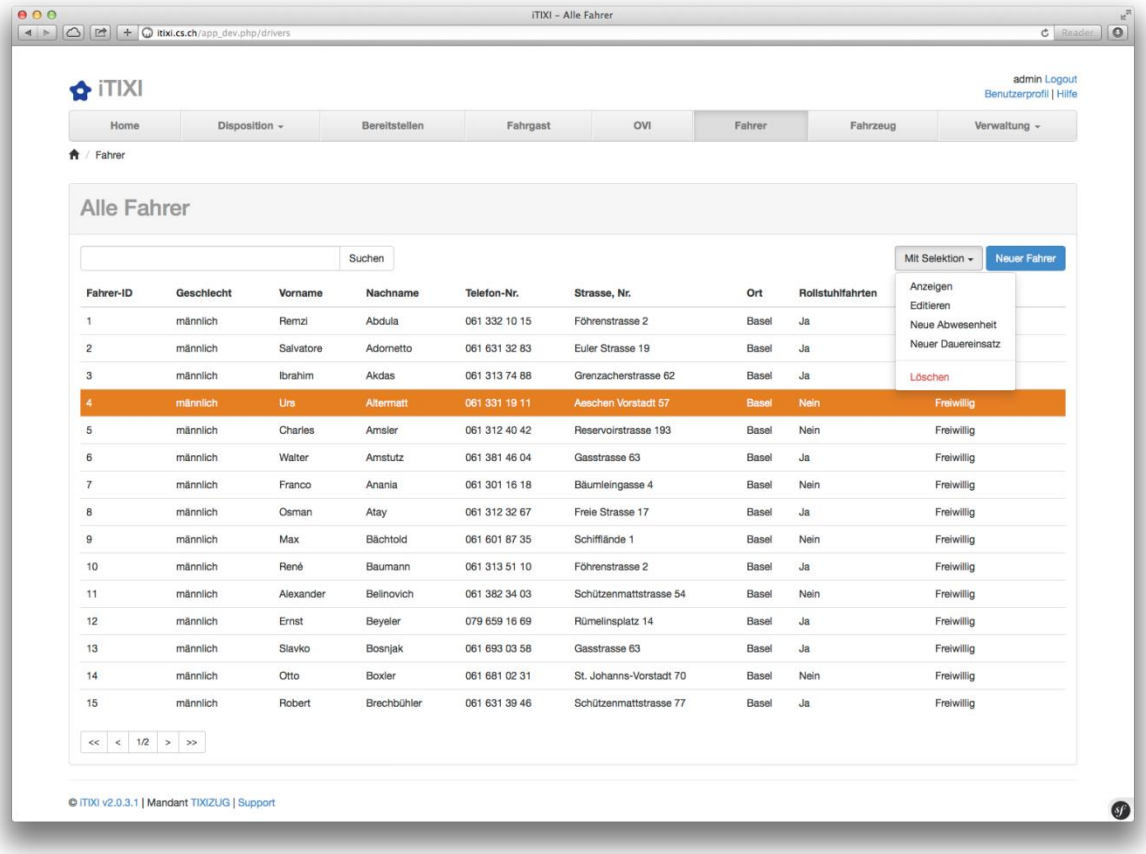


Abbildung 33: View Fahrer – Master-DataGrid

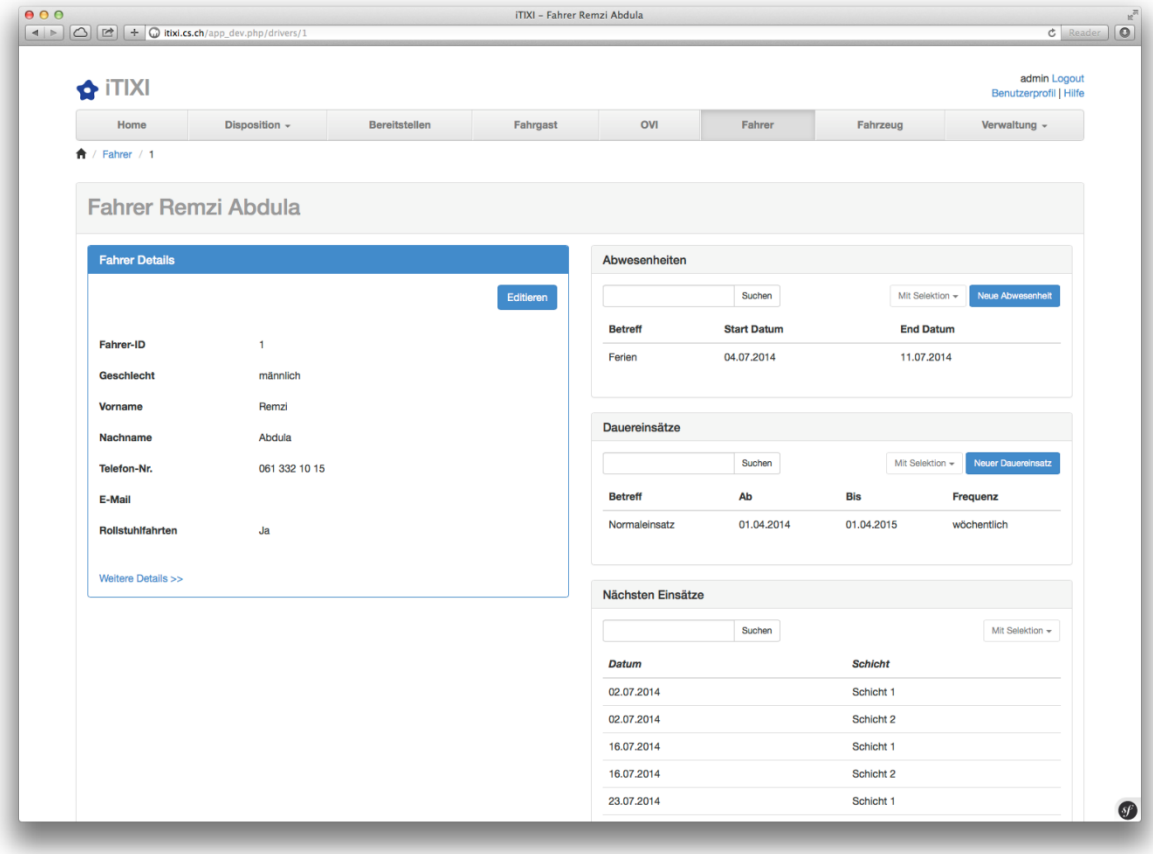


Abbildung 34: View Fahrer – Detail-View

10.2.2 ID-Konzept

Um das Frontend-Testing ermöglichen zu können, wurde ein ID-Konzept für alle informationsrelevanten View-Elemente erarbeitet. Dabei werden grundsätzlich Elemente mit lesendem, schreibenden und ausführenden Charakter unterschieden und mit den Funktionsprefixes *fpr_* (lesend), *fpw_* (schreibend), *fpx_* (ausführend) klassifiziert. Das Konzept im weiteren Sinne lässt sich am besten anhand des Beispiels der Entität Fahrzeug darstellen:

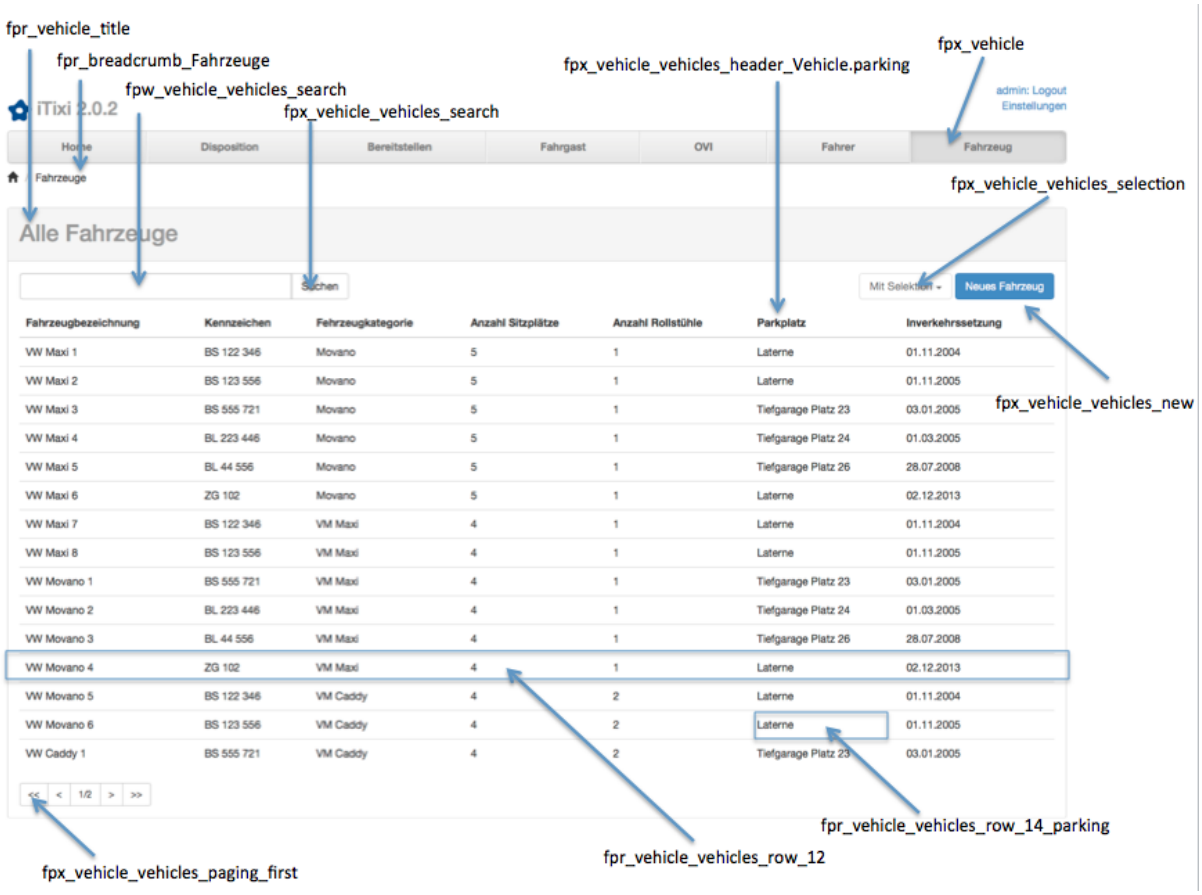


Abbildung 35: ID-Konzept Master-View

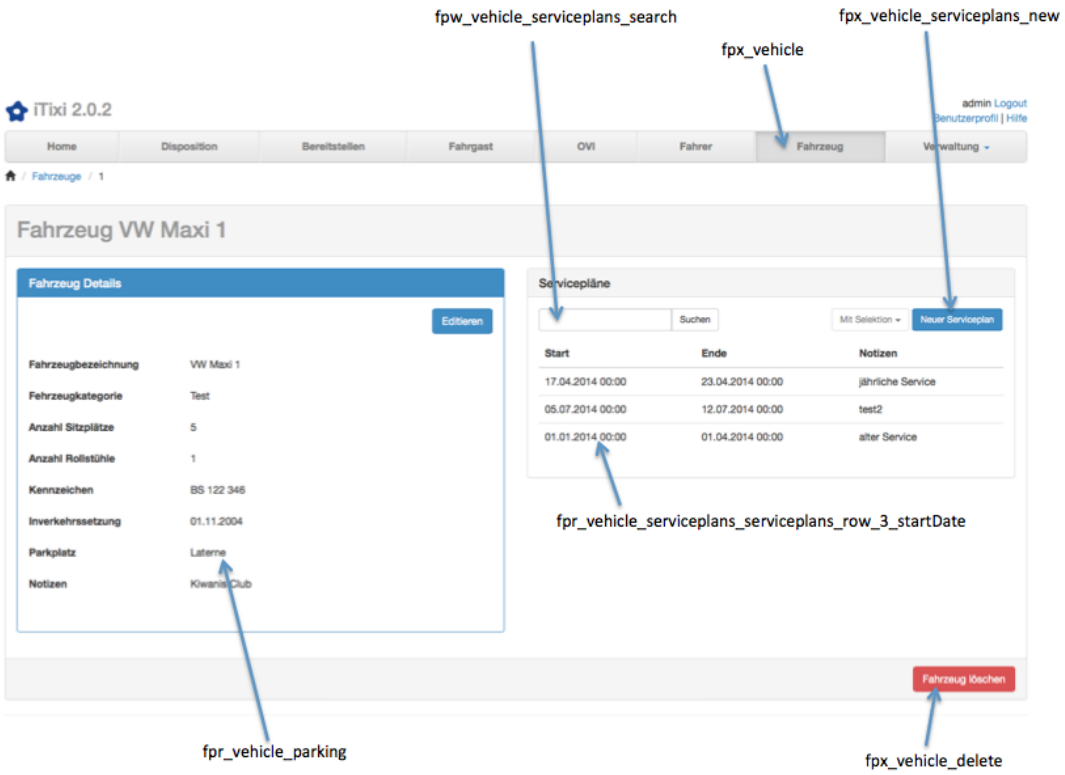


Abbildung 36: ID-Konzept Detail-View get

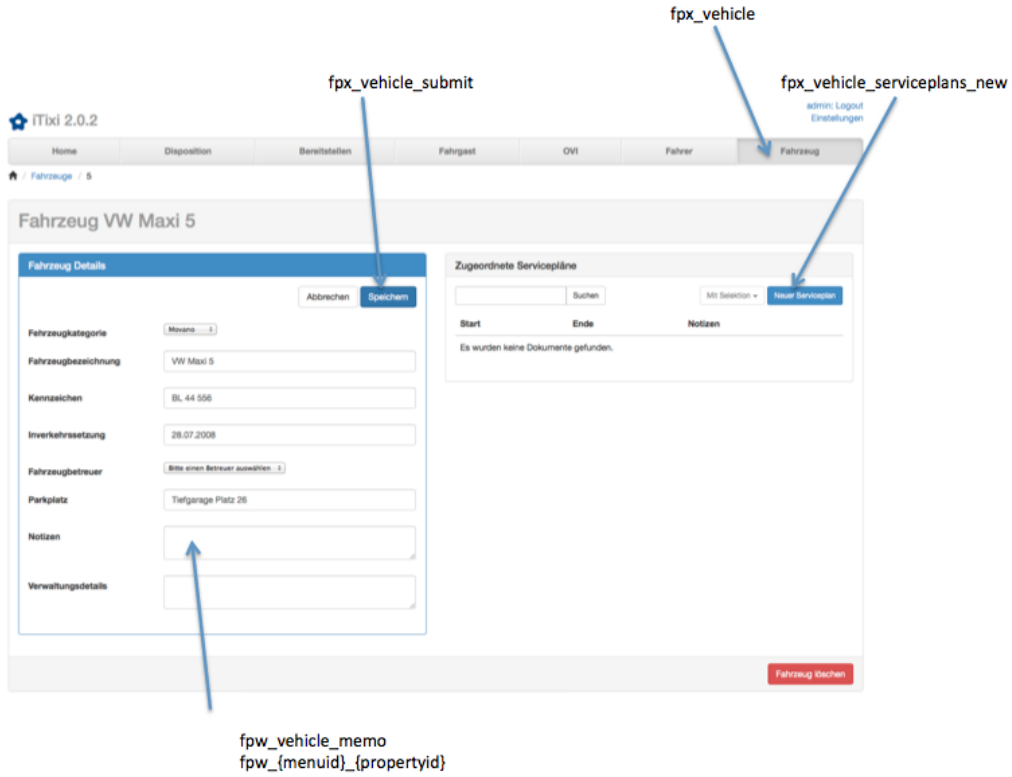


Abbildung 37: ID-Konzept Detail-View edit

10.2.3 Controller-API-Funktionen

Als Eintrittspunkte zum Frontend dienen durch Symfony's Route-Mapping @Route die Controller-Actions zum Aufbau der Response, welche sich für die Entitäten immer gleich aufbaut. Nachfolgend am Beispiel des Fahrers aufgelistet:

getDriversAction() zur Aufbereitung aller Fahrer-Entities für die Master-DataGrid View.

getDriverAction() zur Aufbereitung einzelner Fahrer-Entitiies für die Detail-View.

newDriverAction() zur Erstellung eines neuen Fahrers in der Detail-View Edit.

editDriverAction() zur Bearbeitung eines bestehenden Fahrers in der Detail-View Edit.

```
/**
* Class DriverController
* @Route("/drivers")
*/
class DriverController extends Controller {

        /**
        * @Route("", name="tixiapi_drivers_get")
        */
        public function getDriversAction(...) { ... }

        /**
        * @Route("/{driverId}")
        */
        public function getDriverAction( ... ) { ... }

        /**
        * @Route("/new", name="tixiapi_driver_new")
        */
        public function newDriverAction(Request $request) { ... }

        /**
        * @Route("/{driverId}/delete",name="tixiapi_driver_delete")
        */
        public function deleteDriverAction( ... ) { ... }

        /**
        * @Route("/{driverId}/edit", name="tixiapi_driver_edit")
        */
        public function editDriverAction( ... ) { ... }

}
```

10.2.4 Data Grid

Ein zentrales Element des Informationssystems sind Listen. Hierzu wurde eine eigene Implementierung realisiert für ein geeignetes Pagination grosser Datenmengen.

Diese findet sich im Package: `src\Tixi\ApiBundle\Shared\DataGrid`

> *Aus gesundheitlichen Gründen konnte dieses Kapitel nicht mehr durch CS vervollständigt werden*

10.2.5 Adressverwaltung

Ein weiteres zentrales Konzept für die iTXI-Applikation stellt die Adressverwaltung dar. Im Kontext von iTIXI hat eine Adresse grundsätzlich zwei Funktionen: Einerseits repräsentiert sie eine Postadresse und andererseits beschreibt sie einen Punkt auf der Erde, der eindeutig über geographische Koordinaten definiert ist.

Ziel der Adressverwaltung ist es, eine möglichst redundanzfreie und exakte Adressdatenbank mit Objekten aufzubauen, deren Datenstruktur sowohl aus der strukturierten Postadresse, als auch den zugehörigen Koordinaten besteht.

Um das definierte Ziel zu erreichen ist es notwendig, den Eingabeprozess durch den Benutzer so gut wie möglich zu unterstützen. Dafür wurde ein komplexes Adress-Konzept erarbeitet und implementiert, dessen Komponenten in diesem Abschnitt konzeptionell erklärt werden sollen.

Um das Konzept zu erklären, haben wir uns entschieden, zuerst den Prozess aus Benutzersicht aufzuzeigen und dann die nötigen Systemkomponenten, die dieses Verhalten ermöglichen, aufzuzeigen.

10.2.5.1 Adresseingabemaske / Benutzerinteraktion

Grundsätzlich können die folgenden Benutzerinteraktionen im Zusammenhang mit der Adresseingabemaske identifiziert werden:

Leeres Adressfeld: Der Benutzer trifft auf ein Adress-Formularfeld

Abbildung 38: Adresseingabemaske - Adressfeld

Adressvorschläge: Der Benutzer beginnt, die Postadresse einzugeben. Nach jedem eingegebenen Zeichen erhält der Benutzer immer stärker eingeschränkte Adressvorschläge, sofern das System eine Zuordnung machen kann. Falls keine Zuordnung gefunden werden kann, ist die Vorschlagsliste leer und der Benutzer ist aufgefordert, die Adresse manuell über den Link "Adresse manuell hinzufügen" einzugeben. Grundsätzlich kann die Adresse immer auch manuell eingegeben werden.

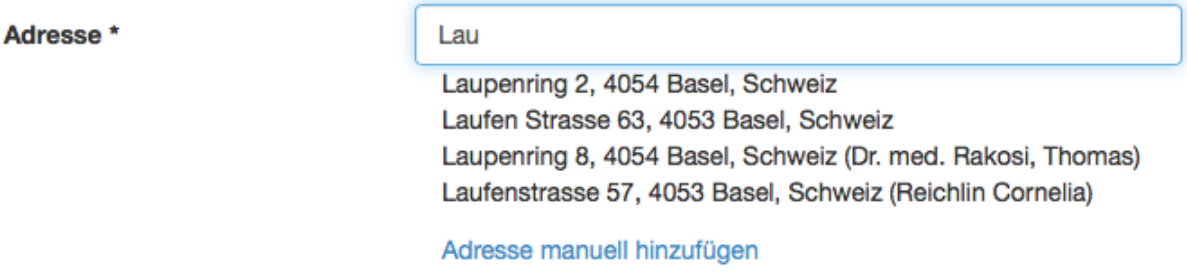


Abbildung 39: Adresseingabemaske - Adressvorschläge

Adressanzeige: Durch das Darüberfahren mit der Maus über die Adressvorschläge, wird die der Adressen direkt in einer Google Maps Karte angezeigt.

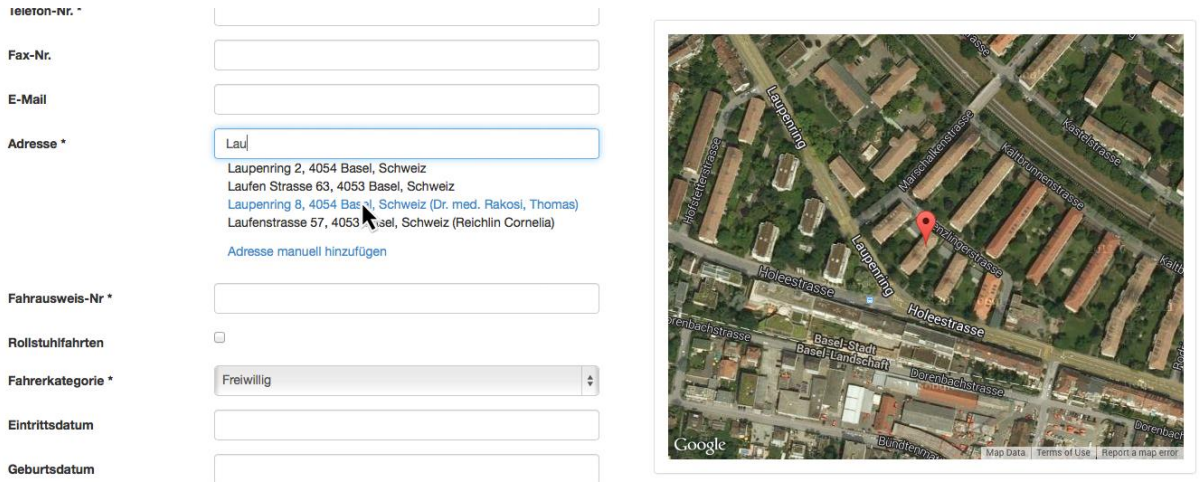


Abbildung 40: Adresseingabemaske - Adressanzeige

Adressauswahl: Durch Klicken kann eine Adresse ausgewählt werden

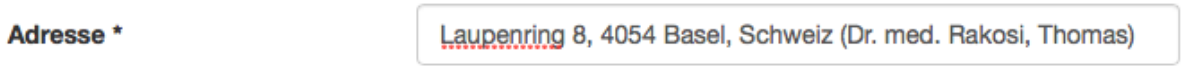


Abbildung 41: Adresseingabemaske - Adressauswahl

Adresse editieren: Durch (erneutes) Anwählen eines ausgefüllten Adressfeldes, wird die Adresse in einer Google Maps Karte angezeigt und der Benutzer hat die Möglichkeit, über den Link "Adresse manuell editieren" die Felder der Postadresse sowie auch die zugeordneten Koordinaten manuell zu korrigieren, sollten die Angaben nicht korrekt sein (was jedoch der Ausnahme entspricht). Die Korrektur der Koordinaten kann auch direkt über das verschieben des Markers auf der Google Maps Karte erfolgen. Eine Korrektur an einer Adresse führt dazu, dass das zugehörige Adressobjekt angepasst wird, wodurch eine Korrektur immer systemweit erfolgt.

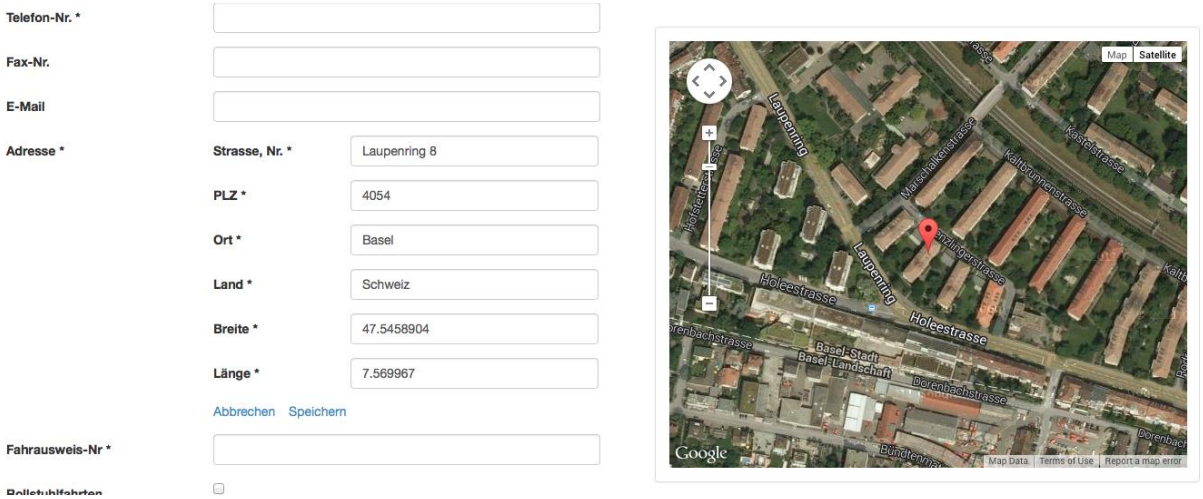


Abbildung 42: Adresseingabemaske - Adresse manuell editieren

Manuelle Adresseingabe: Wie bereits angesprochen, kann eine Adresse immer auch direkt manuell eingegeben werden. Dies sollte jedoch nur im Ausnahmefall gemacht werden, falls wirklich kein Adressvorschlag zutrifft. Für das Auffinden der Koordinaten kann erneut der Marker auf der Google Maps Karte verschoben werden. Dazu wird zuerst ein Ort über das innerhalb der Karte zur Verfügung stehende Eingabefeld gesucht und von dort ausgehend, die korrekte Position durch ziehen des Markers ermittelt. Die Koordinaten können natürlich auch manuell eingeben werden.

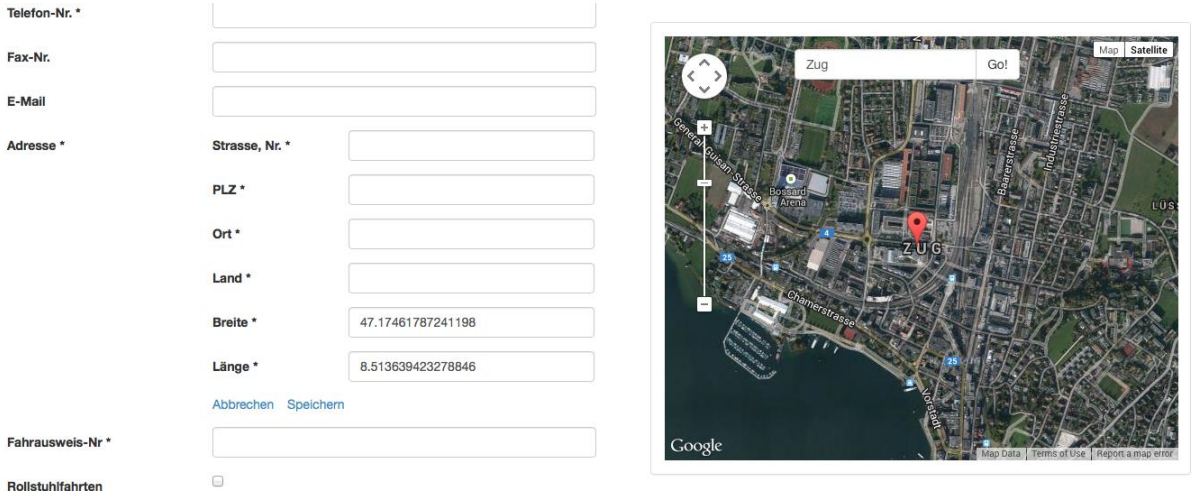


Abbildung 43: Adresseingabemaske - Manuelle Adresseingabe

10.2.5.2 Übersicht Systemkomponenten

Die Implementation des Adressverwaltungskonzeptes kann grundsätzlich in zwei Hauptkomponenten unterteilt werden: Einerseits in die Komponente, die die Adresseingabemaske rendert, die Interaktion mit der Eingabemaske steuert und die Kommunikation mit dem Adress-Service organisiert.

Diese Komponente soll als Adress-Umsystem (Adress-SurroundingSystem) bezeichnet werden. Auf der anderen Seite steht die Komponente, die als Adress-Service bezeichnet wird und die für das generieren und zuordnen von Adress-Entitäten anhand eines unstrukturierten Suchstrings verantwortlich ist.

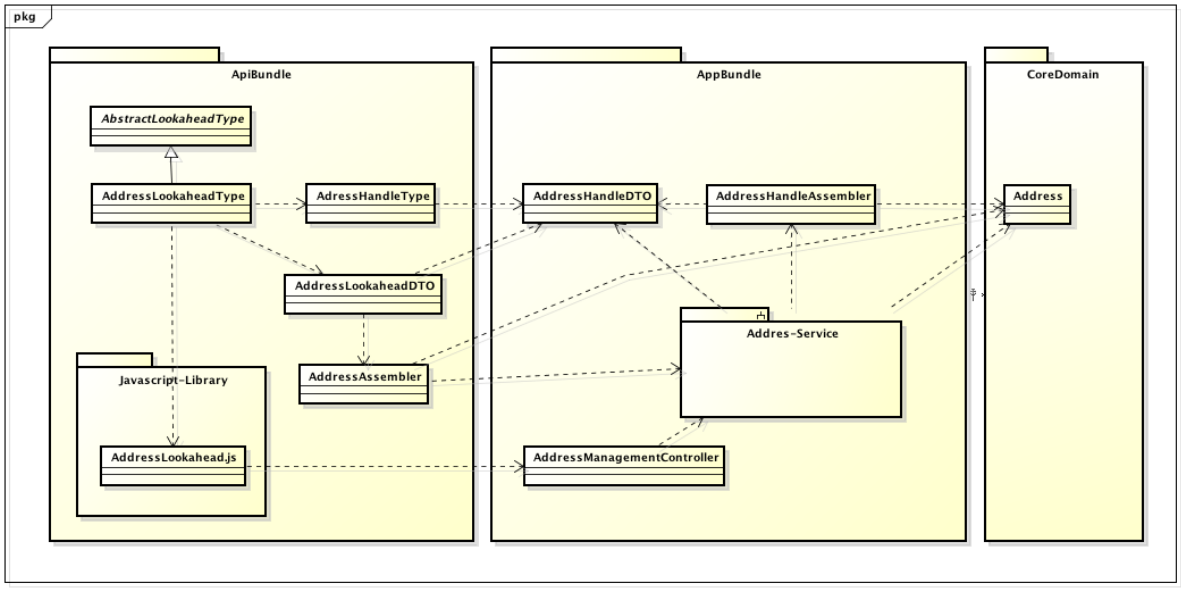


Abbildung 44: Das Adress-Umsystem

Das Adress-Umsystem kommuniziert mit dem Adress-Service grundsätzlich auf zwei Arten:

Einerseits, um anhand eines unstrukturierten Suchstrings als Inputparameter, Objekte zu erhalten, die alle nötigen Informationen enthalten, um daraus eine konkrete Instanz einer Adress-Entität zu erzeugen bzw. einer bestehenden Adress-Entität zuzuordnen. Der Suchstring wird per XMLHttpRequest von der Eingabemaske über den `AddressManagementController` an den Address-Service übertragen. Die Antwort-Objekte werden als AddressHandleDTO's bezeichnet und werden in serialisierter Form als JSON zurück an den Client übertragen, wo sie als Auswahlmöglichkeiten präsentiert werden (vergl. Abbildung 39).

Andererseits bietet der Adress-Service eine Funktion an, um AdressHandleDTO's in konkrete Adress-Objekte zu transformieren und, wenn nötig, zu persistieren. Diese Funktion wird benutzt, um aus den Form-Data-Klassen, die ein AddressLookaheadType und somit ein Adresseingabefeld enthalten, konkrete Adress-Entitäten zu extrahieren.

10.2.5.3 Adress-Service

Die wichtigste Komponente bei der Adressverwaltung ist der Address-Service, der im AppBundle implementiert ist. Kernaufgabe dieses Service ist es einerseits, einem nicht exakten Adresssuchstring, der über das Adress-Umsystem kommt, sogenannte AdressHandleDTO-Objekte zuzuordnen und diese andererseits, in einem separaten Prozess, auf konkrete Adress-Entitäten abzubilden. Als Besonderheit des Systems ist dabei die Einbindung der Google Geocoding API zu erwähnen, über die durch Einsatz eines Wrappers AdressHandles direkt aus den Adressdaten von Google erzeugt werden, falls die Suche in der lokalen Datenbank kein Treffer ergibt.

Das nachfolgende Klassendiagramm soll die Struktur des Adress-Service in der Übersicht zeigen:

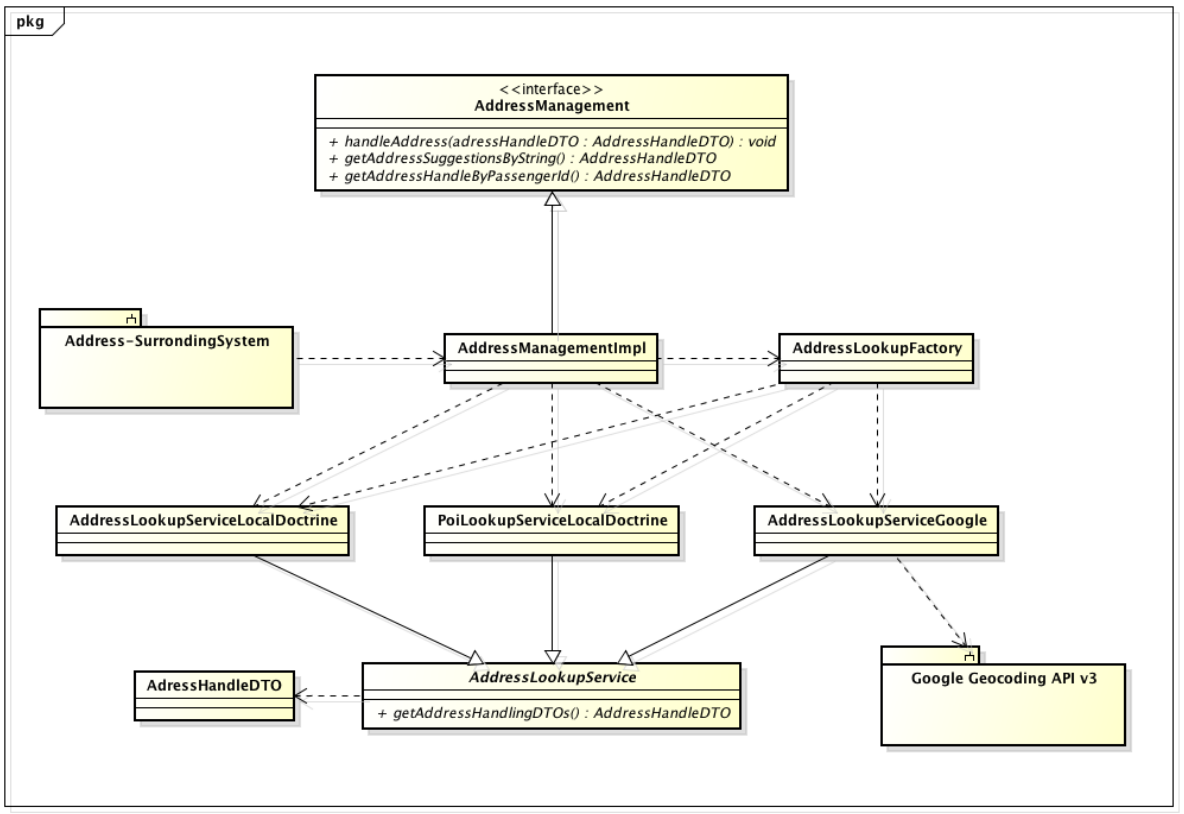
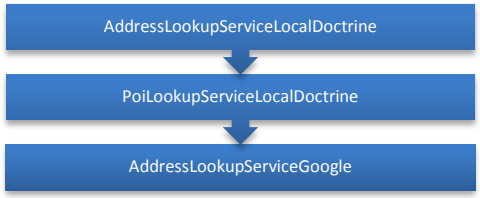


Abbildung 45: Die Komponenten des Adress-Service

Service-Trail Google Geocoding API

Der Versuch, passende Adress-Handle-Objekte zu einem unstrukturierten Suchstring zu finden, erfolgt über sogenannte AddressLookupService-Klassen bzw. über Klassen, die von dieser abstrakten Klasse erben. Jede dieser Klassen bedient sich einer anderen Quelle. Der Address-Service definiert einen sogenannten Service-Trail, der festlegt, in welcher Reihenfolge welche LookupService-Klassen nach möglichen Adress-Handle-Objekten durchsucht werden. Kann eine LookupService-Klasse kein Resultat für einen Suchstring liefern, wird zur nächsten Klasse in der Kette gesprungen, bis entweder ein oder mehrere Resultate gefunden wurden oder kein weiterer Service mehr zur Verfügung steht.

Der Service-Trail, wie er im Rahmen dieser Arbeit entwickelt wurde, ist wie folgt definiert:



AddressLookupServiceLocalDoctrine:

Zuerst wird die lokale Adress-Datenbank über den `AddressLookupServiceLocalDoctrine` durchsucht. Da dabei über alle Properties von potentiell zehntausenden von Address-Entitäten gesucht werden muss, musste der Service aus Gründen der Performanz sehr direkt mit der gewählten Datenbanktechnologie, sprich MySQL, verzahnt werden. Dazu wurde ein Symfony-Command erstellt, der MySQL Full-Text-Indexe über die Address-Table und die POI-Table anlegt. Diese werden durch absetzen der folgenden SQL-Kommandos erzeugt:

```
ALTER TABLE `address` ADD FULLTEXT `address_fts_idx` (`name`, `street`, `postalCode`, `city`,
`country`, `source`);


ALTER TABLE `poi` ADD FULLTEXT `poi_fts_idx` (`name`);
```

PoiLookupServiceLocalDoctrine:

Es ist auch möglich, Adressen der OVI's über den OVI-Name zu suchen. Dazu wird in zweiter Instanz über den `PoiLookupServiceLocalDoctrine` die POI-Table durchsucht.

AddressLookupServiceGoogle:

In letzter Instanz des Service-Trails steht die AddressLookupServiceGoogle-Klasse, die in der Funktion eines Wrappers, Suchanfragen an die Google Geocoding API weiterleitet.

10.3 Disposition

In diesem Abschnitt sollen die Implementationskonzepte für die wichtigsten Prozesse der Disposition erklärt werden.

10.3.1 Produktionsplan verwalten

Der Produktionsplan ist keiner konkreten Entity im Business Model zugeordnet sondern wird über Ein- und Ausgabeprozesse definiert, die mehrere Entitäten umfassen. Das Ziel des Produktionsplan ist es, auf Monatsbasis die Anzahl Fahrzeuge zu definieren, die für eine bestimmte Schicht zur Verfüg stehen.

Basis für den Produktionsplan bildet die Gesamtheit der im Domain Model unter 9.1.3.3 beschriebenen Funktionsgruppe von Business-Entitäten.

Gesteuert wird der Prozess im Wesentlichen durch den Dispositions-Service, der über den Symfony-Container zur Verfügung steht und vom `ProductionPlanAssembler` für die Erzeugung und Modifikation bedient wird. Die System-Prozesse, die im Zusammenhang mit der Eröffnung und Verwaltung eines Produktionsplanes stehen, lassen sich am besten über die Benutzereingabeprozesse erklären.

10.3.1.1 Übersicht Systemkomponenten

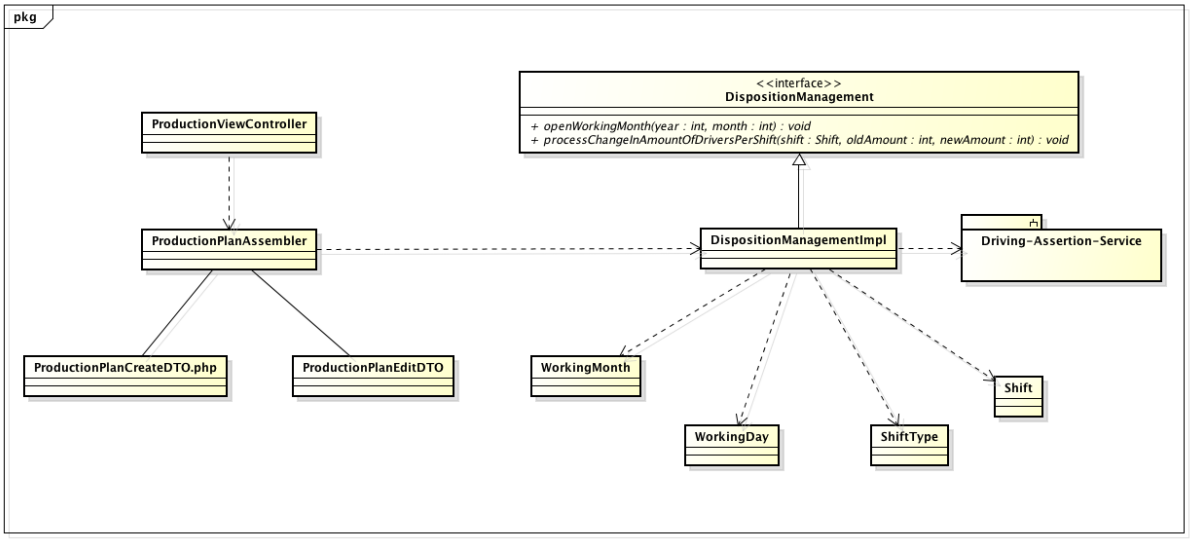


Abbildung 46: ProductionPlan Service Komponenten

Die API-Funktionalität für die Benutzerinteraktion in Zusammenhang mit dem Produktionsplan wird vom `ProductionViewController` bereitgestellt.

10.3.1.2 Produktionsplan eröffnen

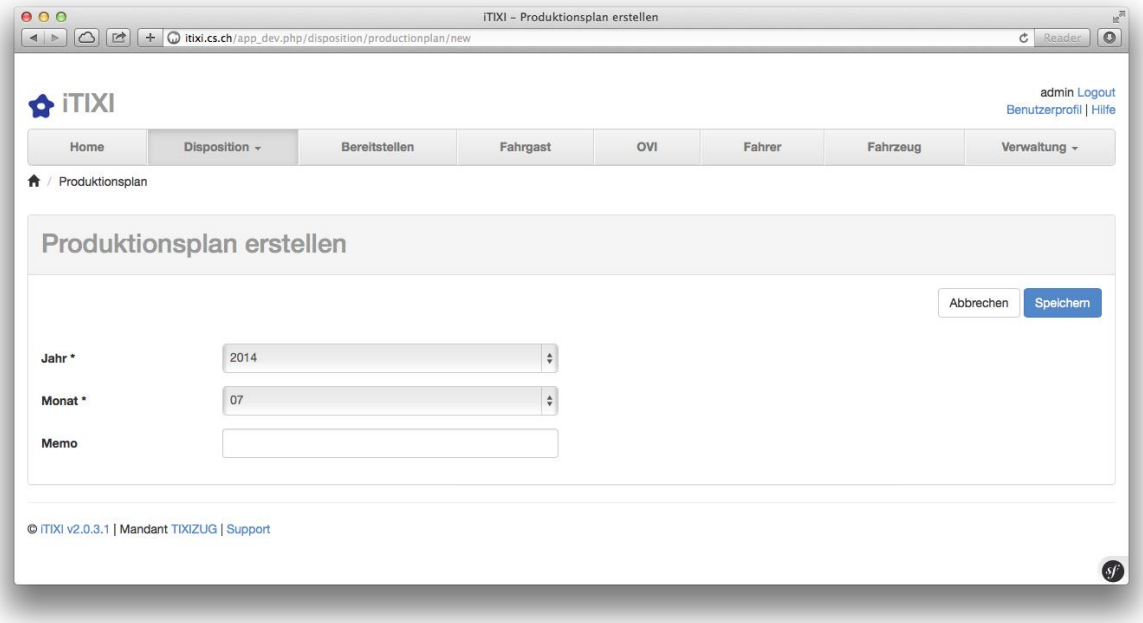


Abbildung 47: View - Produktionsplan eröffnen

Wird ein neuer Produktionsplan erstellt, so wird im Dispositions-Service (`DispositionManagementImpl`) die Methode `openWorkingMonth()` aufgerufen. Diese legt die Grundlage für einen neuen `WorkingMonth` an, sprich erstellt für alle Tage dieses Monats alle Schichten anhand der in der Verwaltung festgelegten `ShiftTypes`.

10.3.1.3 Produktionsplan editieren

Über die in den Shift-Entities für einen bestimmten WorkingMonth enthaltenen Informationen, kann über den ProductionPlanAssembler ein DTO geniert werden, das als Grundlage für eine Form steht, über das die Anzahl Fahrzeuge für eine bestimmte Schicht an das System übermittelt werden kann.

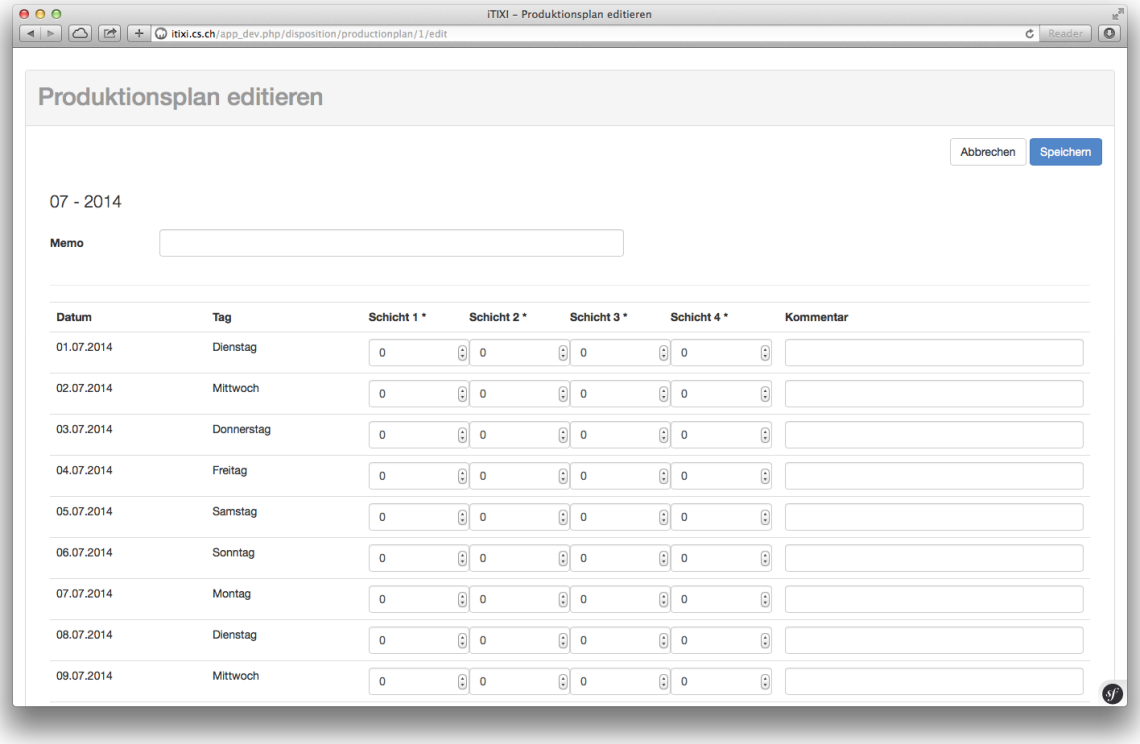


Abbildung 48: Produktionsplan editieren

Wird die Anzahl der Fahrzeuge im Produktionsplan verändert, so hat dies bindenden Einfluss auf alle angegliederten Prozesse der Disposition. Über den ProductionPlanAssembler wird beim Eintreten einer Änderung an einem `ProductionPlanEditDTO` ein Prozess angeworfen, der für das Change-Management zuständig sind. Dieser ist im `DispositionManagementImpl` angesiedelt als `processChangeInAmountOfDriversPerShift()` und erstellt oder löscht anhand der geänderten Anzahl die dazugehörigen DrivingPools.

10.3.2 Monatsplan verwalten

Aus gesundheitlichen Gründen konnte dieses Kapitel nicht mehr durch CS vervollständigt werden

10.4 Routen Abfrage

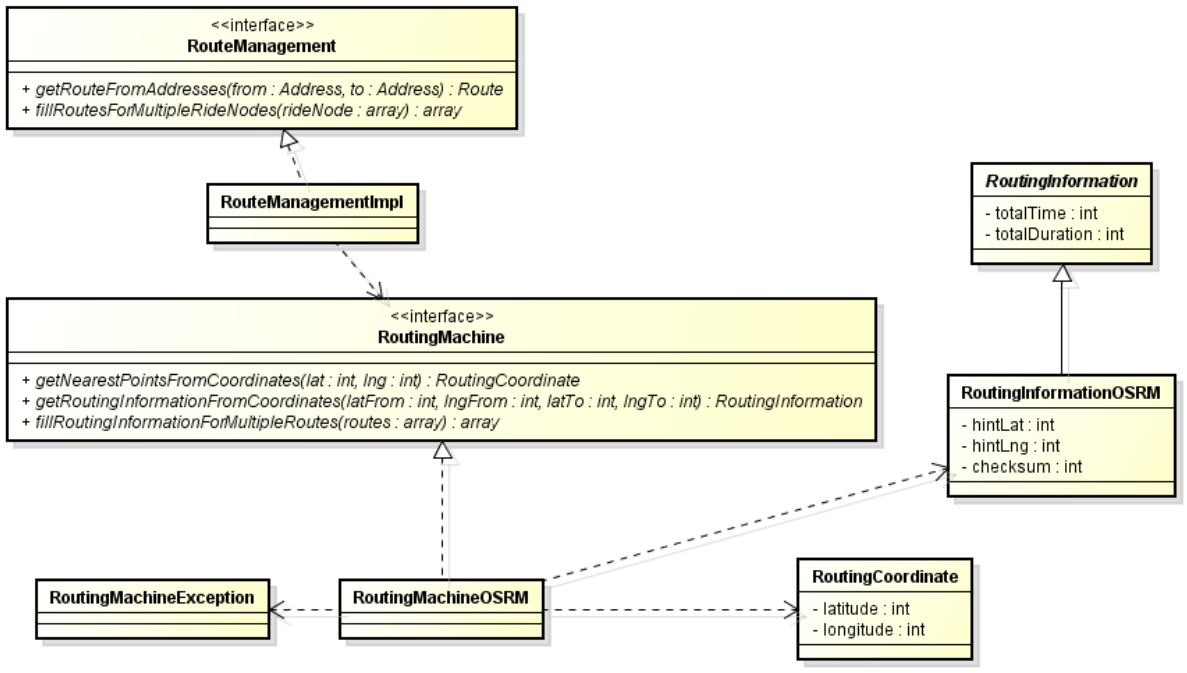


Abbildung 49: RouteManagement und RoutingMachine

Für die Abfrage von Routen wird der Service von OSRM verwendet. Im Kontext der iTIXI Applikation sind zwei Interfaces im Package App definiert:

- `RouteManagement`
- `RoutingMachine`

Mit `RouteManagementImpl` als Realisierung von `RouteManagement` und `RoutingMachineOSRM` als mögliche Realisierung einer `RoutingMachine` über OSRM.

10.4.1 RouteManagement

Für iTIXI werden die Routenabfragen für zwei verschiedene Adressen benötigt bei der Fahrauftragserfassung; über `getRouteFromAddresses()` wird ein `Route` Objekt erstellt mit den Routeninformationen zwischen Start- und Ziel-Adresse. Zwei Werte sind dabei immer erforderlich: Dauer und Distanz. Die abstrakte Klasse `RoutingInformation` sollte diese beiden Informationen immer beinhalten für die Berechnung mit der RoutingMachine.

Der weitere Anwendungsfall ist die mengenmässig Abfrage an Routen für die Fahrtoptimierung, hier werden viele Routeninformationen für die möglichen Leerfahrten berechnet. Bei der Fahrtoptimierung wird mit RideNode Objekten gearbeitet, das RouteManagement stellt deshalb eine geeignete Funktion über `fillRoutesForMultipleRideNodes()` zur Verfügung.

Für einen Fahrauftrag wird immer ein Route Objekt persistiert, da hier auch die Start und Ziel-Adresse festgehalten werden und somit die Informationen zum Fahrauftrag über Route und Address immer abrufbar sind. Man bekommt die Informationen über die Route eines Fahrauftrags somit auch wenn keine RoutingMachine (Lokaler Service OSRM oder andere Online-Dienste) erreichbar ist.

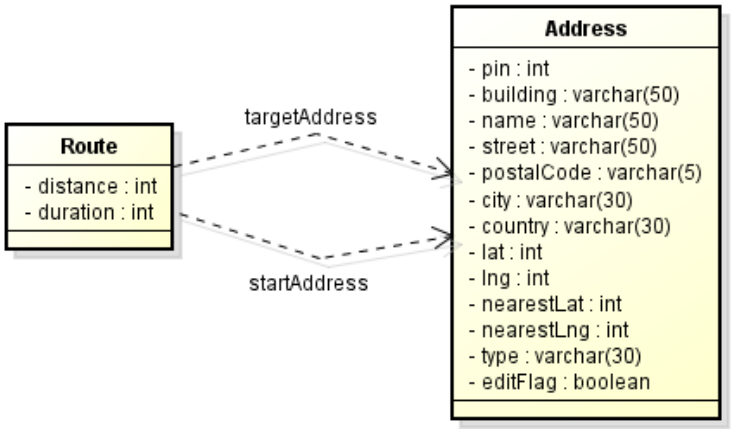


Abbildung 50: Detail Route und Address Beziehung

Eine Route definiert sich immer über die Start und Ziel-Adresse, die Identifier von targetAddress und startAddress bestimmen deshalb den Primary Key von Route:

```
UNIQUE KEY `search_idx` (`address_start_id`,`address_target_id`),
KEY `IDX_2C420799CB2071A` (`address_start_id`),
KEY `IDX_2C420798FCF372E` (`address_target_id`),
CONSTRAINT `FK_2C420798FCF372E` FOREIGN KEY (`address_target_id`) REFERENCES `address` (`id`),
CONSTRAINT `FK_2C420799CB2071A` FOREIGN KEY (`address_start_id`) REFERENCES `address` (`id`)
```

Bei der Abfrage einer Route für zwei Adressen wird zuerst in der Datenbank nach einer vorhandenen Route gesucht, dies kommt vor wenn zwei gleiche Fahraufträge erfasst werden. Wenn keine Route vorhanden ist werden über die RoutingMachine die passenden Informationen abgefragt.

Das Repository für die Route stellt die passenden Abfrage-Funktionen für vorhandene Routen zur Verfügung:

```
/**
 * @param Address $from
 * @param Address $to
 * @return Route
 */
public function findRouteWithAddresses(Address $from, Address $to) {
    $qb = $this->createQueryBuilder('e')
        ->where('e.startAddress = :startAddressId')
        ->andWhere('e.targetAddress = :targetAddressId')
        ->setParameter('startAddressId', $from->getId())
        ->setParameter('targetAddressId', $to->getId());
    return $qb->getQuery()->getOneOrNullResult();
}
```

Erst beim speichern eines Fahrauftrags werden auch dazugehörige Route-Objekte angelegt, falls nicht schon vorhanden.

Während der Erfassung eines Fahrauftrags haben wir den Fall, dass Dauer und Distanz in der entsprechenden View angezeigt werden, zu diesem Zeitpunkt aber noch keine Address-Objekte vorhanden sind.

Durch den Address-Lookahead sind beim Client die Koordinaten einer Adresse bekannt, ob diese Informationen aus der Datenbank oder über einen Online-Dienst kommen ist aber nicht gegeben.

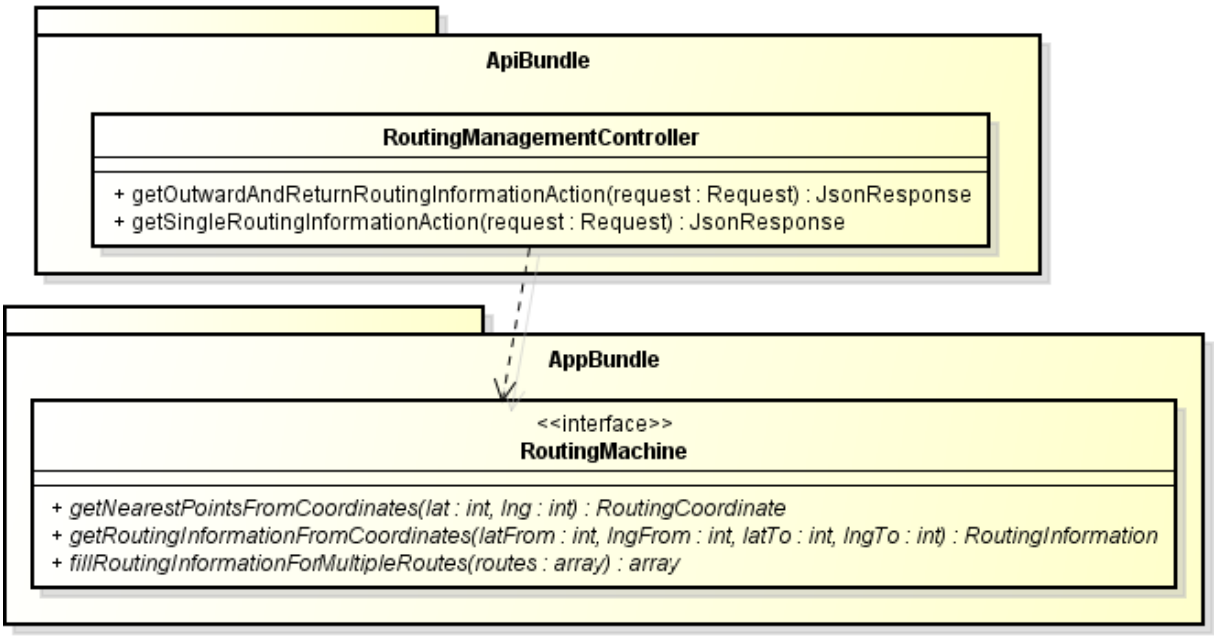


Abbildung 51: RoutingManagementController Zugriff

Der Controller `RoutingManagementController` erwirbt deshalb direkt über eine RoutingMachine die Informationen zu Dauer und Distanz und stellt diese dem Client als JSON zur Verfügung:

Methode	`getOutwardAndReturnRoutingInformationAction()`
API-Call	`/service/routing?latFrom=0.0&lngFrom=0.0&latTo=0.0&lngTo=0.0`
Response	`{"status":"0","routeOutwardDuration":1,"routeOutwardDistance":0,"routeReturnDuration":1,"routeReturnDistance":0}`

Für die Abfrage werden gleich Hin- und Rückfahrt berechnet, dabei werden einfach die Koordinaten vertauscht und dementsprechend zwei Abfragen über die RoutingMachine ausgeführt. Der übliche Fall beim Erfassen eines Auftrags ist mit einer Rückfahrt verbunden, es muss somit nur eine Abfrage vom Client aus getätigt werden.

10.4.2 RoutingMachine

10.4.2.1 OSRM Services

Für die Implementation zur Routen-Berechnung wird OSRM genutzt. Der OSRM Server bietet ein webbasiertes API an und ermöglich einfache Abfragen über HTTP. Die Response kann wahlweise über JSON oder JSONP erfolgen. Die OSRM Server API bietet drei Services an[19]:

Service	Nearest node im Strassennetzwerk zu einer Koordinate abfragen
API-Call	http://seiout.ch:8080/locate?loc=47.096531,8.463097
Response	{"version":0.3,"status":0,"mapped_coordinate":[47.096569,8.463191],"transactionId": "OSRM Routing Engine JSON Locate (v0.3)"}

Nearest Node sucht den nächstgelegenen Punkt auf einem Strassennetzwerk.

Service	Nearest point eines Strassensegments zu einer Koordinate abfragen
API-Call	http://seiout.ch:8080/nearest?loc=47.096531,8.463097
Response	{"version":0.3,"status":0,"mapped_coordinate":[47.096569,8.463097],"name":"Dorfplatz","transactionId":"OSRM Routing Engine JSON Nearest (v0.3)"}

Der Nearest Point sucht die nächstgelegene Koordinate, die auf irgendeinem Strassensegment vorhanden ist.

Diese Koordinate ist, wie im Kapitel Grundlagen erwähnt, notwendig für die Routen-Berechnung.

Service	Routen zu einer Anzahl Koordinaten abfragen
API-Call	http://seiout.ch:8080/viaroute?loc=47.498796,7.760499&loc=47.049796,8.548057
Response	{"version": 0.3,"status":0,"status_message": "Found route between points","route_geometry": "..","route_instructions": [],"route_summary":{"total_distance":108756,"total_time":4000,"start_point":"","end_point":"Parkstrasse"},"alternative_geometries": [".."],"alternative_instructions":[[]],"alternative_summaries":[{"total_distance":117105,"total_time":4321,"start_point":"","end_point":"Parkstrasse"}],"route_name":["",""],"alternative_names":[["",""]],"via_points":[[47.498796,7.760499],[47.049796,8.548057]],"hint_data": {"checksum":1617606390, "locations": ["KodMAAAAAAAbAAAAAAAAAAAAAAAAAAPA_LMbUAnNqdgA", "VYMVAA8FAAASAAAAIQAAAIxX3Wb-Xdc_ROzNAtluggA"]},"transactionId": "OSRM Routing Engine JSON Descriptor (v0.3)"}

Für die Abfrage von Routen-Informationen können mehrere Koordinaten (bis max. 25) im Request mitgegeben werden, wobei für unsere Routen-Berechnung meist nur Start und Ziel Koordinaten notwendig und keine weiteren „Via“ Punkte zu berücksichtigen sind. Mehrere Koordinaten würde man einfach nacheinander aufführen: `viaroute?loc=lat1,lon1&loc=lat2,lon2&loc=lat3,lon3&loc=lat4,lon4&...&latX,lonX`

[19] Siehe auch OSRM-Wiki: https://github.com/DennisOSRM/Project-OSRM/wiki/Server-api

10.4.2.2 OSRM Requests

Um einen Request ab zu setzten wird die cURL[20] Extension von PHP verwendet. Über einige cURL Parameter kann festgelegt werden, ob z.B. ein HTTP Header mitgegeben wird (für OSRM Web API nicht benötigt).

Da OSRM bei der TIXI Applikation auf der gleichen Server-Instanz betrieben wird und nur ein HTTP API anbietet, werden aus Performancegründen alle unnötigen Verifizierungen bei cURL deaktiviert.

```php
/**
 * @param $requestUrl
 * @return resource
 */
private function createCurlRequest($requestUrl) {
    $curl = curl_init();
    $options = array(
        CURLOPT_URL => $requestUrl,
        CURLOPT_POST => 0,
        CURLOPT_TIMEOUT => 30,
        CURLOPT_HEADER => 0,
        CURLOPT_RETURNTRANSFER => 1,
        CURLOPT_DNS_USE_GLOBAL_CACHE => 1,
        CURLOPT_DNS_CACHE_TIMEOUT => 3600,
        CURLOPT_SSL_VERIFYPEER => 0,
        CURLOPT_SSL_VERIFYHOST => 0,
        CURLOPT_FOLLOWLOCATION => 0,
        CURLOPT_IPRESOLVE => CURL_IPRESOLVE_V4,
    );
    curl_setopt_array($curl, $options);
    return $curl;
}
```

Abbildung 52: cURL Request mit Parametern für OSRM Abfrage

Weiter wird durch curl_multi eine parallele Abfrage ermöglicht indem mehrere cURL Handles werden dabei gleichzeitig abgesetzt und parallel abgearbeitet. Dies ergibt einen enormen Geschwindigkeitsvorteil gegenüber mehreren einzelnen Handles.

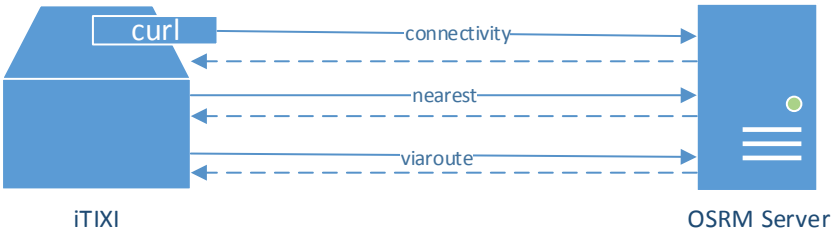


Abbildung 53: cURL single Request OSRM

Eine einzelne Routing-Information wird abgefragt, indem zuerst eine Verbindung über cURL aufgebaut wird und die Konnektivität zum OSRM Server testet. Danach wird ein cURL Handle erstellt um einen Request für die Abfrage der Nearest Point Koordinaten auszuführen. Wurde für die Abfrage eine bestehende Adresse aus der Datenbank gewählt, sind die Nearest Point Koordinaten meistens schon bekannt und der Request ist nicht nötig. Anschliessend wird ein cURL Handle erstellt um einen Request die Abfrage der Route zwischen diesen beiden Nearst Point Koordinaten auszuführen.

Um mehrere Abfragen gleichzeitig zu tätigen, was gerade für die Fahrtoptimierung benötigt wird um eine grosse Menge an Routen für die möglichen Leerfahrten zu berechnen. Über curl_multi werden mehrere Handles gleichzeitig erstellt, abgesetzt und verarbeitet.

[20] cURL: Client for URLs, verfügbar als Extension zu PHP

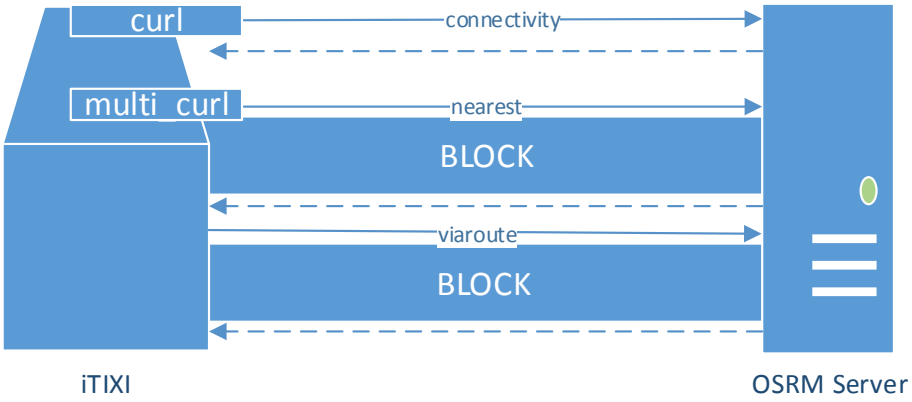


Abbildung 54: cURL multi Request OSRM

Bei unserer Implementierung wird über eine BLOCKSIZE die Anzahl der maximal simultan laufenden cURL Handles für curl_multi angegeben. Pro Block wird immer diese Anzahl cURL Handles ausgeführt, auf die Responses gewartet und die Resultate weiter verarbeitet.

Dabei spielt die Bandbreite und Verbindungsqualität eine Rolle bei der Anzahl an Handles, die gleichzeitig möglich sind. Die Performance von curl_multi variiert je nach Netzwerk-Stack des Betriebssystems und der kompilierten cURL Library. Eine Blocksize von 40 hat sich durch Tests als guter Mittelwert bestätigt zwischen Stabilität und Geschwindigkeit. Über das Abfragen einiger hundert Zufallsrouten wurden sporadisch verschiedene Blocksizes an einem Entwickler-Laptop ausprobiert. Ist eine zu schlechte Verbindung vorhanden, kann cURL nicht alle Handles absetzen und die parallelen Abfragen zu OSRM schlagen fehl.

Device: Lenovo mit IntelPRO, Windows 7 x64, easyPHP + cURL Extension
Verbindung: 100Mbit Ethernet an der HSR, Latenz zu OSRM Server seiout.ch: 32ms

Blocksize	Benötigte Zeit für 912 Abfragen
100	2.74s
80	2.80s
60	3.09s
40	3.52s
20	5.10s
10	8.13s

Schlägt eine Verbindung fehl oder konnten keine Daten vom OSRM Server abgerufen werden, wird eine `RoutingMachineException` geworfen mit Begründung über den Fehler.

Mit dem parallelen Absetzen von Requests errechnen sich mehr als 100 Routen pro Sekunde. Da wir bei den meisten Adressen die Nearest Point Koordinaten zusätzlich abspeichern, entfällt diese Abfrage und es wird zusätzliche Zeit eingespart. Bei der Fahrtoptimierung ist die Berechnung von Leerfahrten nötig zwischen Ziel-Adresse des einen Fahrauftrags und Start-Adresse des anderen Fahrauftrags – hier werden die Nearest Point Koordinaten immer vorhanden sein und minimiert die benötigte Zeit für das Berechnen der optimalen Fahrten um ein Weiteres.

10.5 Fahrwegsoptimierung

Wie im Kapitel Grundlagen erwähnt, wurde versucht eine mögliche Implementierung der Fahrtoptimierung zu realisieren. Im Package `App` wird dieser Service durch das Interface `RideManagement` definiert.

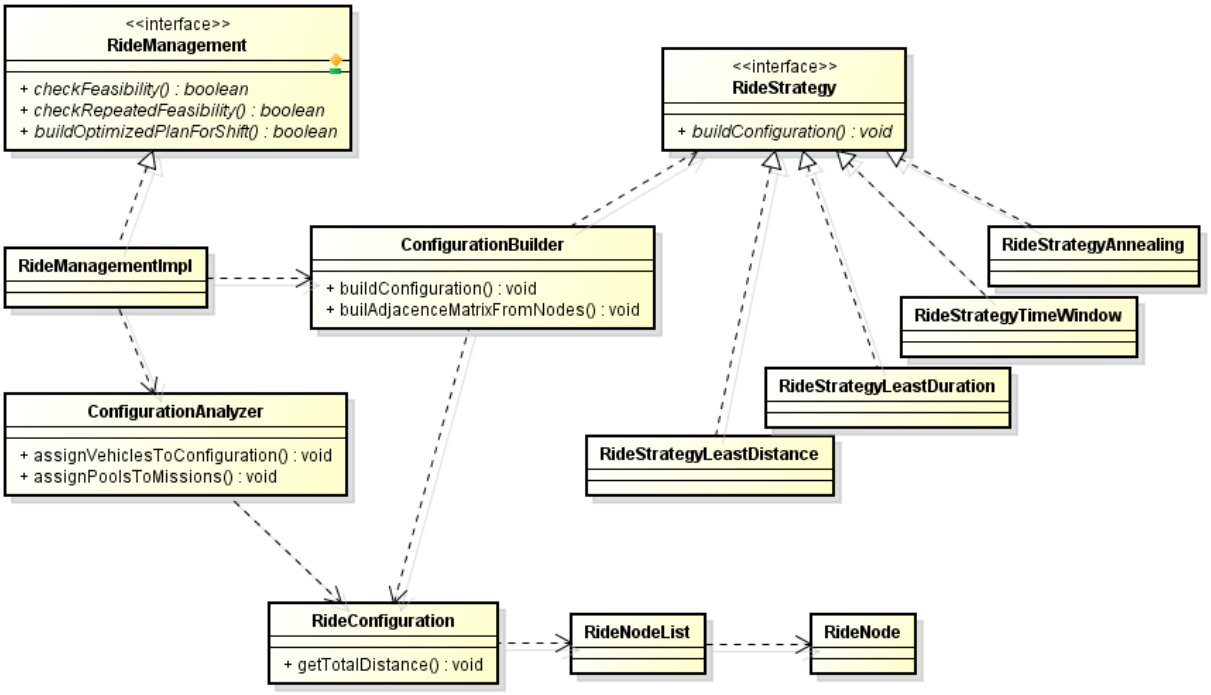


Abbildung 55: RideManagement Klassen Übersicht

Die Konfigurationsbildung lehnt den im Kapitel Grundlagen – DARP definierten Begriffen an.

Es wurde darauf geachtet, dass der Prozess der Konfigurationserzeugung selbst austauschbar ist, d.h. es sollen verschiedene Algorithmen einsetzbar sein. Deshalb wurden Konfigurations-Strategien definiert und gemäss dem Muster des Strategy Patterns[21] implementiert.

10.5.1 Konfigurationserzeugung

Die Klasse `ConfigurationBuilder` hält eine Strategie vom Typ `RideStrategy` als Member-Variable und führt die Konfigurationsbildung `buildConfiguration()` auf einer konkreten Strategie aus, z.B. RideStrategyAnnealing.

[21] Strategy Pattern: Entwurfsmuster aus GoF

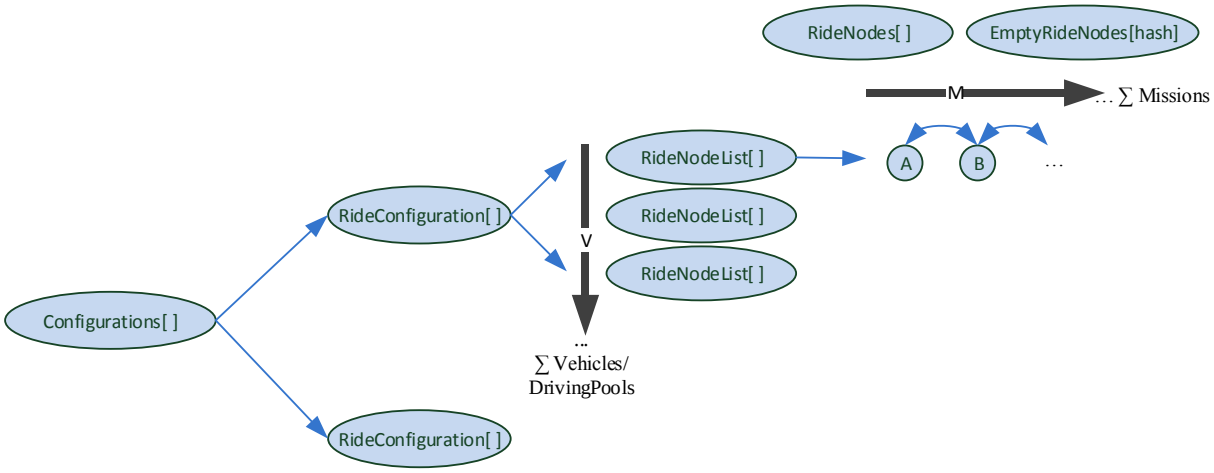


Abbildung 56: Aufbau der Rideconfigurations

Im `ConfigurationBuilder` wird ein Array mit den erzeugten RideConfigurations gehalten, woraus dann die beste Konfiguration genommen werden kann für weitere Zuteilungen.

Die RideConfigurations werden durch eine Strategie erzeugt, in dem pro benötigte Fahrzeuge eine RideNodeList angelegt wird, die eine LinkedList von RideNodes pflegt (siehe Abbildung 56).

Die Menge an RideNodes ergibt sich aus allen DrivingMissions die optimiert werden.

Unsere Implementierung sieht vor, dass pro Schicht optimiert werden kann. Alle `DrivingMissions` die Ihre Startzeit innerhalb einer Schicht haben werden dazu übernommen (siehe Abbildung 57).

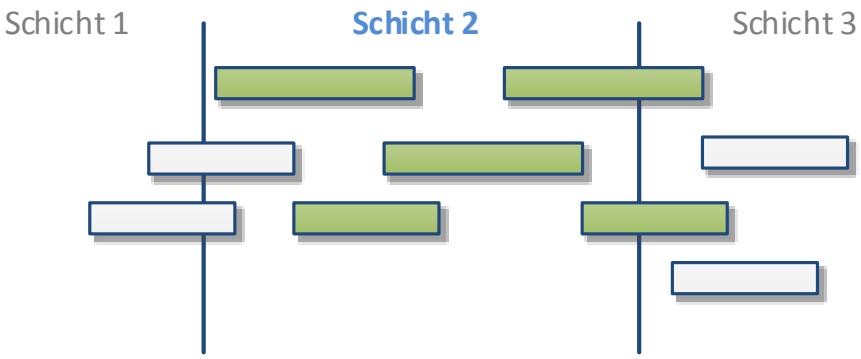


Abbildung 57: Beispiel DrivingMissions in Schicht2

Aus den DrivingMissions werden dann RideNodes erzeugt, wobei nur noch Startzeit in Minuten + Fahrtdistanz + Fahrtdauer + zusätzliche Minuten (Boarding Times etc.) interessieren für die Berechnung. Ein RideNode dient in diesem Sinne als DTO für die Konfigurationsbildung innerhalb einer Strategie.

Die Leerfahrten können dann aus allen vorhandenen RideNodes zusammengestellt und über die RoutingMachine abgerufen werden. Diese werden in einer geeigneten HashTable EmptyRideNodes hinterlegt, welche aus RideNodes besteht die keiner DrivingMission entstehen und als Key einen geeigneten Hash aus den Start und Ziel Koordinaten haben. Der HashKey wird per MD4-Hashfunktion berechnet aus einem zusammengesetzten String von Latitude + Longitude der Startadresse und Zieladresse.

Dieser Hash entspricht damit genau einer Route. Sollte eine gleiche Leerfahrt entstehen zwischen zwei RideNodes, so wird diese unter dem gleichen Value in der Hashtable gespeichert und muss somit nur einmal

abgefragt werden. MD4 wurde gewählt, da diese Hashfunktion in PHP gemäss Benchmarks[22] sehr schnell berechnet ist und mit einem 128-Bit Hash genügend grosse Sicherheit gegenüber möglichen Kollisionen bietet.

Über die Abfrage eines Hashs zwischen zwei RideNodes (z.B. Zieladresse von RideNode 1 und Startadresse von RideNode 2) kann somit innert $O(1)$ die entsprechende Leerfahrt herausgelesen werden.

Aus allen Fahrten und Leerfahrten wird dann eine Adjacence-Matrix erstellt, die Auskunft über machbare aufeinanderfolgende RideNodes gibt, und wenn diese machbar sind, uns gleich die entsprechende Leerfahrt zurückgibt. Mit dieser Adjacence-Matrix als Basis, werden dann die Konfigurationen gemäss einer Strategie erstellt:

RideStrategyTimeWindows

- erstellt aufgrund bereits zugeteilten DrivingMissions und DrivingPools eine Konfiguration von RideNodes und fügt nicht zugeteilte RideNodes, die zur bestehenden Konfiguration in einem Zeitbereich machbar sind, hinzu. Diese Konfiguration dient nur dazu, über Zeitslots herauszufinden, ober ein Fahrauftrag überhaupt machbar wäre und nicht schon mehr Fahraufträge zeitlich im gleichen Fenster liegen als die Menge an vorhandener Fahrzeuge.

RideStrategyLeastDistance

- gemäss einfachem Algorithmus. Hier werden verschiedene RideNodes als Start für eine Fahrt genommen und darauf immer die nächste RideNode, welche machbar ist, mit kleinster Distanz zugeteilt. Aus den verschiedenen machbaren Konfigurationen mit unterschiedlichen Start Nodes wird die günstigste gewählt.

RideStrategyLeastDuration

- gemäss einfachem Algorithmus, einfach darauf ausgelegt alle möglichen RideNodes nach Fahrtzeit in einer Konfiguration zu haben. Bei RideStrategyLeastDistance ist es möglich, nicht machtbare Konfigurationen zu erzeugen wenn nicht alle Aufträge einer RideNodeLists, bzw. einem Fahrzeug, zugeordnet werden können.

RideStrategyAnnealing

- erstellt zuerst eine machbare Konfiguration mit einem einfachen Algorithmus wie *RideStrategyLeastDuration*. Diese Initialkonfiguration wird gemäss dem Optimierungsverfahren Simulated Annealing auf ein globales Minimum gebracht.

Die RideNodeList hält beim Einfügen und Entfernen von RideNodes immer folgende Daten fest:

- gesamte Distanz aller Fahrten und Leerfahrten
- totale Dauer aller Fahrten und Leerfahrten
- maximale Anzahl Passagiere
- maximale Anzahl Rollstühle
- nicht gewünschte Fahrzeugkategorien durch Passagiere

Durch die Summe aller RideNodeList Daten hält eine RideConfiguration dann die gesamte Distanz und Dauer für eine Konfiguration fest, aufgrund dieser man die verschiedenen Konfigurationen messen kann.

[22] http://benchmarks.ro/2009/04/hashing-algorithms-benchmarked/#benchmark_code

10.5.2 Konfigurationsanalyse

Sind erst einmal günstige Konfigurationen zusammengestellt, werden zu der bestmöglichen noch Fahrer und Fahrzeuge zugeordnet. Dies wird über die Klasse `ConfigurationAnalyzer` bestimmt, ein *Analyzer* untersucht eine Konfiguration und ordnet weitere Objekte zu.

Verschiedene *Analyzer* als Strategien zu implementieren sahen wir momentan nicht als Notwendigkeit, die Zuordnung ist meist fix geregelt durch Constraints und für unser Beispiel recht klar welche Fahrer und Fahrzeuge für einen Ride geeignet sind. Trotzdem wäre es vorstellbar, unterschiedliche Constraints zu verfolgen und diese als weitere Strategien zu implementieren.

Vorhandene Vehicles werden über `assignVehiclesToConfiguration()` geprüft auf die Verträglichkeit mit einem Ride. Die Liste der Vehicles wird nach Fahrzeuggrösse sortiert um kleinere und meist energieeffizientere Fahrzeuge bei der Auswahl zu bevorzugen. Der Sortierprozess beruht sich hierbei nur auf die Summer der Anzahl Sitzplätze und Rollstuhlplätze. Es werden alle RideNodeLists mit den Fahrzeugen verglichen und die Verträglichkeit anhand der festgehaltenen Informationen in einer RideNodeList bestimmt.

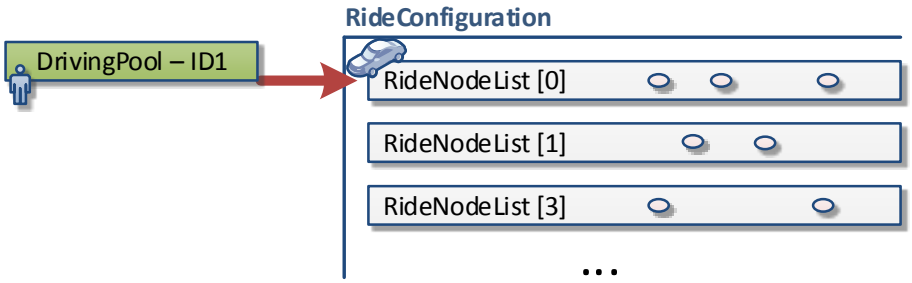


Abbildung 58: Zuteilung Fahrer/Fahrzeuge zu Fahrten

Die vorhandenen DrivingPools, welche bei der Monatsplanung einen Fahrer zugeordnet bekamen, werden nun über `assignPoolsToMissions()` den mit Fahrzeug verknüpften RideNodeLists zugeteilt, wobei auf die Verträglichkeit mit den Fahrern geachtet wird – Fahrer zu Fahrzeugkategorien und wenn der Fahrer Extra-Minuten hat, ob die nacheinander folgenden Aufträge mit dieser zusätzlichen Zeit machbar sind.

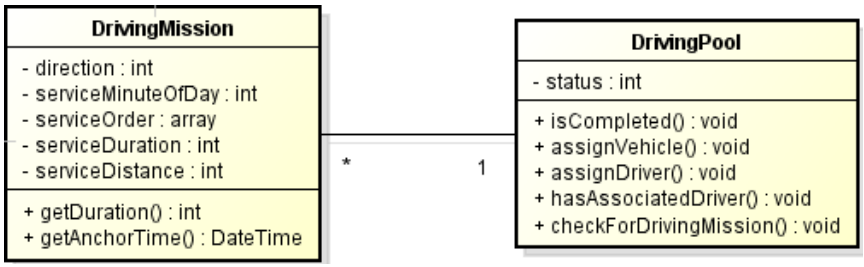


Abbildung 59: DrivingMission zu DrivingPool

Ist die Zuteilung erfolgreich, wird jeweils die aus den RideNodes enthaltene DrivingMission mit dem verträglichen DrivingPool verknüpft. Die Optimierung einer Schicht wird dann beendet.

10.5.3 Annealing

Um kurz auf den Annealing Prozess einzugehen, soll nachfolgende Vereinfachung der Implementation zur Erläuterung dienen.

```
$temperature = 10000.0;
$coolingRate = 0.999;
$absoluteTemperature = 0.0001;
//loop until system has cooled, this is our break criterion
while ($temperature > $absoluteTemperature) {
        //swap two nodes
        $nextConfiguration = $this->getNextRandomConfiguration($currentConfiguration);
        if ($nextConfiguration !== null) {
                $nextDistance = $nextConfiguration->getTotalDistance();
                $deltaDistance = $nextDistance - $distance;
                //random next double 0.0 > ran < 1.0
                $ran = mt_rand(1, 100000000) / 100000000;
                //we accept the configuration if distance is lower or satisfies Boltzmann condition
                if (($deltaDistance < 0) || ($distance > 0 && exp(-$deltaDistance / $temperature) > $ran)) {
                        $distance = $nextDistance;
                        $currentConfiguration = $nextConfiguration;
                        //keep track of the best configuration
                        if ($distance < $bestDistance) {
                                $bestDistance = $nextDistance;
                                $configurations[] = clone $nextConfiguration;
                        }
                }
        }
        $temperature *= $coolingRate;
}
```

Abbruchkriterium ist die Temperatur, wenn diese unter die *absoulteTemperature* fällt endet die Prozedur. Weiter denkbar wären noch Zeit Kriterien; falls der Prozess eine gewisse Zeit überschreitet, soll beendet werden. Bei den Tests wurde aber nie eine Ausführungszeit von 1min überschritten.

über `getNextRandomConfiguration()` werden aus der momentan behandelten Konfiguration zufällig zwei RideNodes miteinander vertauscht und die gehaltenen Informationen aktualisiert, aber nur wenn diese machbar sind. Bleibt die Konfiguration machbar, wird überprüft, ob damit eine bessere Distanz erreicht wird. Wenn eine günstigere Distanz erreicht wurde oder die Wahrscheinlichkeit durch die Boltzmann Verteilung gegeben ist, wird diese Konfiguration aus Ausgangslage für die nächsten Berechnungen gesetzt. Ist die entstandene Distanz günstiger als die globale Distanz *bestDistance*, wird die Konfiguration zu einem Set hinzugefügt, aus dem nach dem Annealing die Beste genommen wird.

Nach jedem Schritt kühlt die Temperatur durch die gegebene *coolingRate* ab.

Um die entstandene Konstellation zu veranschaulichen ohne der implementierten View zur Tagesplanung, wurden die Resultate per *print()* ausgegeben.

Im Beispiel von Abbildung 60 wurden 1608 Routen (die möglichen Leerfahrten) innert 8 Sekunden abgefragt.

59 Aufträge wurden für eine Schicht bearbeitet. Die Initialkonfiguration über RideStrategyLeastDuration ergab eine Gesamtdistanz von 282.969km, über den Annealing Prozess wurde eine Gesamtdistanz von 200.761km innert 4 Sekunden erreicht. Die Tabelle zeigt eine Zuweisung von DrivingPools (ID links), die maximale Anzahl Rollstühle (W) und Passagiere (P) in einer RideNodeList, das zugewiesene Fahrzeug und nachfolgend die Fahraufträge/DirivingMissions (ID in Klammern) hintereinander.

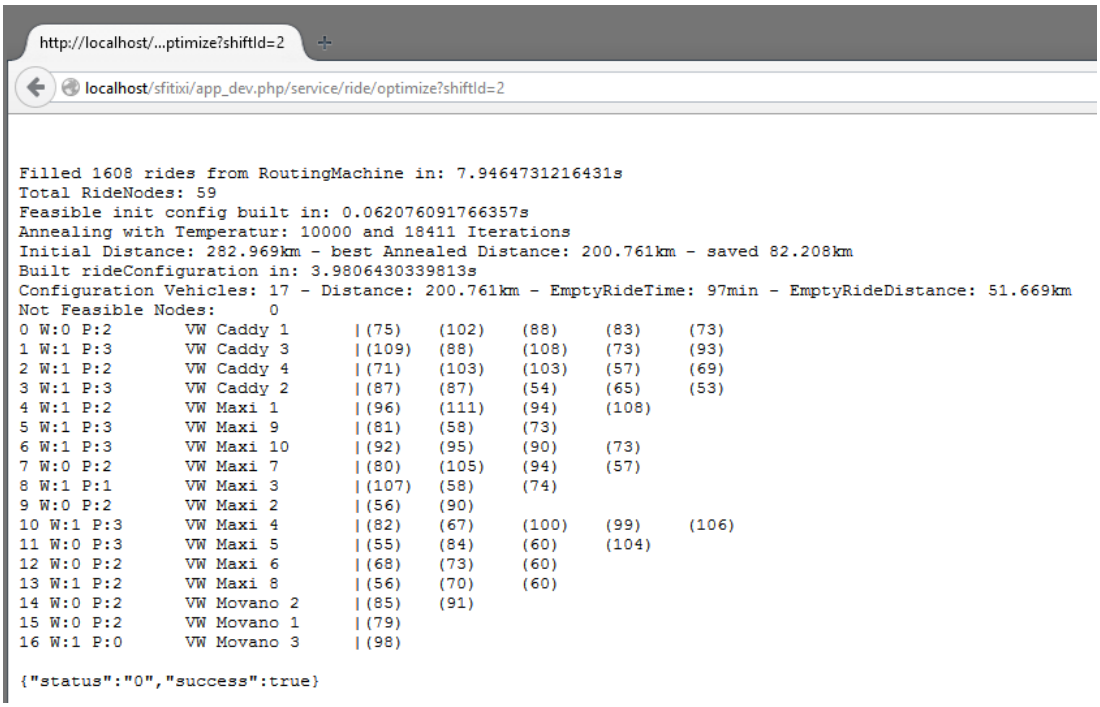


Abbildung 60: Print einer optimierten Schicht

Weitere Testresultate sind im Appendix aufgeführt. Die Tests wurden hier auf einem leistungsstarken Arbeitsplatz (Core i7 3.4GHz) weshalb die Ausführungszeiten recht kurz sind.

10.6 Security

Das gesamte Security-Konzept wurde von Symfony übernommen. Eine Formular-basierte Anmeldung mit Schutz vor Cross-site request forgery (CRSF) zur Authentisierung von Benutzern, Access-Controls von URL's und Access-Control-Lists (ACL) zur Authorisierung und Freigabe von Ressourcen, diese Konzepte werden durch eine PHP Library von Symfony zur Verfügung gestellt. Die Verbindung zwischen Client und Server soll, wie im Kapitel Deployment aufgeführt, durch HTTPS gesichert sein.

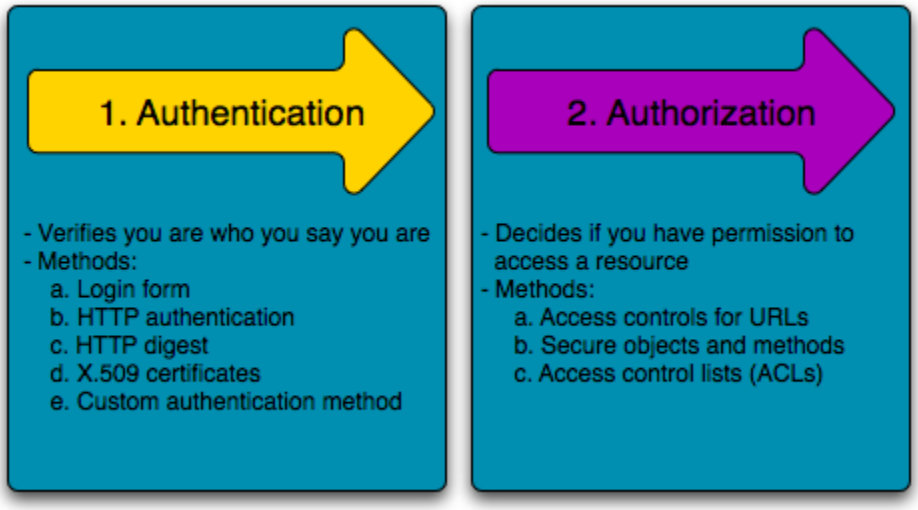


Abbildung 61: Authentisierung und Authorisierung gemäss Symfony (Quelle: symfony.com)

Die Security Klassen wurden gesondert von der Domain Implementierung als zusätzliches Bundle realisiert, da diese sehr nahe an Symfony anzugliedern sind und nicht zur eigentlichen Implementierung gehören.

Über einen UserProvider kann die Ablage und Verwaltung von Benutzern und Rollen für Symfony individuell gestaltet werden. Rollen (bzw. ROLES) entsprechen soweit eine Benutzergruppe, derer man verschiedene Berechtigungen erstatten kann. Ein Benutzer kann in mehreren Benutzergruppen zugeordnet sein.

Bei iTIXI werden Benutzerobjekte und Rollen in der MySQL Datenbank persistiert. Benutzerkennwörter werden per BCrypt[23] zu einem geeigneten Hash für die Ablage umgewandelt.

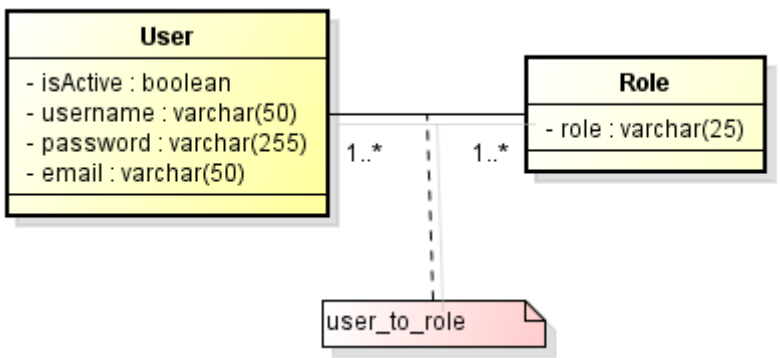


Abbildung 62: User und Role Klassen

[23] BCrypt: Kryptologische Hashfunktion, speziell für das Speichern von Passwörtern

10.6.1 Roles

Durch die Nutzung von ROLES im Symfony, lassen sich einfach gewisse Ressourcen an bestimmte Benutzer freigeben, z.B. der Zugriff über einen Controller:

```
if (false === $this->get('security.context')->isGranted('ROLE_MANAGER')) {
    throw new AccessDeniedException();
}
```

Abbildung 63: Zugriff gestatten an ROLE_MANAGER in einem Controller

Symfony erlaubt den Aufbau einer gewissen Rollen-Hierarchie, für iTIXI wurde diese adaptiert für:

Rolle	Beschreibung
ROLE_DISPO	Disponent, Benutzer der Disposition
ROLE_MANAGER	Manager, Benutzer in der Geschäftsleitung
ROLE_ADMIN	Administrator, Supporter der iTIXI Applikation

Zugriffe auf Freigaben für ROLE_DISPO sind auch ROLE_MANAGER, bzw. für die höheren Rollen möglich.

Verlangt eine Freigabe mindestens die Rolle ROLE_MANAGER, so haben tiefere Rollen wie ROLE_DISPO keinen Zugriff, höhere Rollen wie ROLE_ADMIN erhalten aber Zugriff.

```
role_hierarchy:
    ROLE_DISPO:
    ROLE_MANAGER:       ROLE_DISPO
    ROLE_ADMIN:         ROLE_MANAGER
```

Abbildung 64: ROLE Hierarchie in security.yml von Symfony

10.6.2 OAuth 2.0

Für die weitere Entwicklung der iTIXI Applikation wurde ein Symfony Bundle eingerichtet um einen OAuth 2.0 Server zu betreiben. Dies kann zukünftig gebraucht werden für API-Zugriffe von einer Buchhaltungssoftware oder eines Telefon-Systems, die keine Formular-basierte Anmeldung durchführen können. Weiter möchten wir hierbei nicht auf diese Bundle eingehen, Informationen erhält man über FOSOAuthServerBundle[24].

[24] FOSOAuthServerBundle: https://github.com/FriendsOfSymfony/FOSOAuthServerBundle

11 Deployment

Das Deployment ist nicht Teil der Aufgabenstellung, aber dennoch notwendig um einen Testbetrieb zu gewährleisten. Wie bei der kontinuierlichen Integration erwünscht, wird stets eine aktuelle Version auf einem Server ausgerollt. Auf unserem eigenen Buildserver *tixi.seiout.ch* wurde die entsprechende Konfiguration (Abbildung 65) an Technologien ausgeführt für den Betrieb von iTIXI, wie es in einer ausgerollten und produktiven Umgebung der Fall wäre.

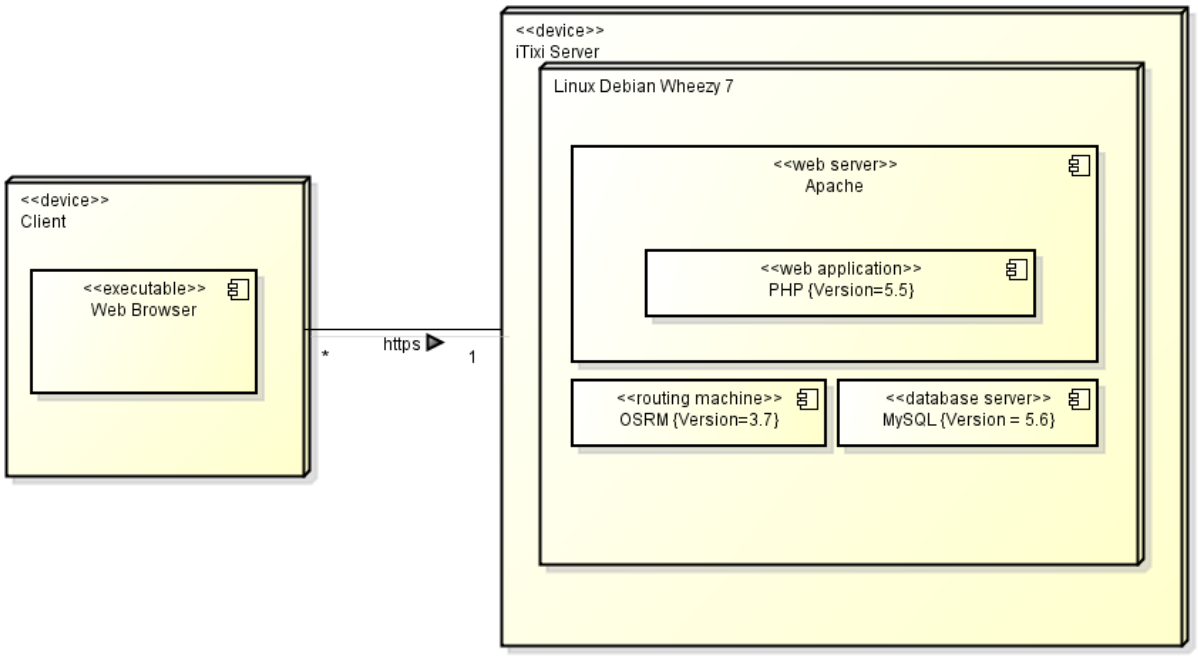


Abbildung 65: Deployment-Diagramm

Vorausgesetzt wird eine lauffähige LAMP Umgebung mit den Mindestversionen:

- PHP 5.5
- MySQL 5.6

Apache sollte über ein SSL-Zertifikat verfügen um eine HTTPS Verbindung zu ermöglichen, wie z.B. https://tixi.seiout.ch/tixi/

Speziell sind die Anforderungen an OSRM v3.7. Zusammen mit den anderen Technologien wurde dafür ein Dokument *Technologie-Deployment* erstellt, welches im Appendix (**Fehler! Verweisquelle konnte nicht efunden werden.**) verfügbar ist. Dieses Dokument wurde auch dem Projektpartner zur Verfügung gestellt für die Auswahl eines geeigneten Hosting-Providers.

11.1 Build

11.1.1 Umgebungsparameter

Auf jeder Umgebung müssen Parameter für Symfony, Bundles und weitere Libraries definiert werden.

Diese werden in der Datei *app/config/parameter.yml* definiert:

```
parameters:
    database_driver: pdo_mysql
```

```
database_host: localhost
database_port: 3306
database_name: tixi
database_user: mysql_tixi
database_password: password
mailer_transport: smtp
mailer_host: 127.0.0.1
mailer_user: null
mailer_password: null
locale: de
secret: ThisTokenIsNotSoSecretChangeIt
time_zone: Europe/Zurich
wkhtmltopdf_path: /usr/bin/wkhtmltopdf
```

Dazu gehören Zugriffsinformationen für Database und SMTP Maildienst, die Zeitzone sowie speziell für die PDF-Erzeugung eine ausführbare Binary zu wkhtmltopdf.

Durch SwiftMail kann möglichst simpel der Maildienst von Google genutzt werden. Dazu müssen nur folgende Parameter gesetzt werden:

```
mailer_transport: gmail
mailer_host: null
mailer_user: myaccount@gmail.com
mailer_password: mypassword
```

11.1.2 Applikations Parameter

Einstellungen zur iTIXI Applikation werden über die Datei *app/config/tixi.yml* definiert:

```
parameters:
     tixi_parameter_app: iTIXI
     tixi_parameter_version: v2.0.3.2
     tixi_parameter_client: TIXIZUG
     tixi_parameter_client_page: http://www.tixizug.ch

     tixi_parameter_files_directory: %kernel.root_dir%/cache/files
     tixi_parameter_admin_mail: itixizug@gmail.com

     tixi_parameter_google_apikey: AIzy…w00uAcAv-…
     tixi_parameter_osrm_server: seiout.ch:8080
```

Dazu gehören Applikationsname, Version und Mandant die im Frontend angezeigt werden.

Mit tixi_parameter_files_directory wird ein Directory bestimmt für die Ablage erzeugter Dateien (z.B. PDF Rapporte). Eine Mailadresse zu einem Administrator sollte über tixi_parameter_admin_mail bestimmt werden für die Zustellung von Informationen/Fehlern zum System. Hinzu kommen der Google API Key für den Mandanten, sowie Adresse und Port eines OSRM-Server.

11.1.3 Symfony + Composer

Alle benötigten Daten für den Buildprozess werden vom Repository aus zur Verfügung gestellt.

Für Symfony werden einige Dependencies zu PHP benötigt, zudem werden einige Bundles genutzt. Diese können komfortabel über den Composer bezogen werden. Fehlen gewisse Angaben in der parameter.yml, wird zur manuellen Eingabe dieser Parameter bei der Ausführung vom Composer aufgeboten.

11.1.4 Doctrine + Database

Um eine frische Installation anzulegen, werden Tabellen für die Datenbank per Doctrine erzeugt. Alle nachfolgenden Befehle werden dabei im Symfony Root-Verzeichnis ausgeführt. Symfony bietet über *console.php* eine Verwendung von Commands an mit der zugehörigen Umgebung.

Befehl für Database Erzeugung:

```
php app/console doctrine:database:create
```

Bei einer lauffähigen Umgebung, die Updates erhält, muss nur das Datenbankschema aktualisiert werden:

```
php app/console doctrine:schema:update –force
```

Da Doctrine über keine Angaben zur Volltext-Suche von MySQL mit InnoDB verfügt, wird bei einer Neuanlegung der Datenbank nachträglich ein Volltext-Index erzeugt über einen von uns erstellten Symfony Command:

```
php app/console project:build-fulltext
```

Dieser fügt per ALTER TABLE einen Volltext-Index zu den benötigten Kolumnen in der Adress-Tabelle hinzu.

Für den weiteren Build wurden auf unserem Buildserver jeweils noch Testdaten aus einer .SQL Datei hinzugefügt.

11.2 Technologie-Stack

Folgende Technologien werden für den Betrieb von iTIXI auf einem Linux-Server aufgeführt:

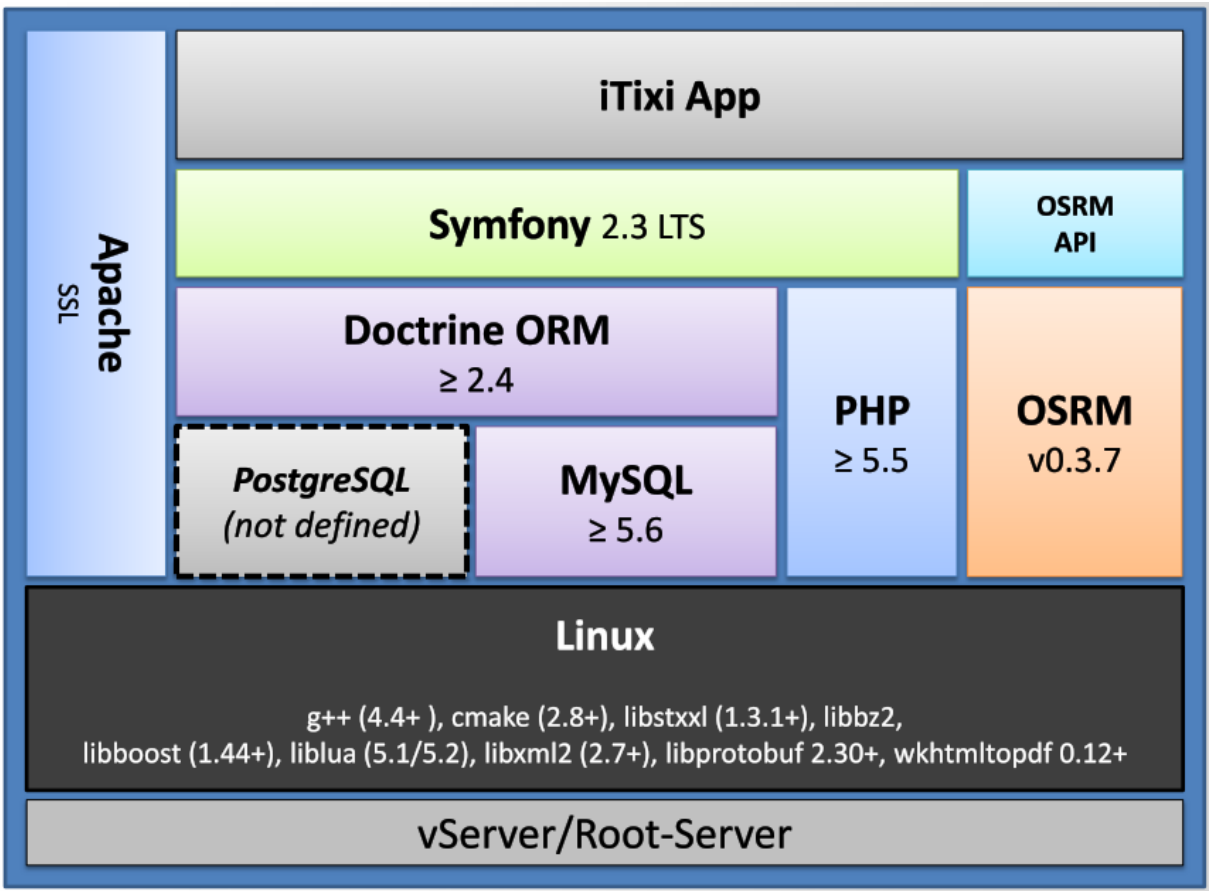


Abbildung 66: Technologie-Stack iTIXI

Dabei wäre auch PostgreSQL als Alternative denkbar und dank OR-Mapper Doctrine2 ohne Probleme austauschbar anstelle von MySQL. Nebst Parametern müsste nur die Erstellung und Verwendung der Volltextsuche unter Umständen angepasst werden.

11.2.1 OSRM Update

Da sich OSRM auf die Daten von OpenStreetMap stützt, sollten die OSM Dateien von Zeit zu Zeit aktualisiert und für OSRM neu vorberechnet werden. Auf einem Linux Server wäre dazu ein BASH-Skript geeignet, das z.B. jedes Wochenende ausgeführt wird:

```
./stop.sh      (aktuellen OSRM Prozess stoppen)

(vorhandene Datei löschen oder verschieben)
cd build && yes | rm switzerland-latest.osm.pbf

(aktuelle OSM herunterladen)
wget http://download.geofabrik.de/europe/switzerland-latest.osm.pbf

(vorhandene konvertierte OSM Dateien löschen und neu konvertieren)
yes | rm swiss.o5m && ./osmconvert switzerland-latest.osm.pbf -o=swiss.o5m

(konvertierte OSM Datei umwandeln)
yes | rm swiss.osm.pbf && ./osmconvert swiss.o5m -o=swiss.osm.pbf

(OSM Dateien für OSRM extrahieren und berechnen lassen)
./osrm-extract build/swiss.osm.pbf
./osrm-prepare build/swiss.osrm

./start.sh     (OSRM Prozess neu starten)
```

11.3 Commands

Um den Betrieb von iTIXI zu gewährleisten, werden einige Symfony Commands angeboten, die über einen Scheduler vom Betriebssystem aus angesteuert werden können. Diese sind für folgende Aufgaben gedacht:

11.3.1 Adresskoordinaten abrufen

Alle in der Datenbank vorhandenen Adressen werden auf die Vollständigkeit von Koordinaten und Nearest Points überprüft und allenfalls ergänzt über Google Maps API und OSRM.

Die könnte täglich am Abend terminiert werden. Soweit werden aber durch die Adressverwaltung alle neu erfassten Adressen immer mit den vollständigen Koordinaten erstellt. Genutzt wurde der Command vorallem bei der Daten-Migration, bzw. Integration von Testdaten, damit alle Adressobjekte über eine Koordinate verfügten.

Command: `project:query-address-coordinates %limit%`

Argument: %limit% bestimmt die maximale Anzahl abzurufender Addressen an. Default: 1000

Häufigkeit: täglich

Sendet ein Informationsmail mit der Menge an aktualisierten Adressen an den Administrator.

11.3.2 Routen aktualisieren

Da zu einem Fahrauftrag eine Route abgespeichert wird, kann es sein, dass diese mit der Zeit nicht mehr so aktuell sind. Über den Command werden Routen aus der Datenbank geladen, die älter als einen Monat sind. Diese werden anschliessend über OSRM erneut abgefragt und wenn nötig aktualisiert.

Command: `project:route-update`

Argument: keine

Häufigkeit: monatlich

Sendet ein Informationsmail mit der Menge aktualisierter Routen an den Administrator.

11.3.3 Fahrtoptimierungen vorausberechnen (angedacht)

Denkbar ist, alle Schichten und Tage über die Fahrtoptimierung bereits zu berechnen, damit in der Tagesplanung Fahraufträge bereits in einer möglichen Form aufgelistet sind und es für die Disponenten vereinfacht, einen zukünftigen Tag zu untersuchen bei der Erfassung weiterer Fahraufträge.

Command: `project:ride-optimization %day%`

Argument: %day% gibt den Tag in Form von dd.mm.YYYY an, der optimiert werden soll
Default: 01.07.2024 (so ergeben aus den Testdaten)

Häufigkeit: täglich

12 Projektmanagement

Verwendet wurde die agile Projektmethodik Scrum mit wöchentlichen Sprints und Planning Meetings, bei denen wir zusammen die weitere Entwicklung betrachteten und kommende Arbeiten aufteilten.

Nach dem fortlaufend die Arbeitspakete definiert und aufgeteilt wurden, wurden die Implementierungsarbeiten oft im Alleingang getätigt. Die Arbeitszeiten waren geprägt durch Tage, an denen Carmelo Schumacher weitere Module im Studium besuchte und Tage, an denen Teilzeitstudent Reto Schelbert im Betrieb arbeitete. Das behinderte die Arbeiten aber überhaupt nicht, es wurde nebst den Sprint Meetings eine intensive Online-Kommunikation über Hangout/Skype gepflegt.

Bis auf einige intensive Implementierungsphasen gab es wöchentliche Meetings mit dem Betreuer Stefan Keller, bei denen auch regelmässig die Projektpartner Martin Jonasse und Josi Conrad dabei waren.

12.1 Arbeitsaufwand

In diesem Modul liegt der vorgegebene Aufwand pro Person bei ca. 360 Stunden. Während KW8 bis KW24 wurden unter der Woche stets 3 Tage à 8 Stunden für die Arbeit reserviert. Effektiv wurden gerne 4 Tage in der Woche für die Arbeit investiert und in den letzten Wochen an die 7 Tage. Dieser Mehraufwand kam dadurch zustande, dass bei einigen Arbeiten der Zeitaufwand nicht gut eingeschätzt werden konnte und die hohe Komplexität der ganzen Disposition die Abschätzungen zudem erschwerten. Wir wollten mit allen Mitteln die vorgegeben Funktionalitäten erreichen. Schlussendlich erwies sich aber, dass die Aufgabenstellung zu gross ist um alles in der knappen Zeit zu erfüllen. Der geleistete Aufwand ist deutlich über dem Richtwert:

Person	Aufwand (h)
Reto Schelbert	468
Carmelo Schumacher	452

12.2 Gantt-Diagramm

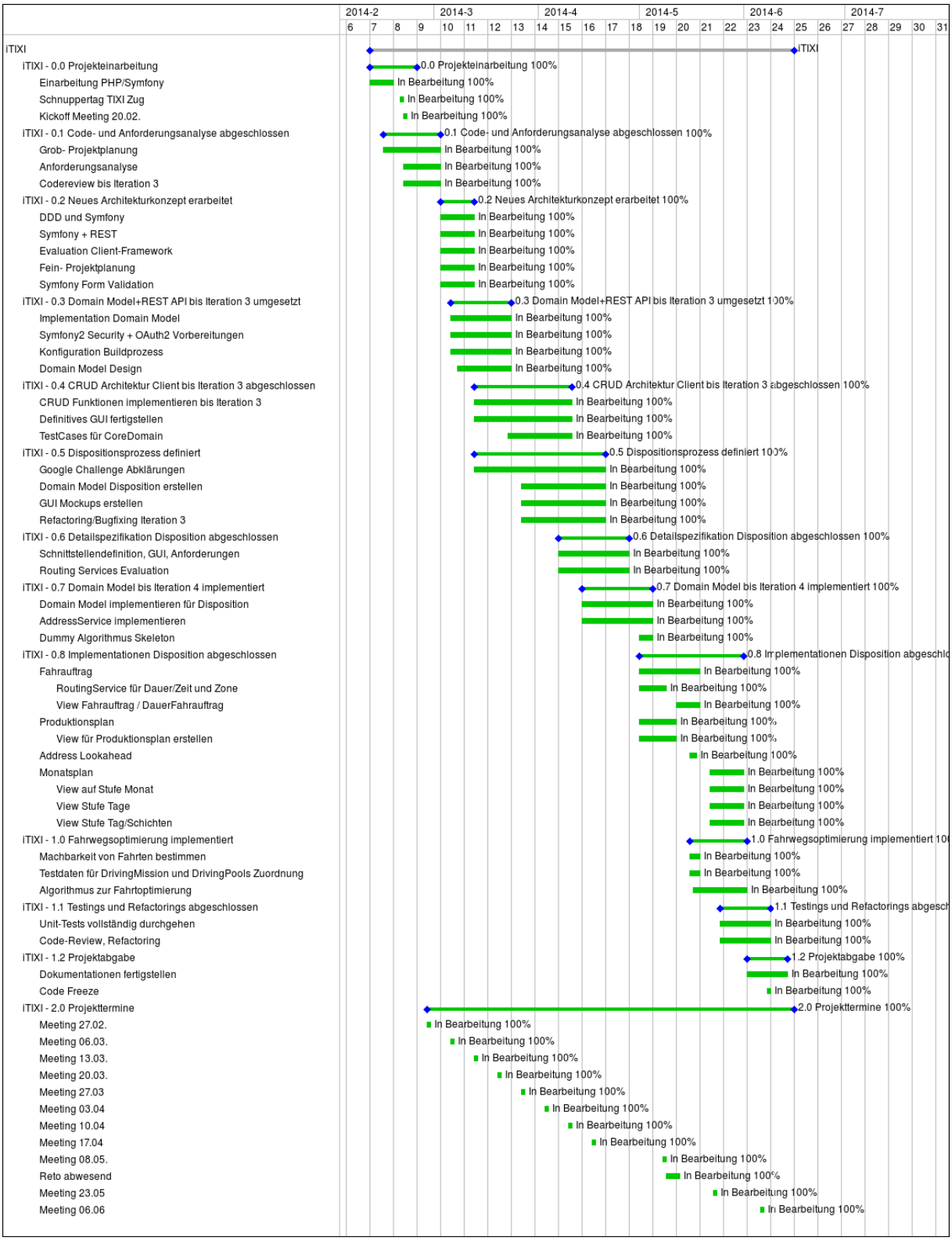


Abbildung 67: Gantt-Diagramm

12.3 Meilensteine und Arbeitspakete

Einige Meilensteine wurden durch die detaillierte Projektplanung vorgängig definiert, durch die iterativen Prozesse aber immer wieder neu bestimmt. Die Planung wurde fortlaufend geändert und neu erarbeitet.

12.3.1 MS0.0 Projekteinarbeitung

Termin: 23.02.2014

Vor dem Kickoff zu wurden bereits ein paar Tätigkeiten eingeplant

Tracker	Thema	Beschreibung	Beginn
Project	Einarbeitung PHP/Symfony		10.02.2014
Project	Schnuppertag TIXI Zug		19.02.2014
Project	Kickoff Meeting 20.02.		20.02.2014

12.3.2 MS0.1 Code- und Anforderungsanalyse abgeschlossen

Termin: 02.03.2014

Der bestehende Code des TIXI Partners wird soweit untersucht sein und es sollen alle Anforderungen klar und definiert sein für die bevorstehenden Arbeiten mit einer vorhandenen Grob-Projektplanung.

Tracker	Thema	Beschreibung	Beginn
Project	Grob- Projektplanung		14.02.2014
Require	Anforderungsanalyse		20.02.2014
Require	Codereview bis Iteration 3	Review mit Projektpartner	20.02.2014

12.3.3 MS0.2 Neues Architekturkonzept erarbeitet

Termin: 12.03.2014

Aus MS0.1 wurde klar, dass der vorhandene Code nicht ausreichend für unsere weiteren Arbeiten ist. Gemäss Meeting 02.20 wurde der Entscheid beschlossen eine neue Grundlage zu erarbeiten und das TIXI Projekt bis Iteration 3 soweit wieder zu implementieren.
Für MS0.2 soll aus dem bestehenden Code ein neues Konzept erarbeitet werden, sowie Spezifikation DDD/ORM Backend und Frontend.

Tracker	Thema	Beschreibung	Beginn
Require	DDD und Symfony	Logische Architekturziele	03.03.2014
Require	Symfony + REST		03.03.2014
Require	Evaluation Client-Framework		03.03.2014
Project	Fein- Projektplanung		03.03.2014
Require	Symfony Form Validation	Konzept erarbeiten	03.03.2014

12.3.4 MS0.3 Domain Model+REST API bis Iteration 3 umgesetzt

Termin: 23.03.2014

Architekturentscheide sollen ausgearbeitet werden für die Implementierung, anhand dieser kann die Projektplanung verfeinert werden.

Tracker	Thema	Beschreibung	Beginn
Require	Implementation Domain Model		06.03.2014
Require	Symfony2 Security + OAuth	Konzept erarbeiten	06.03.2014
Require	Konfiguration Buildprozess	Zu Continous Integration	06.03.2014
Require	Domain Model Design		08.03.2014

12.3.5 MS0.4 CRUD Architektur Client bis Iteration 3 abgeschlossen

Termin: 10.04.2014

Die Funktionalitäten, welche bis Iteration 3 vorhanden waren, werden bis MS0.4 implementiert. Abschliessend dazu wird die Baseline Version 2.8 für einen Review dem Projekt-Partner übergeben.

Tracker	Thema	Beschreibung	Beginn
Require	CRUD Funktionen implementieren bis Iteration 3		13.03.2014
Require	Definitives GUI fertigstellen	TWIG Templating	13.03.2014
Require	TestCases für CoreDomain		23.03.2014

12.3.6 MS0.5 Dispositionsprozess definiert

Termin: 20.04.2014

Bis MS0.5 sollten für die Disposition noch offenen Punkte und Anforderungen geklärt werden, dazu gehört die „Google Challenge“ wegen der Menge an Routen-Abfragen. Dies wurde später durch die Verwendung von OSRM gelöst.

Tracker	Thema	Beschreibung	Beginn
Require	Google Challenge Abklärungen	Limitationen, Alternativen	13.03.2014
Require	Domain Model Disposition erstellen		27.03.2014
Require	GUI Mockups erstellen	Für Disposition View	27.03.2014
Project	Refactoring/Bugfixing Iteration 3	Gemäss Issues Github	27.03.2014

12.3.7 MS0.6 Detailspezifikation Disposition abgeschlossen

Termin: 27.04.2014

GUI Mockups und Schnittstellen sollten für die Implementation der Dispositions-Funktionalitäten erstellt sein und die zu verwendende Routing Lösung (OSRM) evaluiert.

Tracker	Thema	Beschreibung	Beginn
Require	Schnittstellendefinition, GUI, Anforderungen	Für Dispositions-Prozesse	07.04.2014
Require	Routing Services Evaluation	OSRM konfigurieren	07.04.2014

12.3.8 MS0.7 Domain Model bis Iteration 4 implementiert

Termin: 04.05.2014

Implementierung gemäss erarbeiteten Anforderungen und Schnittstellen zur Disposition.

Eine erste Version eines Algorithmus soll erstellt werden (zum diesem Projektzeitpunkt war noch nicht klar, ob genügen Ressourcen vorhanden waren für einen Optimierungs-Algorithmus, deshalb ging man von einer vereinfachten Version aus, man nannte diesen Algorithmus in unserer Projektplanung einen „Dummy“)

Tracker	Thema	Beschreibung	Beginn
Require	Domain Model implementieren für Disposition		14.04.2014
Require	AddressService implementieren	Für Addressverwaltungs-Konzept	14.04.2014
Require	Dummy Algorithmus Skeleton	Einfacher Algorithmus für Fahrtoptimierung	01.05.2014

12.3.9 MS0.8 Implementationen Disposition abgeschlossen

Termin: 31.05.2014

Die Frontend Implementierungen zur Disposition sollten abgeschlossen sein. Dazu gehören einige Services zum routing und Address-Lookahead.

Tracker	Thema	Beschreibung	Beginn
Require	Fahrauftrag		01.05.2014
Require	Produktionsplan		01.05.2014
Require	View für Produktionsplan erstellen		01.05.2014
Require	RoutingService für Dauer/Zeit und Zone		01.05.2014
Require	View Fahrauftrag / DauerFahrauftrag		12.05.2014
Require	Address Lookahead	mit manueller Bearbeitung und Koordinaten-Abfrage Google Maps	16.05.2014

Require	Monatsplan		22.05.2014
Require	View auf Stufe Monat	Monatspläne	22.05.2014
Require	View Stufe Tage	Zusammenzug aller Tage im Monatsplan mit Statistik, Status	22.05.2014
Require	View Stufe Tag/Schichten	Zusammenzug aller Schichten mit Status über zugewiesene Fahrer	22.05.2014

12.3.10 MS1.0 Fahrwegsoptimierung implementiert

Termin: 01.06.2014

Parallel zu MS0.8 wurde an einem möglichen Algorithmus für die Fahrwegoptimierung gearbeitet.

Tracker	Thema	Beschreibung	Beginn
Require	Machbarkeit von Fahrten bestimmen		16.05.2014
Require	Testdaten für DrivingMission und DrivingPools Zuordnung	Zufällige Erstellung von Fahraufträgen	16.05.2014
Require	Algorithmus zur Fahrtoptimierung		17.05.2014

12.3.11 MS1.1 Testings und Refactorings abgeschlossen

Termin: 08.06.2014

Die Implementierungen sollten soweit abgeschlossen mit Systemtests und Code-Analyse mit eventuellem Refactoring.

Tracker	Thema	Beschreibung	Beginn
Require	Unit-Tests vollständig durchgehen		25.05.2014
Require	Code-Review, Refactoring		25.05.2014

12.3.12 MS1.2 Projektabgabe

Termin: 13.06.2014

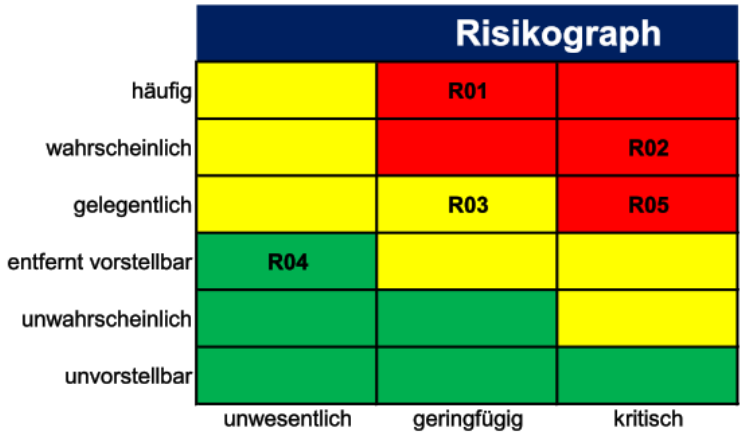
Dokumentationen, besonders zur Implementierung, fertig stellen bis zur Projektabgabe.

Tracker	Thema	Beschreibung	Beginn
Require	Dokumentationen fertigstellen		02.06.2014
Project	Code Freeze		08.06.2014

12.4 Risikomanagement

12.4.1 Risikobewertung

Risiko ID	Risiko	Auswirkung	Massnahme	Kosten der Massnahmen in Tagen	Max. Schaden in Tagen	Wahrschein-lichkeit des Eintreffens	Gewichteter Schaden Tagen	Priorität
R01	**Anforderungen unklar**	Es entsteht ein Mehraufwand	Reviews mit Kunden, Requirement Engineering	3	6	80%	**4.80**	hoch
R02	**Code Iteration 3 nicht den Anforderungen entsprechend**	Hinderniss für Weiterentwicklung	Refactoring bestehender Code	21	30	60%	**18**	hoch
R03	**Fahrwegsoptimierung nicht performant**	Umstellung in der Dispositionsplanung mit Optimierung	Disposition ohne Optimierung in Betracht zieehen. Performancemessungen und Algorithmus Prototype frühzeitig angehen.	6	10	40%	**4.00**	mittel
R04	**Akzeptzanz GUI für Dispositionsplanung nicht vorhanden**	Implementationsänderungen GUI Client	Reviews mit Kunden, Usability Kriterien einhalten, Interaction Design miteinplanen	8	12	40%	**5**	niedrig
R05	**Google Maps Nutzung eingeschränkt, rechtliche oder lizenztechnische**	Fahrwegsoptimierung, Routenplanung nicht wie gewünscht durchführbar	Googles Term of Service Abkärungen mit Kunden, Lizenzkauf miteinbeziehen	2	8	40%	**3.20**	mittel
	Total Kosten in Arbeitspaketen enthalten			**40**				
	Total Rückstellungen						**35**	



R01: Es wurde stets eine sehr intensive und regelmässige Kommunikation mit dem Projektpartner gepflegt, die Anforderungen konnten immer geklärt werden dank Reviews, Issues-Tracking und Meetings.

R02: Ist zu Beginn des Projekts leider eingetroffen. Die Aufgabenstellung wurde dementsprechend angepasst, gegenüber der ursprünglichen Planung ergab sich ein Defizit von gut 5 Wochen.

R03: Die Implementierung eines möglichen Fahrwegoptimierungs-Algorithmus wurde mit einer falschen Annahme zuerst etwas verzögert, es wurde aber schnell reagiert in Rücksprache mit dem Betreuer.

R04: Dank den Reviews und fortlaufenden Tests durch den Projektpartner gut eingedämmt.

R05: Durch die Wahl von OSRM wurde das Risiko umgangen.

13 Projektstand

Worte zum Projektablauf

Selten war die Lernkurve auf unserer Seite so gross wie bei der Durchführung dieser Bachelorarbeit. Ursprünglich geplant war eine Projektübernahme gemäss Aufgabenstellung ab ID F-070, wir entschieden uns nach einer genaueren Analyse aber für eine völlige Neuentwicklung, die Aufgabenstellung wurde dementsprechend angepasst. Dieser Entscheid bedeutete einen grossen Mehraufwand, der aus unserer Sicht aber immer noch realistisch schien im Rahmen dieser Arbeit durchzuführen. Leider waren unsere Aufwandschätzungen des Öfteren zu gering - die ganze Komplexität dieses Projekts wurde unterschätzt – der zeitliche Rahmen war zu eng. Mit einer grossen Aufopferung an Zeit und persönlichen Ressourcen wollten wir eine komplette und vollständige Lösung abliefern die allen Anforderungen gerecht wird, räumten aber nicht mehr machbare Funktionen, wie die der Tagesplanung, viel zu spät ein.

Absenz

Hinzu kamen gesundheitliche Probleme von Carmelo Schumacher in den letzten Tagen vor Abgabe, weshalb einige Kapitel in der Projektdokumentation nicht mehr vervollständigt oder gar ausgelassen wurden.

Ergebnisse

Aus unserer Sicht wurde durch die Neuentwicklung der iTIXI Applikation eine nützliche Abbildung der Geschäftslogik erreicht unter Beachtung von massgebenden objekt-orientierten Ansätzen und sinnvollen Strategieentscheiden beim architektonischen Aufbau. Mit einer eigenen Instanz von OSRM und einem Adressverwaltungskonzept kamen Funktionen hinzu, an die zu Beginn des Projekts kaum jemand gedacht hätte. Die Applikation als Informationssystem bietet dem TIXI Betrieb bereits ein grosses Mass an Verwaltungswerkzeugen für die Stammdatenverwaltung an und lässt sich einfach und ohne grosse Abhängigkeiten erweitern. Hinzu kommen viele erreichte Teilsysteme wie das DataGrid, das Tile-Konzept, ein ID-Konzept für Frontend-Tests und die Routen-Abfrage, die allesamt einen weiteren Entwicklungsprozess begünstigen. Die bereits verwirklichte Datenstruktur der zum Domain Model zugehörigen Entitäten bietet eine durchdachte Basis für die Dispositionsprozesse an, insbesondere der Tagesplanung. Die vollständige Realisierung aller den Dispositionsprozessen zugehörigen Funktionen konnte jedoch nicht erreicht werden, dies betrifft Auftragsverwaltung und die Tagesplanung an sich. Mit einem möglichen Algorithmus zur Fahrtoptimierung wurde soweit aber ein weiterer Punkt konzeptionell abgedeckt.

Bewertung

Die Zeitplanung im Projektmanagement war nur hinreichend gut, hierzu zählen Fehleinschätzungen über den Aufwand gewisser Implementationsschritten die unter anderem durch fehlende Erfahrung zustande kamen. Wegen dem entstandenen Zeitmangel konnte gegen Ende des Projekts kein ausführliches Test-Engineering mehr betrieben werden. Eine nicht ganz vollständige Lösung bis zum Abgabezeitpunkt zu erreichen missfällt uns sehr und wir werden auch nach Abgabe der Bachelorarbeit mit dem Projektpartner weitere Schritte in die Wege leiten. Die bisher erreichten und abgegebenen Versionen wurden soweit aber mit grosser Begeisterung begutachtet und wir sind sicher, mit der geleisteten Arbeit bis vor Abgabe einen guten Eindruck hinterlassen zu haben.

Probleme / Offene Punkte

Einige offene Punkte möchten wir zum Projektstand deshalb vorweg erwähnen:

- Die Fahrauftragserfassung ist nicht vollständig. Frontendabläufe sind zum Erfassen und Editieren vorhanden, ausser die Erstellung einer Rückfahrt. Ein Prozess zum Change-Management, ähnlich dem von Fahrteinsätzen, fehlt.
- Frontend und Backend für die Erstellung von Sammelaufträgen fehlt, dies wurde aber im Vorhinein schon als nicht zu implementierende Funktion abgehackt.
- Ein Frontend zur Tagesplanung fehlt ganz, dies aber gemäss Anforderungen in der Aufgabenstellung.

Empfehlungen

Für die Weiterentwicklung wird ein intensiver Implementierungs-Sprint nötig sein um die Frontend-Arbeiten für die Tagesplanung zu erstellen. Einige Optimierungsgedanken kamen uns auch während der Arbeit auf: so wäre es denkbar in Zukunft unter anderem zur Routen-Berechnung eine noch geeignetere Lösung zu verwenden mit Database-Routing wie es pgRouting zu PostgreSQL anbietet, dazu gehören Funktionen die weitere Performancesteigerungen zur Fahrtoptimierung bringen könnte.

14 Erfahrungsbericht

14.1 Reto Schelbert

Die Arbeit am iTIXI Projekt, insbesondere die ausführliche Kommunikation zum Projektpartner, hat mit sehr gefallen. Mit Carmelo im Team konnte ich viel profitieren im Hinblick auf die Webentwicklung, ein Gebiet auf dem ich vorher noch keine Projekte verwirklichte. Gerade die gemeinsam erarbeiteten Architekturentscheide und die Anwendung vieler objekt-orientierter Ansätze wiederspiegeln viele Aspekte dir mir durch das Studium an der HSR vermittelt wurden.

Die Arbeit hat viele lehreiche Eindrücke hinterlassen. Nicht zuletzt lernt man insbesondere die Erfahrungen wertschätzen, die notwendig sind um in einer bestimmten Technologie gute Zeitschätzungen zu machen für benötigte Implementierungsarbeiten. Es war deshalb auch eine sehr intensive, wenn nicht auch stressige Zeit.

Die Zusammenarbeit verlief stets positiv und ich möchte mich herzlichst bei Stafen Keller und den Projektpartnern Martin Jonasse und Josi Conrad bedanken. Ein besonderer Dank gilt auch der TIXI Geschäftsleitung Yeldez Gwerder für die Möglichkeit an einem realen Projekt zu arbeiten und insbesondere noch für einen Verein der sich dermassen solidarisch und gemeinnützig engagiert.

14.2 Carmelo Schumacher

Aus gesundheitlichen Gründen konnte CS keinen Erfahrungsbericht mehr abliefern

15 Glossar

Das Glossar umfasst Abkürzungen und Begrifflichkeiten, die speziell im Kontext dieser Arbeit verwendet wurden.

15.1 Domain

Begriff	Term	Beschreibung
GL	Manager	Abkürzung für Geschäftsleitung
Auftrag/Fahrauftrag	Driving Order	Eine Fahrt, die von einem Kunden/Fahrgast in Auftrag gegeben wurde.
Sammelauftrag	Collective Order	Werden mehrere Aufträge werden in einer Fahrt erledigt, resultiert daraus ein Sammelauftrag
Disposition	Disposition	Einteilung von Aufträgen mit aktuellen Leistungsanforderungen. Fahraufträge werden zu Fahrern und Fahrzeugen zugeordnet.
Disponent		Mitarbeiter, der die Disposition durchführt oder dazu beiträgt.
Fahrer	Driver	Freiwilliger Mitarbeiter der TIXI, der Fahrten mit einem Fahrzeug durchführt
Fahrzeug	Vehicle	Transportmittel, das für die Ausführung eines Auftrags benötigt und einem Fahrer zugeteilt wird.
Fahrgast	Passenger	Kunde, Auftragsgeber der TIXI, hat Anrecht auf einen Transport.
Fahrt	Ride	Ein Auftrag oder Sammelauftrag, welcher ausgeführt wird über eine Route.
Route	Route	Eine Route bestimmt eine Fahrt von A nach B. Nicht zu verwechseln mit @Route bei Symfony
Leerfahrt	Empty Ride	Fahrt, welche zwischen zwei Aufträgen nötig ist.
Fahreinsatz	Driving Mission	Entspricht dem Einsatz eines Fahrers, der zu einer bestimmten Schicht fährt.
Schicht/Arbeitsschicht	Shift	Definiert einen Zeitraum während des Tages, an dem Einsätze geleistet werden.
Produktionsplan	Production Plan	Dient zur Ressourcenplanung und gibt die Menge an Fahrzeugen pro Tag und pro Schicht an, die benötigt werden.
Monatsplan	Monthly Plan	Dienst zur Ressourcen- und Einsatzplanung und ordnet Fahrer für die Bereitschaft an einem Tag und Schicht zu.
Einsatzplan		Dokument, welches die genaue Zuteilung von Fahrern zu Tag und Schicht eines Monates angibt.
Tagesplan	Daily Plan	Ergibt sich aus der Disposition über einen Tag hinweg. Im Tagesplan werden pro Schicht die zeitlichen Abfolgen von Aufträgen und Sammelaufträgen mit den zugeteilten Fahrern und Fahrzeugen angezeigt.
Zone	Zone	Eine Zone dient dem TIXI Betrieb für die Verrechnung eines Auftrags. Vom Kanton werden Fahrten innerhalb der Kantonsgrenzen subventioniert, dies entspricht der Zone = kantonal. Fahrten ausserhalb dieser Zone werden z.B. mit höheren Kosten verrechnet.
Zonenplan	Zone Plan	Ein Zonenplan besteht aus Ortschaften die je einer Zone zugeordnet werden. Im Falle der TIXI wird der Zonenplan über die Ortsnamen erstellt.
OVI	POI	OVI = Ort von Interesse, POI = Point of Interest
Fahrtoptimierung	Optimize	Ermöglicht die Berechnung idealer Fahrten zu einem bestimmten Kriterium. Für die TIXI Betriebe wird ein globales Minimum an entstehenden Leerfahrt-Kilometer angestrebt.

15.2 Weitere Begriffe

Begriff	Beschreibung
VRP	Vehicle Routing Problem
DARP	Dial-a-Ride Problem
PDP	Pickup and Delivery Problems
OSRM	Open Source Routing Machine
OSM	OpenStreetMap
JSON	JavaScript Object Notation, kompaktes Datenformat für den Datenaustausch zwischen zwei Anwendungen.
JSONP	JSON mit Padding: ermöglicht Übertragung von JSON über Domaingrenzen hinweg ohne einhalten der Same Origin Policy
DataGrid	Ein Steuerelement, welches Daten in einem definierten Raster anzeigt.
Contraction Hierarchies	Vorberechnete Version eines verbundenen Graphen um die Shortest Path Routing Berechnung zu beschleunigen.
BPMN	Business Process Model and Notation dient der Visualisierung von Business Prozessen.
Apache	Apache ist ein quelloffener und freier Webserver der Apache Software Foundation und wird unter der Apache-Lizenz vertrieben. http://httpd.apache.org
MySQL	Ein relationales Datenbankverwaltungssystem
InnoDB	Freies Speichersubsystem für MySQL, das Transaktionssicherheit und referentiellen Integrität über Fremdschlüssel gewährleistet.
Bootstrap	Bootstrap ist eine Ansammlung von HTML, CSS, JavaScript Vorlagen für Typografie, Formulare, Buttons, Tabellen, Grid-Systeme und Navigationselemente. Bootstrap ist als freie Software unter der Apache-Lizenz erhältlich. http://getbootstrap.com
CSS	Cascading Style Sheet: Deklarative Sprache für Stilvorlagen von strukturierten HTML-Dokumenten
DBAL	Database Abstraction Layer
Doctrine2	Ein in PHP geschriebener Object Relational Mapper (ORM). Setzt auf Database Abstraction Layer (DBAL). http://www.doctrine-project.org
DOM	Document Object Model: Spezifikation einer Schnittstelle für den Zugriff auf HTML-Dokumente.
GUI	Graphical User Interface
HTML	Hypertext Markup Language: Textbasierte Auszeichnungssprache zur Strukturierung von Webseiten
jQuery	Freie JavaScript-Library, die Funktionen zur DOM-Navigation und -Manipulation zur Verfügung stellt. jQuery steht unter der MIT-Lizenz. https://jquery.com
LAMP	**L**inux **A**pache **M**ySQL **P**HP/**P**ython: Programmkombination, welche im Sinne einer Software-Distribution eine Infrastruktur definiert, in deren Rahmen dynamische Webseiten und Anwendungen entwickelt und bereit gestellt werden können.
MVC	Model-View-Controller
ORM	Object Relational Mapper
PHP	PHP: Hypertext Preprocessor, Programmiersprache die hauptsächlich zur Erstellung dynamischer Webseiten oder Webapplikationen verwendet wird. PHP wird als freie Software unter der PHP-Lizenz verbreitet.
SPL	Standard PHP Library
PHPDoc	Tool zur Dokumentation von PHP-Code.
PHPUnit	Testingframework für PHP-Code.
Symfony2	Symfony2 ist ein Webapplikations-Framework, das in der Programmiersprache PHP geschrieben ist und als freie Software unter der MIT-Lizenz verbreitet wird.
UTC	Coordinated Universal Time: Koordinierte Weltzeit
Ajax	Asynchronous JavaScript and XML: Konzept der asynchronen Datenübertragung zwischen Browser und Server.

SMB	Server Message Block: Kommunikationsprotokoll für Dateizugriff von Microsoft.
OPCache	OPCache beschleunigt die Verwendung von PHP mit der Verwendung von vorkompiliertem Skript Bytecode.
BCrypt	Kryptologische Hashfunktion, speziell für das Speichern von Passwörtern. Ab PHP Version 5.5 in der SPL enthalten.
@Route	Bei Symfony ein Mapping einer URL zu einem Controller
Bundle	Die Funktionalitäten von Symfony werden durch sogenannte Bundles erweitert, dies können auch PHP Libraries sein die über Bundles mit dem Symfony Framework genutzt werden können und einem PHP Namespace gleichkommen.
Entity	Entität, in der Datenmodellierung ein eindeutig zu bestimmendes Objekt
Repository	Verwaltetes Verzeichnis zur Speicherung und Beschreibung von Entitäten
Twig	Twig ist eine freie Template Engine für PHP und steht unter der MIT-Lizenz.
Composer	Composer ist ein Dependency Manager für PHP
wkhtmltopdf	Tool, das die Erzeugung von PDF Dateien aus HTML Code ermöglicht.

16 Literaturverzeichnis

[1] G. B. D. a. J. H. Ramser, «The Truck Dispatching Problem,» 1959.

[2] G. S. J.F.Cordeau, «Recent Models Algorithms for the One-To-One Pickup and Delivery Problems,» Springer, 2008.

[3] G. L. Jean-François Cordeau, «The Dial-a-Ride Problem (DARP): Variants, modeling issues and algorithms,» Springer, 2002.

[4] C. D. G. M. P. V. S. Kirkpatrick, «Optimization by Simulated Annealing,» Science (American Association for the Advancement of Science), 1983.

[5] D. R. M. F. E. H. R. M. R. S. M. Fowler, «Patterns of Enterprise Application Architecture,» Addison Wesley, 2002.

17 Abbildungsverzeichnis

Abbildung 1: Grober Arbeitsablauf TIXI Zentrale 14
Abbildung 2: Fahrereinsätze Monatsplanung 15
Abbildung 3: Einsatzplan Ergänzungen und Änderungen von Hand 16
Abbildung 4: Fahrauftragserfassung (Quelle: iTIXI_GUI.ppt auf Sourceforge) 17
Abbildung 5: Tagesansicht Disposition (Quelle: iTIXI_GUI.ppt auf Sourceforge) 18
Abbildung 6: Informationsblatt Fahrauftrag 19
Abbildung 7: Vollständiges Use-Case Diagramm 22
Abbildung 8: BPM Monatsplan Übersicht 31
Abbildung 9: BPM Monatsplan Teil 1 31
Abbildung 10: BPM Monatsplan Teil2 32
Abbildung 11: BPM Tagesdisposition Übersicht 33
Abbildung 12: BPM Tagesdisposition Teil 1 33
Abbildung 13: BPM Tagesdisposition Teil 2 34
Abbildung 14: BPM Tagesdisposition Teil 3 34
Abbildung 15: BPM Tagesdisposition Teil 4 35
Abbildung 16: Mögliche RideConfiguration für ein Fahrzeug 38
Abbildung 17: Adjazenzmatrix durchführbarer RideNodes 39
Abbildung 18: Konfigurationsmöglichkeiten bei 3 RideNodes 39
Abbildung 19: Konfigurationsmöglichkeiten bei 4 RideNodes 39
Abbildung 20: Machbare Konfiguration mit 3 Fahrzeugen aus gleichem Depot 40
Abbildung 21: Continous Integration Build-Prozess 46
Abbildung 22: Aufbau Testklassen 46
Abbildung 23: Beziehung Komplexität und Entwicklungsaufwand (Fowler [5], S.42) 47
Abbildung 24: Domain Model Gesamtübersicht 49
Abbildung 25: Domain Model Stammdatenverwaltung 50
Abbildung 26:Domain Model Fahrauftrag 51
Abbildung 27: Domain Model Ressourcenplanung 52
Abbildung 28: Domain Model Fahrerzuteilung 53
Abbildung 29: Systemgesamtsicht 54
Abbildung 30: Sequenzdiagramm Controllerprozess 56
Abbildung 31 Illustration des Data Transfer Object (DTO)-Konzepts 57
Abbildung 32: Menükonzept 65
Abbildung 33: View Fahrer – Master-DataGrid 66
Abbildung 34: View Fahrer – Detail-View 67
Abbildung 35: ID-Konzept Master-View 68
Abbildung 36: ID-Konzept Detail-View get 69
Abbildung 37: ID-Konzept Detail-View edit 69

Abbildung 38: Adresseingabemaske - Adressfeld 71
Abbildung 39: Adresseingabemaske - Adressvorschläge 71
Abbildung 40: Adresseingabemaske - Adressanzeige 72
Abbildung 41: Adresseingabemaske - Adressauswahl 72
Abbildung 42: Adresseingabemaske - Adresse manuell editieren 72
Abbildung 43: Adresseingabemaske - Manuelle Adresseingabe 73
Abbildung 44: Das Adress-Umsystem 73
Abbildung 45: Die Komponenten des Adress-Service 74
Abbildung 46: ProductionPlan Service Komponenten 76
Abbildung 47: View - Produktionsplan eröffnen 77
Abbildung 48: Produktionsplan editieren 78
Abbildung 49: RouteManagement und RoutingMachine 80
Abbildung 50: Detail Route und Address Beziehung 81
Abbildung 51: RoutingManagementController Zugriff 82
Abbildung 52: cURL Request mit Parametern für OSRM Abfrage 84
Abbildung 53: cURL single Request OSRM 84
Abbildung 54: cURL multi Request OSRM 85
Abbildung 55: RideManagement Klassen Übersicht 86
Abbildung 56: Aufbau der Rideconfigurations 87
Abbildung 57: Beispiel DrivingMissions in Schicht2 87
Abbildung 58: Zuteilung Fahrer/Fahrzeuge zu Fahrten 89
Abbildung 59: DrivingMission zu DrivingPool 89
Abbildung 60: Print einer optimierten Schicht 91
Abbildung 61: Authentisierung und Authorisierung gemäss Symfony (Quelle: symfony.com) 92
Abbildung 62: User und Role Klassen 92
Abbildung 63: Zugriff gestatten an ROLE_MANAGER in einem Controller 93
Abbildung 64: ROLE Hierarchie in security.yml von Symfony 93
Abbildung 65: Deployment-Diagramm 94
Abbildung 66: Technologie-Stack iTIXI 97
Abbildung 67: Gantt-Diagramm 101

18 Inhalt der CD

01-Dokumente

- Arbeitsbericht
- Sourceforge
    - Spezifikationsliste iTIXI_SRS_Liste.xlsm
    - Anforderungsdokument iTIXI_SRS.docx
- Poster

02-Abbildungen

- Domain Model Grossformat

03-Sourcecode

- Symfony Projekt als .ZIP Datei

04-PHPDoc

- PHPDoc Code-Dokumentation im HTML-Format
    - Einstieg über index.html

19 Appendix

19.1 Sitzungsprotokolle

Projekt – Sitzungsprotokoll

SITZUNGSPROTOKOLL

Vom 20.02.2014

1 Beteiligte

Stefan Keller, Reto Schelbert, Carmelo Schumacher, Martin Jonasse, Josi Conrad

2 Traktanden

Kickoff

Projektübergabe Tixi

Besprechung Vorgehensweise

3 Diskussion / Beschlüsse

Projektanforderungen	TIXI wächst, Anforderungen gerecht werden? 2 Telefonlinien, Zeitgleiches arbeiten bedingt Realtime Applikation? Lauf Stefan Keller nicht kein Realtime nötig, RESTful über Transaktionen im Hintergrund am gleichen Dokument arbeiten.
Projekt/Rahmenbedingungen	3min Video nicht zwingend, Technologiestack LAMP zwingend. In Kauf genommen, dass Lösung nicht 100% funktionieren muss. Auch Fahrtoptimierung kann 2-3min gehen, ist recht reduziert gemäss VBS im Excel von Martin. Das Bereitstellen von Dokumenten ist ausserhalb der Aufgabenstellung (Statistik, Rapporte, Fahrtplan, Einsatzplan etc.)
Software Architektur	Langlebigkeit wichtig, deshalb nicht bereits angefangen mit Doctrin -> decoupled vom Symfony Framework und wenige Entwickler dahinter? Deshalb wurde nur DBAL von Doctrine verwendet, sei simpel. DBAL sehen wir jedoch genauso als Abhängigkeit.
Latin1	DB alles Latin1 – sonst sei ÄöÜ nicht korrekt sortiert. UTF8 sollte dies jedoch abdecken und wird vermehrt verwendet.
Vorstellung von Martin	Im Auftrag/POI googlen für Benutzer zur zum Verifizieren, ob diese nutzbar sind. Geocode von Google kann angepasst werden durch Benutzer, auch aus dem Grund, weil es Fahrgäste gibt die im Niergendwo wohnen, gar keine Hausnummer haben und Google u.U. keine Adresse dafür hat, aber so mit dem Geocode rechnen kann.
DB Mapping	Feldnamen in MySQL deutsch, diese erscheinen momentan 1 zu 1 im Formular -> sollte man mappen, Mapping Tabelle – Aufgabe für uns.
Parameter Symfony	Globale werte tixi werden intensiv genutzt welche in der parameters.yml liegen, die müssen wir soweit noch ändern.
Google Maps Challenge	Anzahl Abfragen speichern/zählen Fallbackszenario – wenn Google API nicht mehr erreichbar. Routen abspeichern, viele wiederkehrende Routen, Anzahl an Googles API abfragen wird kleiner.

4 Aufgaben

- OR-Mapper aufzeigen, was es kann und wo es besser ist
- Form/Table-Mapper für MySQL
- Account anmelden Sourceforge
- Architektur bestimmen und mit Projektplanung beginnen

12. Juni 2014 Seite **1** von **1**

Projekt – Sitzungsprotokoll

SITZUNGSPROTOKOLL

Vom 27.02.2014

1 Beteiligte

Stefan Keller, Reto Schelbert, Carmelo Schumacher

2 Traktanden

- Ablauf BA
- Projektplanung
- Architekturumstellungen (Domain Model)
- OR-Mapper
- Rich Client Tagesplanung (Technologie für Umsetzung)
- Administratives
- GIS Funktionen MySQL/Postgres

3 Diskussion / Beschlüsse

Requirementsanalyse	Durch Schnuppertag entstenaden
Codeanalyse	Doctrine gehört zu Symfony als Gesamtpaket, sollte kein Hindernis sein.
Tagesplanung	Rich Client für Tagesplanung muss mit oder ohne Anpassungen durch Benutzer bereits realisiert werden, Darstellung muss in beiden Fällen implementiert sein einfach mit oder ohne „Editor". Implementierung also durchaus möglich ohne Kenntnisse über den wirklichen Ablauf der Fahrtoptimierung zu haben. Multiuser Tagesplanung? -> Realtime nicht zwingend erforderlich.
Google API	Verträglichkeit überprüfen wegen Cache und abspeichern etc. Distance Matrix - auf Google Maps darstellen (Routen), alles andere nicht erlaubt. Darstellung öffentlich erlaubt? Evtl. schwierig da sensible Daten – Patienten Adresse auffindbar.
OSRM	Alternative über OSRM? OpenStreetMap für Datenbeschaffung. Online-Service ist aber sicher nicht auf grosse Abfragen ausgelegt, Performance, und ausserdem zur Verfügung gestellt von freiwilligen – wie lange hält das Bestand?
Clustering für Gebiete	Clustering von den Gebieten so nicht möglich/nützlich -> GIS-Frage geklärt

4 Aufgaben

- Eintrag von Administrativen Links ins Wiki
- Refactoring bestehender Code durchführen und nächstes Meeting Beispiel zeigen für Entwickler
- Projektplanung und Dokumente liefern

12. Juni 2014 Seite **1** von **1**

SITZUNGSPROTOKOLL

Vom 06.03.2014

1 Beteiligte

Stefan Keller, Reto Schelbert, Carmelo Schumacher

2 Traktanden

2.1.1 Was wurde erreicht?

- Feingliederung Projekt
- Proof-Of-Concept DDD Symfony
- Einarbeitung Technologien, Evaluation Frontend-Framework

2.1.2 Was gibt es für Hindernisse?

- Limitierung Symfony als Rich-Web-Application-Provider -> Auftrennung in Frontend/Backend mit eigenem JS-Client-Framework für Frontend
- GoogleChallenge-Abklährung iTIXI (sensitive Daten/Routen public im Web? BusinessAccount nötig oder Alternativen)

2.1.3 Was ist geplant?

- Absegnung Fein-Projektplanung durch Stakeholders und interne Zielvorgaben
- Domain Model bis Iteration3 umgesetzt (Backend)
- CRUD-Architektur Client bis Iteration 3 umgesetzt
- Clientseitige Kernmodule Anzeige CRUD (Datagrid/Forms etc)

3 Diskussion / Beschlüsse

Risikoplanung	„Code bis Iteration 3 nicht so wie erwartet“
Frontend	Vorschlag für clientseitige Umsetzung mit AngularJS um eine saubere Trennung Frontend/Backend zu erhalten und mit dem Framework Unterstützung für die Darstellung in der Disposition zu bekommen.
Changes	Bis nächste Woche bestehendes Template (ohne Client Auslagerung) zur Architekturänderung übernehmen und präsentieren. Bei nächster Besprechung mit allen Stakeholdern clientseitige Änderungen berücksichtigen. Google Challenge mit Martin und Josi abklären.

4 Aufgaben

- Architekturumsetzung einer Entität in Symfony für nächstes Meeting

SITZUNGSPROTOKOLL

Vom 13.03.2014

1 Beteiligte

Stefan Keller, Reto Schelbert, Carmelo Schumacher, Martin Jonasse, Josi Conrad

2 Traktanden

2.1.1 Was wurde erreicht?

- Prototype Architektur DDD in Symfony

2.1.2 Was ist geplant?

- Umsetzung DDD, Stand Iteration 3 wiederherstellen
- Dispositionsprozess definieren

2.1.3 Zu klärende Punkte mit TIXI

- Einführung DDD
- Öffnung Fahrer geplant, Zugriff? -> Fahrersuche/Monatsplanung
- Google ToS, Business Licence?
- Daten für Zug bis 30.03 möglich?
- Termin für GUI-Testing
- DB/Domain Model (z.B. PIN Felder alter Datenbank?)

3 Diskussion / Beschlüsse

ORM	Einführung über Entscheid zu DDD und ORM, vorgängige Besprechung von Stefan und Martin. Heutige Lehre vertreten, ORM wird immer in irgendeiner Form verwendet. Weiter: MySQL wird durch Mehrbenutzerbetrieb sicher eine Transaktionsebene benötigen.
DDD	DDD Erklärung, Persistence und Business Logic Layer. Terminologie „CoreDomain" entspricht dem Business Layer, muss aber auf das angewendete Framework eingebunden werden. Normalisierungsformen in der rel. DB so nicht mehr angewendet, Domain Model erstellt, wird aber vom ORM übernommen welches die meiste Erfahrung mit sich bringt zu der Persistierung. Domain Driven der Data Driven, wobei Data Driven schnell an Grenzen kommt, wenn man viele Interkationen zwischen den Entitäten hat und Domain Driven dadurch mehr Vorteile bietet.
Termine	Codeabgabe von Iteration 3 am 23.März und Besprechung am 27.März mit TIXI.
Allgemeines Vorgehen	Gearbeitet wird auf Github in eigenem Account und Major Commits kommen auf Sourceforge.
Frontend	Client Framework wird momentan nicht verwendet. Tagesplanung für Disposition wird auch versucht per Native JS mit TWIG zu lösen. Sollte es sich aber erweise, dass wir ohne Client Framework nicht zu Recht kommen werden, müsste man es in Betracht ziehen und als agiles Vorgehen ansehen mit einer Verbesserung. Symfony als Backend auch mit Client Framework wie AngularJS, da Routing, Security etc. dennoch benötigt wird.Konservativer Entscheid ist gefällt, da neue Technologie AngularJS neu und Zukunft ebenfalls unsicher (wie langlebig dies durch Google ist)

Telefondienst	Benötigt eine REST-Schnittstelle über PW/Token etc. um einen PUSH an alle Clients zu senden über ein eintreffendes Ereignis.
Google Challenge	Es wird eine Map mit den Tagesrouten zur Verfügung gestellt im About o.ä. welche öffentlich zugänglich ist. Josi und Martin versichern, es handle sich soweit nicht um sensitive Daten, die Auflösung könne begrenzt und auch statisch dargestellt werden in der Map, somit dürfen Daten persistiert werden.
Limitations	Es wird aber durch die Begrenzung von 100 elements per 10s eine Business Licence angestrebt für 1000 requests per 10s.
Daten	Daten von TIXI Zug, sprich Adressen und Daueraufträge bis 30.3 wird versucht zu liefern von Martin und Josi.

4 Aufgaben

- Client Framework Entscheid
- Iteration3 implementieren
- OAuth Entscheid auch für spätere Verwendung (Telefon REST API)
- Google API Account lösen für TIXI

SITZUNGSPROTOKOLL

Vom 20.03.2014

1 Beteiligte

Stefan Keller, Reto Schelbert, Carmelo Schumacher

2 Traktanden

2.1.1 Was wurde erreicht?

- Besprechung mit TIXI letzte Woche, erfolgreiche Abklärungen
- Architektur und Implementation von DDD, Frontend und Security in Symfony

2.1.2 Was gibt es für Hindernisse?

- Addressen eindeutig persistieren (Frage an DB-Experte)
  - Volltextsuche über Addressstrings für Googles API

2.1.3 Was ist geplant?

- Stand von Iteration3 bis Sonntag wiederherstellen

3 Diskussion / Beschlüsse

Kurzmeeting	
PHPDoc	Wird für Code-Dokumentation verwendet
RUP	Nach RUP implementieren, Projektplanungskonzepte erstellen
SQL	Volltextsuche definieren über mehrere Felder möglich – MySQL Fulltext Search anschauen.
Build	Es wurden Skripts angefertig für einfacheren Build auf den Entwicklungsumgebungen + Build Server

SITZUNGSPROTOKOLL

Vom 27.03.2014

1 Beteiligte

Stefan Keller, Reto Schelbert, Carmelo Schumacher, Josi Conrad, Martin Jonasse

2 Traktanden

2.1.1 Was wurde erreicht?

- GUI und Styling für CRUD Operationen
- Erkenntnisse Performance, vorwiegend Gebrauch von TWIG/Symfony für Darstellung (kein FOSRestBundle)
- Generischer Entity Filter für verschachtelte Abfragen zum DataGrid
- Persistierung Adressen und Volltextsuche MySQL
- Deployment und Mindestversionen

2.1.2 Was gibt es für Hindernisse?

- Konflikt zwischen sauberem Design und pragmatischem Vorgehen / Zeitdruck
  - Detaillösungen vor Gesamtsicht
  - Aufwand und Nutzung des Frameworks unterschätzt
  - Dafür weitere Erkenntnisse gewonnen

2.1.3 Was ist geplant?

- GUI-Mokups, Client-Workflow, Detailspezifikation
  Google API Implementieren für Adressen-Abruf

3 Diskussion / Beschlüsse

Mandantenfähigkeit	Wird erreicht über verschiedene Applikationsinstanzen und Datenbanken
Anpassungen	- Kalender nicht ganze Monate durchklicken - Eigenschaft Übergewicht… alle Datenfelder wurde mit GL bestimmt, diese bei Implementation so berücksichtigen - Links für Hilfe/Einstellungen schlau platzieren
Integration Tests	werden durch Martin gemacht bei definitiver Code Abgabe, fehlende Datenwerte werden per Feedback gemeldet.
Client Tests	Für Clientseitige Tests müssen alle Elemente eine ID haben (dokumentiert – so versuchen zu übernehmen)
Build	Es wurden Skripts angefertig für einfacheren Build auf den Entwicklungsumgebungen + Build Server
Stateless	Client State auf allen Seiten beibehalten – ganze Sessions immer auf Server beibehalten? Mit Tabs + Browserfenstern nicht nötig, man hat sich auf diese Arbeitsweise momentan geeinigt. Kommen plötzliche Ereignisse herein während Disponentin z.B. am Kunden erfassen ist, dann öffnet Sie einen neuen Tab. Es muss nicht sein, dass alle Formulardaten beibehalten werden, auf eine andere Seite gewechselt wird, und beim wiederaufrufen von „Kunde" z.B. alle Felddaten noch vorhanden sind.
Google Challenge	Google API Business Kosten decken sich momentan nicht mit Nutzen. OpenStreetMap wird evaluiert für TIXI Problem.

SITZUNGSPROTOKOLL

Vom 03.04.2014

1 Beteiligte

Stefan Keller, Reto Schelbert, Carmelo Schumacher

2 Traktanden

2.1.1 Was wurde erreicht?

- GUI Abstraction mit Tiles, modularer Aufbau
- Fulltext search via MySQL
- GUI Elemente weiter spezifiziert
- Elemente der Disposition spezifiziert, Domain Model

2.1.2 Was gibt es für Hindernisse?

- Routing Service Nutzung, Google oder weitere Anbieter
  - Berechnung der Datenmengen
  - MapQuest, YOURS, ORS
- Schnittstelle FIBU

2.1.3 Was ist geplant?

- Implementation Stammdatenverwaltung
- Geocoding und Routing Service Abfragen spezifizieren

3 Diskussion / Beschlüsse

Tiles	Tiles – der verwendete Begriff (wie z.B. Android) ist Fragements
OSRM API	Wird definitiv genutzt, lokal oder online. Online – Service -> z.B. mal bei Sarah Hoffmann anfragen um den Service zu nutzen

Projekt – Sitzungsprotokoll

SITZUNGSPROTOKOLL

Vom 10.04.2014

1 Beteiligte

Stefan Keller, Reto Schelbert, Carmelo Schumacher

2 Traktanden

2.1.1 Was wurde erreicht?

- Implementation einzelner CRUD Views für Iteration 3
  - Validation HTML5
  - Translation durch Symfony

2.1.2 Was gibt es für Hindernisse?

- Routingservice

2.1.3 Was ist geplant?

- Wünsche und Anregungen übernehmen von heutiger Besprechung
- Refactoring und Bugfixing
- Detailspezifikationen der Disposition

3 Diskussion / Beschlüsse

DARP	PostgreSQL eine Frage wert
Optimierungsalgorithmus	Benutzer soll Hoheit behalten Änderungen in der Tagesplan manuell zu behandeln – Vorschlag mit: Optimierung laufen lassen und dann den State freezen, änderungen können manuell gemacht werden aber nicht mehr optimiert, ansonsten wieder entfreezen und ändern.

4 Aufgaben

- Dokumentationen nachführen
- Disposition konkreter spezifizieren

12. Juni 2014 Seite **1** von **1**

SITZUNGSPROTOKOLL

Vom 17.04.2014

1 Beteiligte

Stefan Keller, Reto Schelbert, Carmelo Schumacher, Martin Jonasse

2 Traktanden

2.1.1 Was wurde erreicht?

- Weitere CRUD Views für Iteration 3

2.1.2 Was ist geplant?

- Iteration 3 State abschliessen
- Disposition fertig spezifizieren
- Algorithmus Komponenten festlegen

3 Diskussion / Beschlüsse

Data-Mapping	Data-Mapping per Informationssystem durch Pentaho oder Google Refine als Möglichkeit, anstatt alte ID's von Objekten in die neue DB zu nehmen anstatt zusätzlicher Column PIN.
GAP Analyse	GAP Analyse wird durchgeführt und anschliessend werden Tickets erstellt von fehlenden Werten oder Änderungen. Tickets oder Bugs in Redmine/Github unter den Entwicklern.
OSRM	OSRM kann grundsätzlich eingesetzt werden. Dies würde eine lokale Installation bedeuten und uns einen Vorteil verschaffen bei der Routing-Abfrage dank sehr schneller Performance. Ein „Fallback" Szenario wäre ebenfalls enthalten, da man auf keinen weiteren Routing Online Service angewiesen ist.

4 Aufgaben

- Bei Formularen ein Aufruf falls man auf andere Seite klickt „Wollen Sie das Formular wirklich verlassen?" -> JS im Form
- Reto: Technologie-Stack auf Linux zusammenstellen

SITZUNGSPROTOKOLL

Vom 08.05.2014

1 Beteiligte

Stefan Keller, Reto Schelbert, Carmelo Schumacher, Martin Jonasse, Josi Conrad

2 Traktanden

2.1.1 Was wurde erreicht?

- Kontinuerliches Bugfixing und Korrespondenz seit Übergabe 2.8
- Algorithmus zur Lösung des Routing-Problems und dafür notwendige Datenstruktur konzeptionell erarbeitet
- OSRM Client API
- Basis für Address-Lookahead inklusive Google Geocode API
- Produktionsplan View
- Abruf der Address-Koordinaten per Google Geocode API

2.1.2 Was gibt es für Hindernisse?

- Erhöhte Komplexität durch Sammelaufträge

2.1.3 Was ist geplant?

- Implementation des Routing-Algorithmus
- Views für Monatsplanung, Fahrtauftragserfassung, Tagesplanung
  Intensive Implementationsphase

3 Diskussion / Beschlüsse

Zeitproblem	Was kann reduziert werden an den Anforderungen? • Grundsätzlich solides System zum arbeiten. Sammelaufträge z.B. nicht vollständig in Algorithmus implementieren, manuelle Bearbeitung durch Benutzer muss aber möglich sein. Disponentin wird somit als menschlicher Faktor immer noch sehr gut „gebraucht“
Abholzeiten	Wann wird dem Fahrgast die Abholzeit mitgeteilt? Relevant für Sammelaufträge, Vorgehensweise • Abholzeit schlussendlich wird beim Disponieren durchgegeben • Abholzeit relativ fix bei Daueraufträgen • Sammelauftrag manuell machen, die können die Disponenten gut. Zusammengehörigkeit der Fahrgäste ist wichtig, muss nicht unbedingt automatisiert sein. Dabei wird die Abholzeit beim Disponieren durchgegeben.

GAP	• Addresse: Haus Name? Gehört nicht in Strasse?
	  ○ Feld „Name“ gehört in Adresse, auch für POI. Namen besser für die Disponenten zum Unterscheiden, Bezeichnung wie „Altersheim da“, nicht relevant fürs Routing
	• Fahrgast: bevorzugte Fahrzeugkategorie?
	  ○ Nach Ausschlussverfahren welche Fahrzeugkategorie er auf keinen

SITZUNGSPROTOKOLL

Vom 23.05.2014

1 Beteiligte

Stefan Keller, Reto Schelbert, Carmelo Schumacher, Martin Jonasse

2 Traktanden

2.1.1 Was wurde erreicht?

- Übergabe 3.0
  - Beheben der offenen Gaps durch Übergabe 2.8
- Addresslookahead und Darstellung mit Möglichkeit zur manuellen Korrektur per Google Maps
- Basis für Fahrtoptimierungsalgorithmus mit erzeugten Testdaten
  - einfache Überprüfung der Durchführbarkeit einer Fahrt über Time-Windows
- Paralleler Abruf mehrerer tausend OSRM Routeninformationen über curl_multi für Berechnung der Leerfahrten
- View für Fahrauftragserfassung erstellt
- Konzeptionelle Views für Monats- und Tagesplanung erarbeitet

2.1.2 Was gibt es für Hindernisse?

- Viele Änderungen und Einflüsse durch Benutzer möglich, Change Management nötig
- Umsetzung Optimierungsalgorithmus in PHP

2.1.3 Was ist geplant?

- Bereitstellung aller für die Disposition notwendigen Funktionen im Backend
- Views für Monatsplanung und Tagesplanung
- Fahrtoptimierungsalgorithmus fertigstellen
- Offene Punkte und Issues bearbeiten

3 Diskussion / Beschlüsse

Daten Export	Martin: Die bestehenden Daten werden möglichst bald aus der jetzigen Access-Lösung herausexportiert. Einige unsaubere Daten liegen nun einmal vor und werden noch bereinigt bzw. richtig gestellt.
Sammelauftrag	Sammelauftrag möchte man noch genauer spezifizieren. Eine Disponentin fügt Sammelauftrag manuell hinzu, dies wird nicht autonom vom System übernommen da eine gewisse Freiheit für den Benutzer bleibt. Ein Sammelauftrag aus einem Dauerauftrag zu erstellen ist hinsichtlich aber schwierig und würde momentan den ganzen Ablauf bei der Erstellung Produktionsplan->Monatsplan beeinflussen. „Dauerhafter" Sammelaufträge aus Daueraufträgen (wiederkehrende Fahraufträge) wird nicht mehr in dieser Version implementiert. Die Lösung soll keinem Drive-By Betrieb dienen und der Sammelauftrag soll so einfach wie möglich implementiert sein.
Addresslookahead	wird vorgeführt und soweit für gut befunden. Benutzer müssen soweit instruiert werden, über Welche Stufen z.B. gesucht wird, wie man OVI Einträge findet etc.

19.2 System-Testdokumente

iTIXI

Testdokumentation

Baseline 2.8

Carmelo Schumacher, Reto Schelbert
24. April 2014

1 Inhalt

2 Systemtests 2
2.1 Bemerkungen 2
2.2 Voraussetzungen 2
2.3 Vorbereitungen 2
2.4 UC05: Fahrzeug verwalten 2
2.4.1 Testprotokoll 2
2.5 UC09: Fahrzeugservice verwalten 3
2.5.1 Testprotokoll 3
2.6 UC08: OVI verwalten 3
2.6.1 Testprotokoll 3
2.7 UC13: Fahrgast verwalten 3
2.7.1 Testprotokoll 4
2.8 UC06: Sensitive Fahrgastdaten verwalten 4
2.8.1 Testprotokoll 4
2.9 UC14: Fahrer verwalten 4
2.9.1 Testprotokoll 4
2.10 UC07: Sensitive Fahrerdaten verwalten 5
2.10.1 Testprotokoll 5
2.11 UC19: Absenzen verwalten 5
2.11.1 Testprotokoll 5
2.12 UC22: Managementdaten verwalten 6
2.12.1 Testprotokoll 6
3 Unittests 7
3.1 Fahrer 7
3.2 Fahrgast 7
3.3 OVI 7
3.4 Fahrzeug 7
3.5 Management 8
4 Performance Tests 9
5 Usability Tests 10

2 Systemtests

2.1 Bemerkungen

Die bisher erreichten Funktionen der Stammdatenverwaltung werden getestet. Eine GAP-Analyse durch den Projektpartner wird durchgeführt um fehlende Anforderungen zu überprüfen. Baseline 2.8 wird gegen die iTIXI SRS Liste getestet für die Funktionen F-001, F-010, F-020, F-030, F-040, F-050, F-060, F-140 und F-160. Diese Liste wurde ergänzt im Dokument: **iTIXI_SRS_BA_v2.8.pdf**

Systemtests werden zu den entsprechenden Use-Cases getestet.

2.2 Voraussetzungen

Es werden aktuelle Versionen von PHP 5.5 und MySQL 5.6 benötigt.

2.3 Vorbereitungen

Ein Build steht auf dem Buildserver bereit. Eine Person meldet sich über den Webbrowser auf dem Server an, die nachfolgenden Funktionen werden getestet.

2.4 UC05: Fahrzeug verwalten

#	Beschreibung	Erwartetes Ergebnis
1	Der Benutzer kann ein Fahrzeug erfassen.	Die Daten werden korrekt abgelegt.
2	Der Benutzer kann Fahrzeug bearbeiten.	Die Daten werden korrekt abgelegt.
3	Der Benutzer kann ein Fahrzeug löschen	Die Daten werden korrekt entfernt (logisch) und nicht mehr in der Liste angezeigt.
4	Der Benutzer kann sich die Liste aller Fahrzeuge über den Browser anzeigen lassen.	Die Daten werden korrekt angezeigt.
5	Kategorie, Betreuer und Depot können zugeordnet werden	Die Daten werden korrekt abgelegt und zu den Entitäten assoziiert

2.4.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	Fahrzeug wurde mit korrekten Daten erstellt	Ok
2	Daten wurden im Fahrzeug korrekt angepasst	Ok
3	Fahrzeug wurde erfolgreich gelöscht (logisch)	Ok
4	Liste aller Fahrzeuge wurde erfolgreich angezeigt	Ok

2.5 UC09: Fahrzeugservice verwalten

#	Beschreibung	Erwartetes Ergebnis
1	Der Benutzer kann Serviceplan erfassen.	Die Daten werden korrekt abgelegt.
2	Der Benutzer kann Serviceplan bearbeiten.	Die Daten werden korrekt abgelegt.
3	Der Benutzer kann einen Serviceplan löschen	Die Daten werden korrekt entfernt (logisch) und nicht mehr in der Liste angezeigt.
4	Der Benutzer kann sich die Liste aller aktuellen Servicepläne zu einem Fahrzeug anzeigen lassen.	Die Daten werden korrekt angezeigt, vergangene Pläne werden nicht angezeigt.
5	Der Benutzer kann kein Datum falsch angegeben	Speichern ist nicht möglich

2.5.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	Serviceplan wurde mit korrekten Daten erstellt	Ok
2	Daten wurden im Fahrzeug korrekt angepasst	Ok
3	Fahrzeugservice wurde erfolgreich gelöscht (logisch)	Ok
4	Fahrzeugservice-Liste wird zum korrespondierenden Fahrzeug angezeigt	Ok
5	End-Datum wird vor Start-Datum gesetzt, speichern nicht möglich -> Validierung	OK

2.6 UC08: OVI verwalten

#	Beschreibung	Erwartetes Ergebnis
1	Der Benutzer kann OVI erfassen.	Die Daten werden korrekt abgelegt.
2	Der Benutzer kann OVI bearbeiten.	Die Daten werden korrekt abgelegt.
3	Der Benutzer kann ein OVI löschen	Die Daten werden korrekt entfernt (logisch) und nicht mehr in der Liste angezeigt.
4	Der Benutzer kann sich die Liste aller OVI anzeigen lassen	Die Daten werden korrekt angezeigt.
5	Der Benutzerkann beliebig mehrere Keywords einem OVI zuordnen	Die Daten werden korrekt abgelegt und zu den Entitäten assoziiert

2.6.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	OVI wurde mit korrekten Daten erstellt	Ok
2	Daten wurden im OVI korrekt angepasst	Ok
3	OVI wurde erfolgreich gelöscht (logisch)	Ok
4	Liste aller OVI wird angezeigt	Ok
5	Keywords werden dem OVI zugewiesen	OK

2.7 UC13: Fahrgast verwalten

#	Beschreibung	Erwartetes Ergebnis
1	Der Benutzer kann Fahrgast erfassen.	Die Daten werden korrekt abgelegt.
2	Der Benutzer kann Fahrgast bearbeiten.	Die Daten werden korrekt abgelegt.
3	Der Benutzer kann ein Fahrgast löschen	Die Daten werden korrekt entfernt (logisch) und nicht mehr in der Liste angezeigt.
4	Der Benutzer kann sich die Liste aller Fahrgäste	Die Daten werden korrekt angezeigt.

	anzeigen lassen	
5	Der Benutzerkann Kategorien dem Fahrgast zuordnen	Die Daten werden korrekt abgelegt und zu den Entitäten assoziiert
6	Der Benutzer gibt fehlerhafte Daten ein: Telefon, EMail, Adresse	Die Daten werden korrekt Validiert und ein Speichern ist nicht möglich

2.7.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	Fahrgast wurde mit korrekten Daten erstellt	Ok
2	Daten wurden im Fahrgast korrekt angepasst	Ok
3	Fahrgast wurde erfolgreich gelöscht (logisch)	Ok
4	Liste aller Fahrgäste wird angezeigt	Ok
5	Kategorien werden dem Fahrgast zugewiesen	Ok
6	Fehlerhafte Felder werden validiert	Ok

2.8 UC06: Sensitive Fahrgastdaten verwalten

#	Beschreibung	Erwartetes Ergebnis
1	Manager meldet sich an und kann Fahrgast Details (vertraulich) bearbeiten.	Die Daten werden korrekt abgelegt, nur Benutzer mit Manager-Rolle sieht Feld.

2.8.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	Fahrgast Details (vertraulich) wurden korrekt geändert	Ok
2	Fahrgast Details (vertraulich) nur über Manager einsehbar	Ok

2.9 UC14: Fahrer verwalten

#	Beschreibung	Erwartetes Ergebnis
1	Der Benutzer kann Fahrer erfassen.	Die Daten werden korrekt abgelegt.
2	Der Benutzer kann Fahrer bearbeiten.	Die Daten werden korrekt abgelegt.
3	Der Benutzer kann ein Fahrer löschen	Die Daten werden korrekt entfernt (logisch) und nicht mehr in der Liste angezeigt.
4	Der Benutzer kann sich die Liste aller Fahrer anzeigen lassen	Die Daten werden korrekt angezeigt.
5	Der Benutzerkann Kategorien dem Fahrer zuordnen	Die Daten werden korrekt abgelegt und zu den Entitäten assoziiert
6	Der Benutzer gibt fehlerhafte Daten ein: Telefon, EMail, Adresse	Die Daten werden korrekt Validiert und ein Speichern ist nicht möglich

2.9.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	Fahrer wurde mit korrekten Daten erstellt	Ok
2	Daten wurden im Fahrer korrekt angepasst	Ok
3	Fahrer wurde erfolgreich gelöscht (logisch)	Ok
4	Liste aller Fahrer wird angezeigt	Ok

5	Kategorien werden dem Fahrer zugewiesen	Ok
6	Fehlerhafte Felder werden validiert	Ok

2.10 UC07: Sensitive Fahrerdaten verwalten

#	Beschreibung	Erwartetes Ergebnis
1	Manager meldet sich an und kann Fahrer Details (vertraulich) bearbeiten.	Die Daten werden korrekt abgelegt, nur Benutzer mit Manager-Rolle sieht Feld.

2.10.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	Fahrer Details (vertraulich) wurden korrekt geändert	Ok
2	Fahrer Details (vertraulich) nur über Manager einsehbar	Ok

2.11 UC19: Absenzen verwalten

#	Beschreibung	Erwartetes Ergebnis
1	Der Benutzer kann Absenz erfassen.	Die Daten werden korrekt abgelegt.
2	Der Benutzer kann Absenz bearbeiten.	Die Daten werden korrekt abgelegt.
3	Der Benutzer kann eine Absenz löschen.	Die Daten werden korrekt entfernt (logisch) und nicht mehr in der Liste angezeigt.
4	Der Benutzer kann sich die Liste aller aktuellen Absenzen zu einem Fahrer oder Fahrgast anzeigen lassen.	Die Daten werden korrekt angezeigt, vergangene Absenzen werden nicht angezeigt.
5	Der Benutzer kann kein Datum falsch angegeben	Speichern ist nicht möglich

2.11.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	Absenz wurde mit korrekten Daten erstellt	Ok
2	Daten wurden im Absenz korrekt angepasst	Ok
3	Absenz wurde erfolgreich gelöscht (logisch)	Ok
4	Absenz -Liste wird zum korrespondierenden Fahrer oder Fahrgast angezeigt	Ok
5	End-Datum wird vor Start-Datum gesetzt, speichern nicht möglich -> Validierung	OK

2.12 UC22: Managementdaten verwalten

Dies sind vereinzelt kleine CRUD Funktionalitäten die nicht ganz im Detail erwähnt werden.

#	Beschreibung	Erwartetes Ergebnis
1	Der Admin kann Benutzer erfassen und bearbeiten.	Die Daten werden korrekt abgelegt.
2	Der Manager kann Fahrzeugkategorien erfassen und bearbeiten	Die Daten werden korrekt abgelegt.
3	Der Manager kann Fahrzeugdepots erfassen und bearbeiten	Die Daten werden korrekt abgelegt.
4	Der Manager kann OVI-Keywords erfassen und bearbeiten	Die Daten werden korrekt abgelegt.
5	Der Manager kann Personenkategorien erfassen und bearbeiten	Die Daten werden korrekt abgelegt.
6	Der Manager kann Feiertage erfassen und bearbeiten	Die Daten werden korrekt abgelegt.
7	Der Manager kann Schichten erfassen und bearbeiten	Die Daten werden korrekt abgelegt.

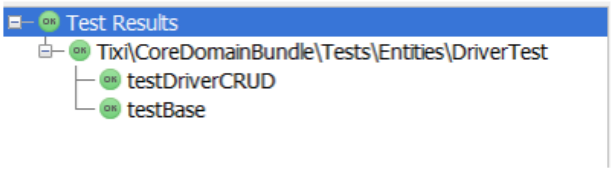
2.12.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	Benutzer wurden erstellt/angepasst	Ok
2	Fahrzeugkategorien wurden erstellt/angepasst	Ok
3	Fahrzeugdepots wurden erstellt/angepasst	Ok
4	OVI-Keywords wurden erstellt/angepasst	Ok
5	Personenkategorien wurden erstellt/angepasst	Ok
6	Feiertage wurden erstellt/angepasst	Ok
7	Schichten wurden erstellt/angepasst	Teilweise

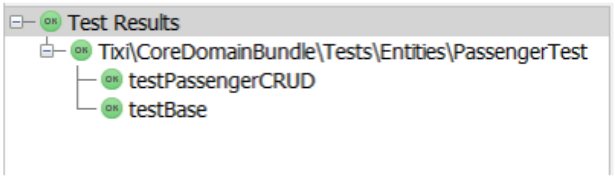
7 – Validierung der Schicht Zeiten muss verbessert werden

3 Unittests

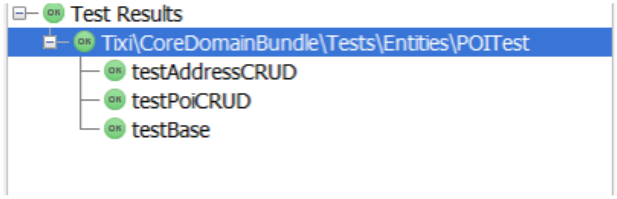
3.1 Fahrer



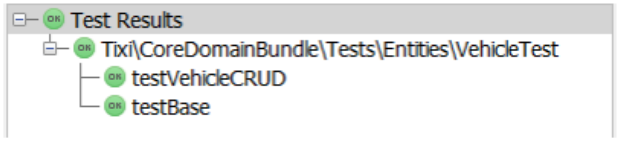
3.2 Fahrgast



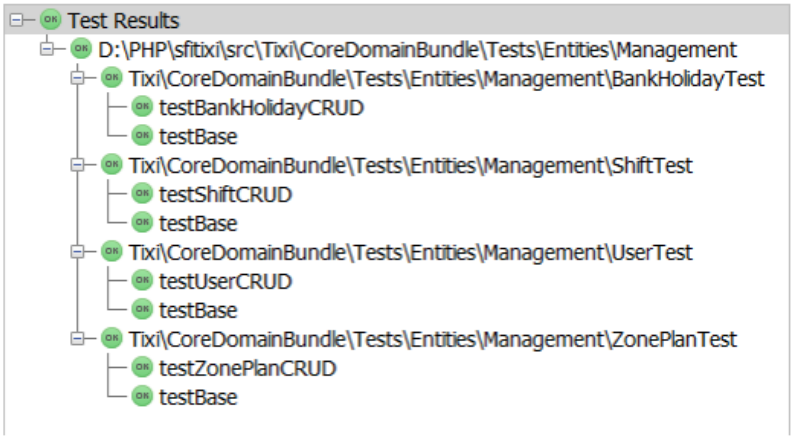
3.3 OVI



3.4 Fahrzeug



3.5 Management

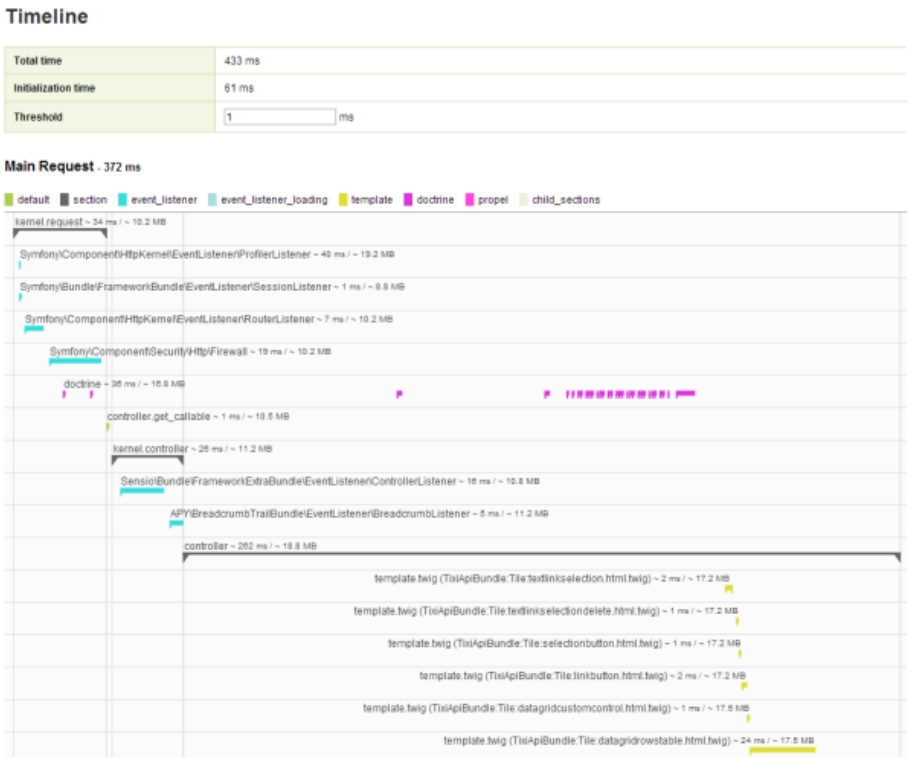



```
D:\PHP\server\binaries\php\php_runningversion\php.exe C:\Temp\ide-phpunit.php --configuration
D:\PHP\sfitixi\app\phpunit.xml D:\PHP\sfitixi\src\Tixi\CoreDomainBundle\Tests\Entities\Management
Testing started at 15:56 ...
PHPUnit 3.7.37 by Sebastian Bergmann.

Configuration read from D:\PHP\sfitixi\app\phpunit.xml

Time: 2.74 seconds, Memory: 24.00Mb

OK (8 tests, 12 assertions)

Process finished with exit code 0
```

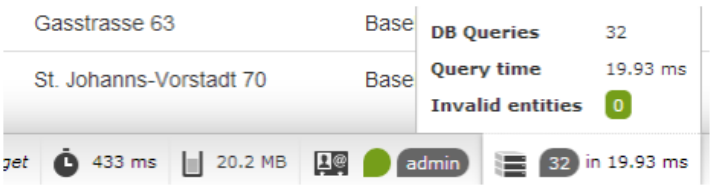
4 Performance Tests

Die Bedienung auf Client-Seite erweist sich als sehr performant. Es war zu keiner Zeit eine längere Verzögerung spürbar. Die Produktiv-Umgebung des Symfony-Frameworks rendert Seiten noch einmal um einiges schneller als die Development-Umgebung, da hier ein Cahing genutzt wird, aktuell PHP 5.5 OPcache. Grosse Data-Grids werden korrekt und sehr geringer Verzögerung durch Pagination und Lazy-Loading vom ORM angezeigt.

View Fahrgäste mit 554 Einträgen



Page-Aufbau und DataGrid Queries



5 Usability Tests

Über die Tests durch den Projektpartner sind wir soweit immer informiert gewesen, falls Probleme bei der Bedienung aufgetaucht sind. Einige Versionen wurden durch beide Partner und meist auch mit der Geschäftsleitung durchgegangen. Soweit wurden keine grossen Probleme gemeldet, Funktionen über das Menü konnte gefunden werden, diese sind auch sehr angelehnt an die anfängliche Version und entsprechen den Spezifikationen in den Anforderungen.

iTIXI

Testdokumentation

Baseline 3.2

Carmelo Schumacher, Reto Schelbert
12. Juni 2014

1 Inhalt

2 Systemtests 2

2.1 Bemerkungen 2

2.2 Voraussetzungen 2

2.3 Vorbereitungen 2

2.4 UC01: Produktionsplan verwalten 2

2.4.1 Testprotokoll 2

2.5 UC02: Monatsplan verwalten 3

2.5.1 Testprotokoll 3

2.6 UC03: Monatsplan versenden (proof of concept) 3

2.6.1 Testprotokoll 3

2.7 UC15: Einsatz verwalten 3

2.7.1 Testprotokoll 3

2.8 UC20: Dauereinsatz verwalten 3

2.8.1 Testprotokoll 4

2.9 UC16: Auftrag verwalten 4

2.9.1 Testprotokoll 4

2.10 UC17: Mehrfachauftrag verwalten 4

2.10.1 Testprotokoll 4

3 Unittests 5

3.1 Dispositionsmanagement 5

4 Performance Tests 7

2 Systemtests

2.1 Bemerkungen

Die bisher erreichten Funktionen der Disposition werden getestet. Eine GAP-Analyse durch den Projektpartner wird durchgeführt um fehlende Anforderungen zu überprüfen. Baseline 3.2 wird nicht mehr gegen eine detailliert SRS Liste zusammen mit dem Projektpartner getestet, da hier von beiden Seiten aus zu wenig Zeit vorhanden war. Im nachfolgenden werden unsere Tests aufgeführt entsprechend unseren Use Cases.

Die Funktionen/Use Cases zur Tagesplanung werden nicht mehr vollumfänglich getestet, da sie kein Bestandteil mehr sind dieser Arbeit.

2.2 Voraussetzungen

Es werden aktuelle Versionen von PHP 5.5 und MySQL 5.6 benötigt.

Ein OSRM Server muss erreichbar sein für das Abrufen von Routen.

Ein Google API Key für die Nutzung des Addresslookahead wird verlangt.

2.3 Vorbereitungen

Ein Build steht auf dem Buildserver bereit. Eine Person meldet sich über den Webbrowser auf dem Server an, die nachfolgenden Funktionen werden getestet.

2.4 UC01: Produktionsplan verwalten

#	Beschreibung	Erwartetes Ergebnis
1	Der Benutzer kann einen Produktionsplan erfassen.	Die Daten werden korrekt abgelegt.
2	Der Benutzer kann einen Produktionsplan bearbeiten.	Die Daten werden korrekt abgelegt und über alle assoziierten Objekte aktualisiert.
3	Der Benutzer kann sich die Liste aller Produktionspläne über den Browser anzeigen lassen.	Die Daten werden korrekt angezeigt.
4	Benutzer kann Produktionsplan nach Monatsplanung ändern	Die Daten des Produktionsplan werden korrekt abgelegt und Monatsplan wird anhand dessen aktualisiert

2.4.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	Produktionsplan wurde mit korrekten Daten erstellt	Ok
2	Daten wurden im Produktionsplan korrekt angepasst, entsprechende Objekte wurden aktualisiert	Ok
3	Liste aller Produktionspläne wird angezeigt	Ok
4	Produktionsplan wurde geändert und Monatsplan wurde entsprechende aktualisiert mit mehr Slots für Fahrer	Ok

2.5 UC02: Monatsplan verwalten

#	Beschreibung	Erwartetes Ergebnis
1	Der Benutzer kann Monatsplan erfassen.	Die Daten werden korrekt abgelegt.
2	Der Benutzer kann Monatsplan bearbeiten.	Die Daten werden korrekt abgelegt und über alle assoziierten Objekte aktualisiert.
3	Der Benutzer kann sich die Liste aller Monatspläne über den Browser anzeigen lassen.	Die Daten werden korrekt angezeigt.

2.5.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	Monatsplan wurde mit korrekten Daten erstellt	Ok
2	Daten wurden im Monatsplan korrekt angepasst, entsprechende Objekte wurden aktualisiert	Ok
3	Liste aller Monatspläne wird angezeigt	Ok

2.6 UC03: Monatsplan versenden (proof of concept)

#	Beschreibung	Erwartetes Ergebnis
1	System kann Monatsplan PDF per Mail an alle Fahrer senden.	Die Daten werden korrekt erzeugt und kommen per Mail an.

2.6.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	System erstellt Mail per Service und versendet Monatsplan als PDF	Teilweise Ok

1 System Services zum Mail und PDF Erzeugen sind vorhanden und funktionieren, die genauen Funktionen zur „Bereitstellung“ sind nicht vorhanden und auch nicht Teil dieser Arbeit.

2.7 UC15: Einsatz verwalten

Da Einsätze aus der vorhandenen Ressourcenplanung im Monatsplan erstellt werden, kommt der Benutzer nur noch indirekt mit dem Erstellen eines Einsatzes in Kontakt. Einätze von zugewiesenen Fahrern können gelöscht werden.

#	Beschreibung	Erwartetes Ergebnis
1	Der Benutzer kann Einsätze von Fahrern löschen	Die Daten werden korrekt entfernt und aus dem Monatsplan ausgetragen.
2	Der Benutzer kann Einsätze an Fahrern über Monatsplan zuteilen.	Die Daten werden korrekt abgelegt und für den Fahrer in der Monatsplanung aktualisiert.

2.7.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	Einsätze wurden von Fahrer gelöscht und kamen nicht mehr im Monatsplan vor	Ok
2	Slot im Monatsplan wurde mit Fahrer gefüllt, Daten wurden im Fahrer korrekt angepasst	Ok

2.8 UC20: Dauereinsatz verwalten

#	Beschreibung	Erwartetes Ergebnis
1	Der Benutzer kann Dauereinsatz für Fahrer erstellen und ändern.	Die Daten werden korrekt angelegt und löst Updates aus bei Monatsplanung.
2	Der Benutzer kann Dauereinsatz von Fahrern entfernen.	Die Daten werden korrekt entfernt und über alle Instanzen aktualisiert.

2.8.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	Dauereinsatz wurde mit korrekten Daten erstellt und Änderungen wurden bei Monatsplanung aktualisiert	Ok
2	Löschung eines Dauereinsatzes löst Aktualisierung im Monatsplan aus, Daten wurden korrekt angepasst	Ok

2.9 UC16: Auftrag verwalten

#	Beschreibung	Erwartetes Ergebnis
1	Benutzer erstellt oder ändert Fahrauftrag für einen Fahrgast	Daten werden korrekt abgelegt und mit Fahrgast assoziiert.
2	Benutzer löscht Fahrauftrag für Fahrgast	Daten werden korrekt entfernt, Tagesplanung und Zuordnung zu Einsätzen werden aktualisiert.

2.9.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	Neuer Fahrauftrag wird für Fahrgast erstellt, Änderungen von Fahrgast aus werden korrekt übernommen	Ok
2	Fahrauftrag wird gelöscht, zugeordnete Einsätze werden aktualisiert	Teilweise Ok

2 Das konkrete Change-Management für die Tagesplanung konnte nur teilweise getestet werden bis Projekt-Ende und die Tagesplanung (mit View) ist nicht mehr Teil dieser Arbeit.

2.10 UC17: Mehrfachauftrag verwalten

#	Beschreibung	Erwartetes Ergebnis
1	Benutzer erstellt oder ändert Mehrfachauftrag für einen Fahrgast	Daten werden korrekt abgelegt und mit Fahrgast assoziiert. Einzelaufträge werden aus dem Mehrfachauftrag erstellt pro Monat.
2	Benutzer löscht Mehrfachauftrag für Fahrgast	Daten werden korrekt entfernt, Tagesplanung und Zuordnung zu Einsätzen werden aktualisiert.

2.10.1 Testprotokoll

#	Erhaltenes Ergebnis	Resultat
1	Neuer Mehrfachauftrag wird für Fahrgast erstellt, Änderungen von Fahrgast aus werden korrekt übernommen	Ok
2	Mehrfachauftrag wird gelöscht, zugeordnete Einzeleinsätze werden aktualisiert	Teilweise Ok

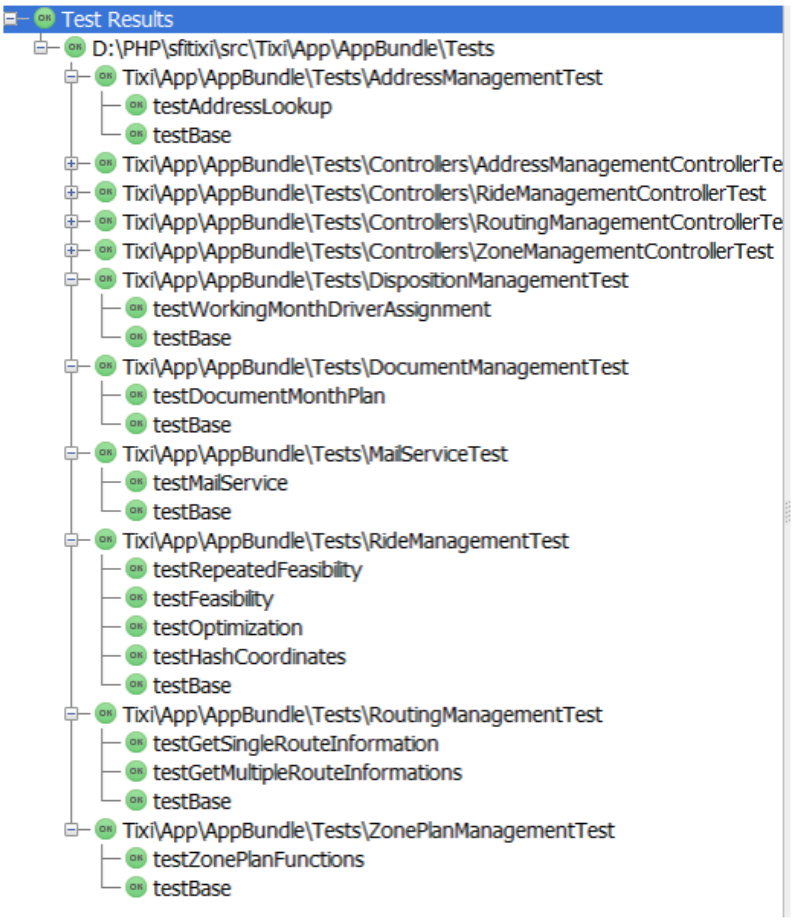
2 Das konkrete Change-Management für die Tagesplanung konnte nur teilweise getestet werden bis Projekt-Ende und die Tagesplanung (mit View) ist nicht mehr Teil dieser Arbeit.

3 Unittests

3.1 Dispositionsmanagement

Getestet wurde über die Services hinaus, welche u.A. die changeAmount Prozesse einleiten bei Änderungen von Produktionsplan, damit Monatsplan und alle korrespondierende Einträge aktualisiert werden.

Dazu gehören Controller für die API, welche beim Fahrauftrag erstellen per Ajax benötigt werden, mit AddressLookahead und Berechnung der Route, ZonenPlan abfragen für die Adressen, sowie Machbarkeitscheck eines Auftrags oder Mehrfachauftrags verglichen mit bestehenden Aufträgen oder Mehrfachaufträgen. PDF Dokument-Erstellung und Mailversand als *Proof of Concept*, sowie der Fahrtoptimierung, die später bei der Tagesplanung benötigt wird.



iTIXI - Testdokumentation

D:\PHP\server\binaries\php\php_runningversion\php.exe C:\Temp\ide-phpunit.php --configuration D:\PHP\sfitixi\app\phpunit.xml D:\PHP\sfitixi\src\Tixi\App\AppBundle\Tests

Testing started at 18:58 ...

PHPUnit 3.7.37 by Sebastian Bergmann.

Configuration read from D:\PHP\sfitixi\app\phpunit.xml

Filled 1439 rides from RoutingMachine in: 6.36455202102666s

Total RideNodes: 57

Feasible init config built in: 4.177551984787s

Annealing with Temperatur: 10000 and 18411 Iterations

Initial Distance: 724.556km - best Annealed Distance: 717.654km - saved 6.902km

Built rideConfiguration in: 9.0613539218903s

Configuration Vehicles: 17 - Distance: 717.654km - EmptyRideTime: 106min - EmptyRideDistance: 56.811km

Not Feasible Nodes: 0

0 W:1 P:1	VW Caddy 1	(684:Pfeff->Schö)	(676:Kaltb->Bläs)	(694:Clara->Engel)	(697:Solot->Hohe)	(667:Neuwe->Wiese)
1 W:1 P:2	VW Caddy 3	(650:Frobu->Mülh)	(644:Lothr->Allsc)	(669:Eiche->Flugh)	(653:Grien->Gotth)	
2 W:1 P:1	VW Caddy 4	(651:Prau ->St. A)	(692:In de->St. A)			
3 W:1 P:2	VW Caddy 2	(672:Hofst->Grell)	(682:Gelle->Lange)	(657:Lange->Elsä)	(690:Toten->Dorna)	
4 W:1 P:1	VW Maxi 1	(678:Schü->Kornh) (655:Markt->Prunt)	(700:Spale->Fasan)	(695:Rümm->Augus)	(685:Fisch->Laupe)	(670:St. G->Kembs)
5 W:1 P:1	VW Maxi 9	(686:Fisch->Rüti)	(654:Burgu->Istei)	(664:Riehe->Nadel)		
6 W:1 P:1	VW Maxi 10	(683:Rhein->Baumg)	(652:J. J.->Nadel)	(658:Nadel->Kienb)	(671:Berga->St. J)	
7 W:1 P:2	VW Maxi 7	(677:Gunde->Im La)	(646:Pilat->Thier)	(668:Reina->Kraft)	(698:Dorna->Hebel)	
8 W:1 P:1	VW Maxi 3	(691:Wilhe->Leuen)	(666:St. J->Rennw)	(645:Reina->Matte)		
9 W:1 P:1	VW Maxi 2	(673:Wande->Schau)	(648:St. A->Insel)	(674:Wilhe->Spale)	(662:Höhe->Eiche)	(675:Parad->Spita)
10 W:1 P:2	VW Maxi 4	(647:Insel->Grell)	(656:St. A->Schau)	(649:Peter->Peter)	(687:Hegen->Redin)	(688:Aeuss->Engel)
11 W:0 P:3	VW Maxi 5	(660:St. A->Im La)	(679:Morga->Hamme)	(665:Selti->St. A)		
12 W:1 P:3	VW Maxi 6	(693:Fisch->Insel)	(659:Schwa->Rixhe)	(663:Im He->Fisch)		
13 W:1 P:2	VW Maxi 8	(661:Gener->Realp)	(680:St. G->Unter)	(699:Weihe->Müll)		
14 W:0 P:1	VW Movano 2	(689:Marig->Schwe)				
15 W:0 P:2	VW Movano 1	(681:Marig->Unter)				
16 W:1 P:1	VW Movano 3	(696:Prau ->Kaltb)				

Time: 29.63 seconds, Memory: 74.25Mb

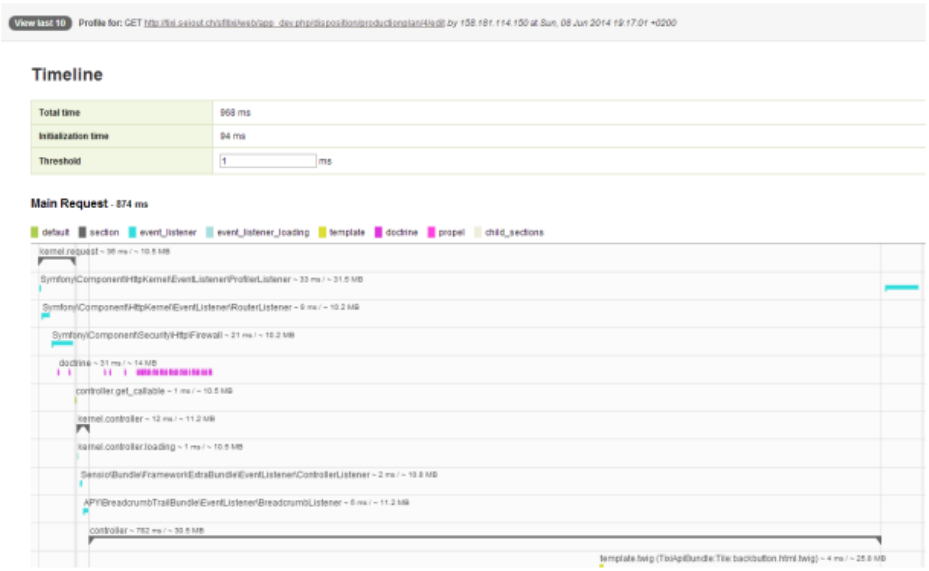
OK (29 tests, 34 assertions)

Process finished with exit code 0

iTIXI - Testdokumentation

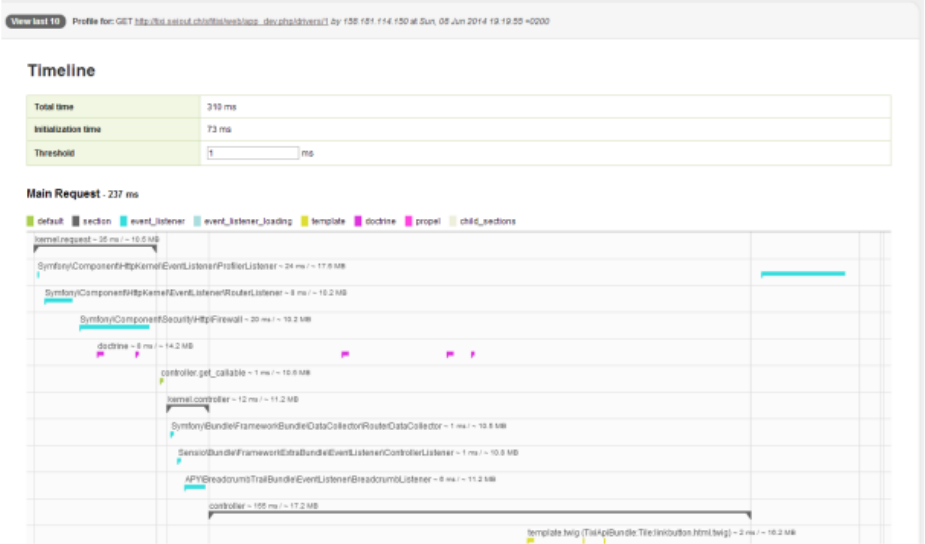
4 Performance Tests

Die Erstellung eines Produktionsplan benötigt etwas Zeit, da hier viele Objekte erzeugt und persistiert werden. Es wird aber dennoch alles in sehr kurzer Zeit erstellt und durchgeführt. So oder so, da dies aber nur einmal monatlich durchgeführt wird, ist es völlig verkraftbar. Etwa in gleicher Zeit werden Monatspläne eröffnet und Fahrer zugeteilt.

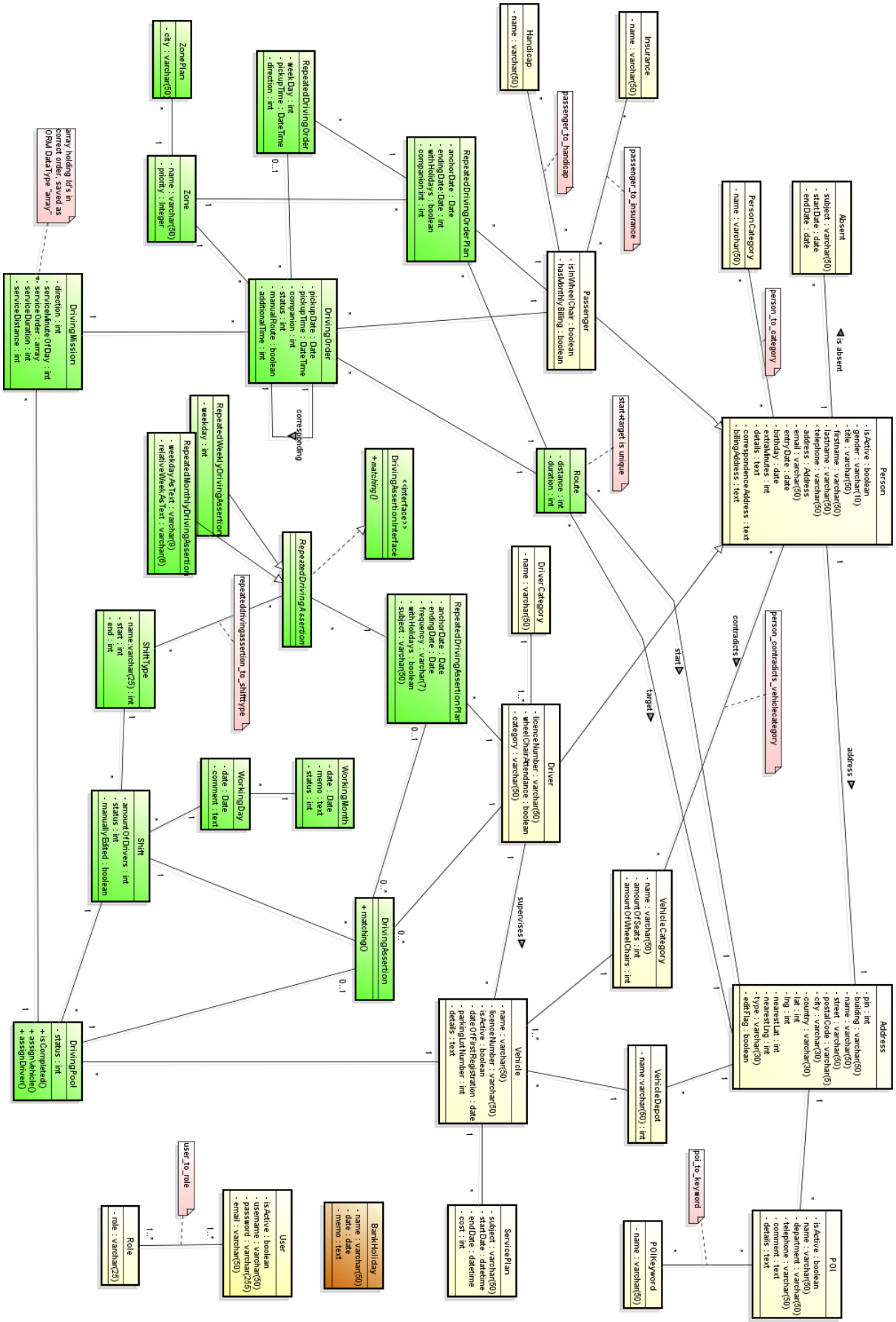


iTIXI - Testdokumentation

Die Zuteilung von Fahrern im Monatsplan und die Changes bei Einsatz und Dauereinsatzänderungen werden performant durchgeführt.



19.3 Domain Model (Grossformat)



19.4 Annealing Tests

Initial Konfiguration für Annealing über einfacher Zuweisung nach kürzester Zeit, einfacher Algorithmus von RideStrategyLeastDuration

```
Filled 1002 rides from RoutingMachine in: 4.5973479747772s
Total RideNodes: 47
Feasible init config built in: 0.047523021697998s
Annealing with Temperatur: 10000 and 18411 Iterations
Initial Distance: 714.377km - best Annealed Distance: 430.101km - saved 284.276km
Built rideConfiguration in: 3.8421280384064s
Configuration Vehicles: 13 - Distance: 430.101km - EmptyRideTime: 125min - EmptyRideDistance: 61.618km
Not Feasible Nodes: 0
0 W:1 P:3   VW Caddy 1  |(117)  (152)   (134)   (143)   (129)   (147)   (114)
1 W:1 P:2   VW Caddy 3  |(122)  (147)   (147)   (136)
2 W:0 P:2   VW Caddy 4  |(149)  (131)   (148)   (127)   (146)   (124)
3 W:1 P:2   VW Caddy 2  |(140)  (112)   (153)   (150)
4 W:1 P:3   VW Maxi 1   |(113)  (148)   (153)   (126)
5 W:1 P:2   VW Maxi 9   |(141)  (120)   (134)   (146)   (121)
6 W:1 P:3   VW Maxi 10  |(142)  (155)   (138)
7 W:1 P:1   VW Maxi 7   |(135)  (158)   (134)   (118)   (157)
8 W:1 P:3   VW Maxi 3   |(116)  (115)   (134)   (123)
9 W:1 P:2   VW Maxi 2   |(148)  (144)
10 W:0 P:2  VW Maxi 4   |(147)
11 W:0 P:2  VW Maxi 5   |(130)
12 W:1 P:0  VW Maxi 6   |(144)
```

```
Filled 1002 rides from RoutingMachine in: 4.4946568012238s
Total RideNodes: 47
Feasible init config built in: 0.043308019638062s
Annealing with Temperatur: 10000 and 18411 Iterations
Initial Distance: 714.377km - best Annealed Distance: 445.805km - saved 268.572km
Built rideConfiguration in: 3.7751269340515s
Configuration Vehicles: 13 - Distance: 445.805km - EmptyRideTime: 122min - EmptyRideDistance: 65.828km
Not Feasible Nodes: 0
0 W:1 P:3   VW Caddy 1  |(142)  (144)   (132)   (118)   (129)   (133)   (124)
1 W:0 P:3   VW Caddy 3  |(131)  (153)   (146)   (114)
2 W:1 P:3   VW Caddy 4  |(117)  (122)   (142)   (134)   (146)   (121)
3 W:1 P:1   VW Caddy 2  |(135)  (142)   (153)   (128)
4 W:1 P:2   VW Maxi 1   |(149)  (112)   (147)   (157)
5 W:1 P:3   VW Maxi 9   |(141)  (122)   (151)   (146)   (146)
6 W:1 P:1   VW Maxi 10  |(142)  (144)   (126)
7 W:1 P:2   VW Maxi 7   |(140)  (137)   (145)   (143)   (127)
8 W:1 P:3   VW Maxi 3   |(113)  (148)   (119)   (144)
9 W:1 P:1   VW Maxi 2   |(142)  (150)
10 W:0 P:2  VW Maxi 4   |(115)
11 W:0 P:2  VW Maxi 5   |(136)
12 W:1 P:0  VW Maxi 6   |(138)
```

```
Filled 1002 rides from RoutingMachine in: 4.5439460277557s
Total RideNodes: 47
Feasible init config built in: 0.042546033859253s
Annealing with Temperatur: 10000 and 18411 Iterations
Initial Distance: 714.377km - best Annealed Distance: 466.298km - saved 248.079km
Built rideConfiguration in: 3.5080490112305s
Configuration Vehicles: 13 - Distance: 466.298km - EmptyRideTime: 134min - EmptyRideDistance: 73.083km
Not Feasible Nodes: 0
0 W:1 P:3   VW Caddy 1  |(116)  (145)   (134)   (127)   (147)   (123)   (114)
1 W:1 P:3   VW Caddy 3  |(141)  (116)   (146)   (136)
2 W:1 P:3   VW Caddy 4  |(117)  (131)   (139)   (132)   (147)   (153)
3 W:1 P:2   VW Caddy 2  |(120)  (119)   (112)   (129)
4 W:0 P:3   VW Maxi 1   |(152)  (118)   (125)   (133)
5 W:1 P:3   VW Maxi 9   |(142)  (122)   (154)   (128)   (124)
6 W:0 P:3   VW Maxi 10  |(135)  (131)   (155)
7 W:1 P:2   VW Maxi 7   |(140)  (156)   (151)   (126)   (121)
8 W:0 P:2   VW Maxi 3   |(149)  (148)   (115)   (129)
9 W:1 P:2   VW Maxi 2   |(143)  (130)
10 W:1 P:0  VW Maxi 4   |(144)
11 W:1 P:0  VW Maxi 5   |(150)
12 W:1 P:0  VW Maxi 6   |(158)
```

```
Filled 1002 rides from RoutingMachine in: 4.4779698848724s
Total RideNodes: 47
Feasible init config built in: 0.042863845825195s
Annealing with Temperatur: 10000 and 18411 Iterations
Initial Distance: 714.377km - best Annealed Distance: 439.599km - saved 274.778km
Built rideConfiguration in: 3.7715458869934s
Configuration Vehicles: 13 - Distance: 439.599km - EmptyRideTime: 139min - EmptyRideDistance: 75.494km
Not Feasible Nodes: 0
0 W:1 P:3   VW Caddy 1  |(122)  (134)   (119)   (118)   (129)   (146)   (114)
1 W:0 P:2   VW Caddy 3  |(148)  (120)   (153)   (153)
2 W:1 P:3   VW Caddy 4  |(113)  (140)   (137)   (151)   (147)   (124)
3 W:1 P:3   VW Caddy 2  |(141)  (142)   (133)   (136)
4 W:1 P:1   VW Maxi 1   |(117)  (122)   (138)   (121)
5 W:1 P:1   VW Maxi 9   |(131)  (132)   (134)   (138)   (153)
6 W:1 P:1   VW Maxi 10  |(135)  (144)   (153)
7 W:1 P:2   VW Maxi 7   |(149)  (142)   (134)   (127)   (150)
8 W:1 P:2   VW Maxi 3   |(122)  (115)   (143)   (157)
9 W:1 P:1   VW Maxi 2   |(158)  (153)
10 W:0 P:1  VW Maxi 4   |(153)
11 W:0 P:1  VW Maxi 5   |(152)
12 W:1 P:0  VW Maxi 6   |(122)
```

```
Filled 1002 rides from RoutingMachine in: 4.4906718730927s
Total RideNodes: 47
Feasible init config built in: 0.042847871780396s
Annealing with Temperatur: 10000 and 18411 Iterations
Initial Distance: 714.377km - best Annealed Distance: 438.666km - saved 275.711km
Built rideConfiguration in: 3.7808458805084s
Configuration Vehicles: 13 - Distance: 438.666km - EmptyRideTime: 117min - EmptyRideDistance: 61.392km
Not Feasible Nodes: 0
0 W:1 P:3   VW Caddy 1  |(131)  (122)   (115)   (143)   (129)   (146)   (114)
1 W:0 P:2   VW Caddy 3  |(149)  (149)   (155)   (157)
2 W:1 P:1   VW Caddy 4  |(141)  (158)   (132)   (151)   (151)   (126)
3 W:1 P:1   VW Caddy 2  |(117)  (112)   (151)   (134)
4 W:1 P:3   VW Maxi 1   |(134)  (127)   (147)   (121)
5 W:1 P:3   VW Maxi 9   |(142)  (142)   (153)   (153)   (136)
6 W:1 P:1   VW Maxi 10  |(152)  (118)   (119)
7 W:1 P:2   VW Maxi 7   |(140)  (156)   (134)   (146)   (146)
8 W:1 P:2   VW Maxi 3   |(120)  (148)   (151)   (124)
9 W:1 P:1   VW Maxi 2   |(135)  (150)
10 W:0 P:1  VW Maxi 4   |(139)
11 W:0 P:2  VW Maxi 5   |(145)
12 W:1 P:1  VW Maxi 6   |(134)
```

```
Filled 1002 rides from RoutingMachine in: 4.5644958019257s
Total RideNodes: 47
Feasible init config built in: 0.043385028839111s
Annealing with Temperatur: 10000 and 18411 Iterations
Initial Distance: 714.377km - best Annealed Distance: 461.386km - saved 252.991km
Built rideConfiguration in: 3.5396409034729s
Configuration Vehicles: 13 - Distance: 461.386km - EmptyRideTime: 108min - EmptyRideDistance: 54.349km
Not Feasible Nodes: 0
0 W:1 P:3   VW Caddy 1  |(135)  (131)   (158)   (155)   (129)   (146)   (121)
1 W:1 P:1   VW Caddy 3  |(142)  (144)   (143)   (157)
2 W:1 P:2   VW Caddy 4  |(120)  (140)   (115)   (119)   (133)   (136)
3 W:1 P:1   VW Caddy 2  |(141)  (137)   (112)   (138)
4 W:1 P:2   VW Maxi 1   |(145)  (154)   (147)   (123)
5 W:1 P:1   VW Maxi 9   |(117)  (139)   (153)   (153)   (114)
6 W:1 P:1   VW Maxi 10  |(122)  (132)   (126)
7 W:1 P:2   VW Maxi 7   |(149)  (156)   (118)   (128)   (124)
8 W:0 P:3   VW Maxi 3   |(113)  (148)   (127)   (130)
9 W:1 P:1   VW Maxi 2   |(116)  (150)
10 W:0 P:1  VW Maxi 4   |(152)
11 W:0 P:1  VW Maxi 5   |(125)
12 W:1 P:1  VW Maxi 6   |(151)
```

19.5 Benutzeranleitungen

19.5.1 Build on Windows

iTixi

Build on Windows

Howto

Carmelo Schumacher, Reto Schelbert
11. Juni 2014

1 Dokumentinformationen

1.1 Änderungsgeschichte

Datum	Version	Änderung	Autor
10.04.2014	1.0	Erste Version des Dokuments	RS
19.04.2014	1.1	Web Environment after git	RS

2 Inhalt

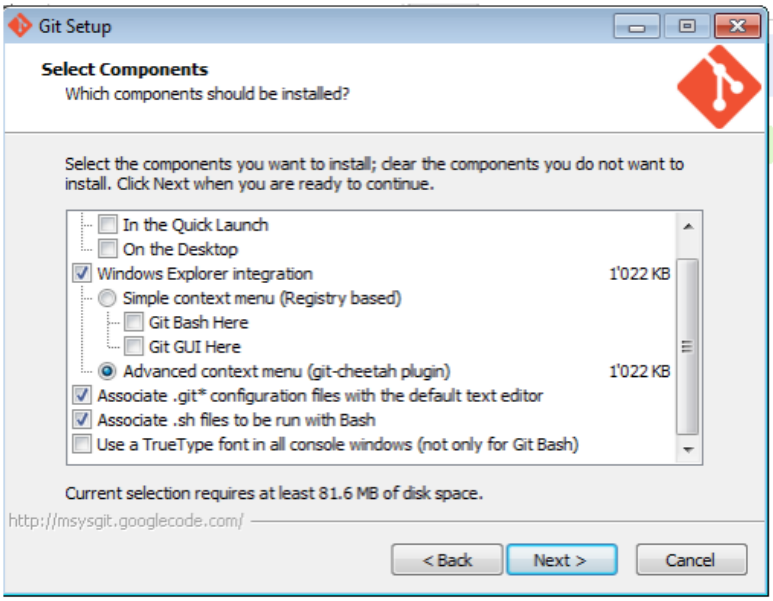
1 Dokumentinformationen 1
1.1 Änderungsgeschichte 1
3 Git 2
4 Web Environment 3
5 Configuration sfitixi 4
5.1 MySQL connection infos 4
5.2 PHP 4
5.3 openssl 5
5.4 XDebug 5
6 Install script 6
6.1 Windows Shell 6
6.2 run_build.sh 6
7 Manual Installation 8
8 Test Website 9

3 Git

Use Git under Windows with **msysgit**

http://git-scm.com/download/win

Best way: Install msysgit with the .sh extension



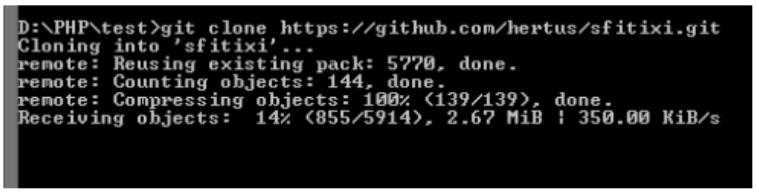
Or use **Github**:

https://windows.github.com/

It will install an **Git Shell**

Git clone in work directory

git clone https://github.com/hertus/sfitixi.git



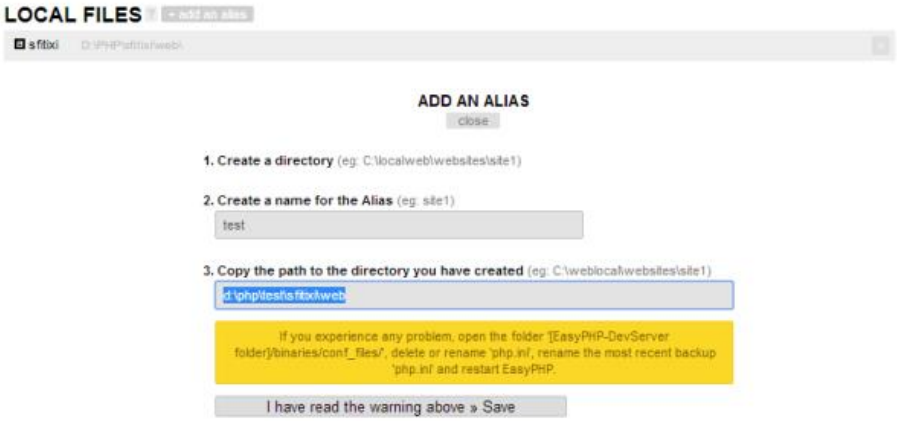
4 Web Environment

MySQL 5.6.x and PHP 5.5.x should be up and running.

For Windows we can use the easy installation package easyPHP, V14.1 VC11:
http://www.easyphp.org/save-easyphp-devservervc11-latest.php

Notice down the MySQL Credentials

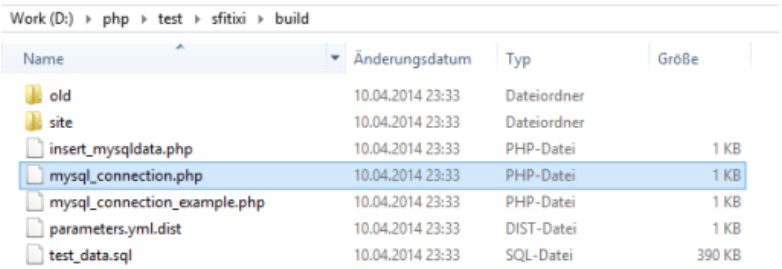
Best way to start your site later: create an alias to the web directory / working directory



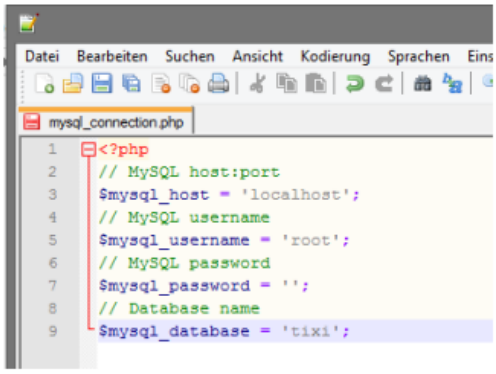
5 Configuration sfitixi

5.1 MySQL connection infos

Copy **mysql_connection_example.php** (for test data import scripts) to **mysql_connection.php** in **sfitixi/build** directory



Change connection infos to your mysql server and db-name



5.2 PHP

We should be able to start **php.exe** from CMD / Console / Shell, so make shure php is in PATH:

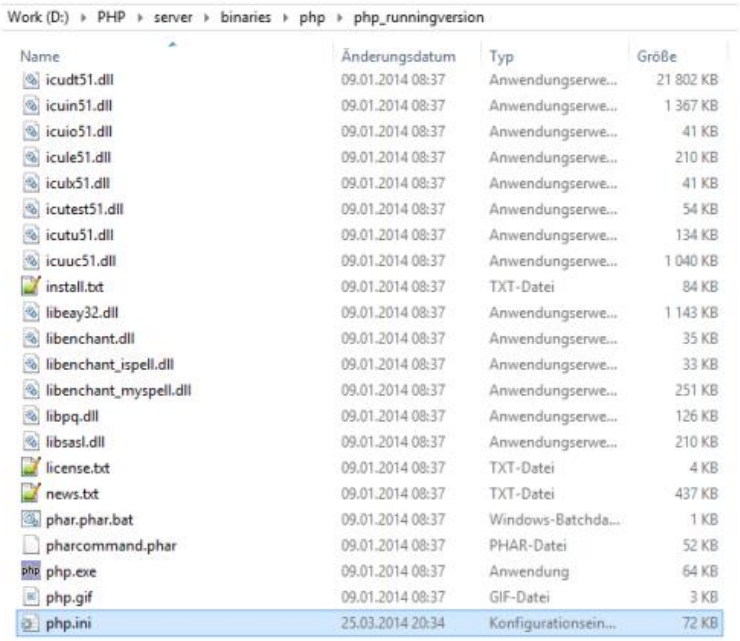
```
D:\>set path
Path=d:\PHP\server\binaries\php\php_runningversion;C:\Program
ommon;C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Fil
```

Set it with admin rights (restart CMD after this)

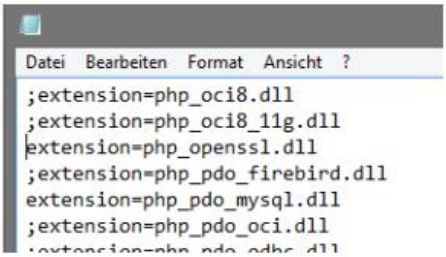
set PATH=%PATH%;D:\PHP\server\binaries\php\php_runningversion

5.3 openssl

On default installation of easyPHP, we have to enable the openssl extension to allow composer to download from https sources. So edit your php.ini



Search for **openssl** and enable the extension



5.4 XDebug

Set following in php.ini for the nested level (recursive actions need more)
xdebug.max_nesting_level=200

Restart Apache after this

6 Install script

6.1 Windows Shell

Go to the root directory sfitixi run the .sh script **run_build.sh**, confirm with yes.

We use Shell under Windows, like Github Shell (which get installed within the Github for Windows)

Windows CMD with msysgit: **run_build.sh**

```
D:\PHP\test\sfitixi>run_build.sh
Welcome to Git (version 1.9.0-preview20140217)
```

Git Shell: **sh ./run_build.sh**

```
C:\PHP\sfitixi [master]> sh .\run_build.sh
Starting new project build
.
This will completely recreate the TIXI project database! Confirm with y(es) or n
(o):
```

6.2 run_build.sh

Start the **run_build.sh** script under the sfitixi root-directory

```
D:\PHP\test\sfitixi>run_build.sh
Welcome to Git (version 1.9.0-preview20140217)


Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.
Starting new project build
.
This will completely recreate the TIXI project database! Confirm with y(es) or n(o):  y
.
Starting Composer dependencies
Loading composer repositories with package information
Installing dependencies (including require-dev)
  - Installing twig/twig (v1.15.1)
    Downloading: 100%

  - Installing doctrine/lexer (v1.0)
    Downloading: 100%

  - Installing doctrine/annotations (v1.1.2)
    Downloading: 100%

  - Installing doctrine/collections (v1.2)
    Downloading: 100%

  - Installing doctrine/cache (v1.3.0)
    Downloading: 100%

  - Installing doctrine/inflector (v1.0)
    Downloading: 100%

  - Installing doctrine/common (v2.4.1)
    Downloading: 100%

  - Installing symfony/icu (v1.0.1)
    Downloading: 100%

  - Installing symfony/symfony (v2.3.12)
    Downloading: 55%
```

composer install will install all dependencies and create the parameters.yml for symfony,

install assets and clear cache

Initial composer will set database connection, mail and timezone infos as we need it

```
Creating the "app/config/parameters.yml" file
Some parameters are missing. Please provide them.
database_driver (pdo_mysql):
database_host (127.0.0.1):
database_port (null):
database_name (tixi):
database_user (root):
database_password (null):
mailer_transport (smtp):
mailer_host (127.0.0.1):
mailer_user (null):
mailer_password (null):
locale (en): de_DE
secret (ThisTokenIsNotSoSecretChangeIt):
time_zone (Europe/Zurich):
Clearing the cache for the dev environment with debug true
```

This values will be saved in **sfitixi/app/config/parameters.yml**

After composer installed dependencies, the run_build script will create database schemas with doctrine and insert .SQL test-data with **insert_mysqldata.php** in **sfitixi/build**, which is using the **mysql_connection.php**

```
Recreate database
Dropped database for connection named `tixi`
Created database for connection named `tixi`
Updating database schema...
Database schema updated successfully! "70" queries were executed
Alter address table for fulltext search executed
.
Insert test_data
Tables from test_data.sql imported successfully!

D:\PHP\test\sfitixi>
```

7 Manual Installation

If run_build.sh script fails, we can just use the command in the script:

In sfitixi folder run the following commands:

php composer.phar install

php app/console doctrine:database:drop --force
php app/console doctrine:database:create
php app/console doctrine:schema:update --force
php app/console project:build-fulltext

cd build
php insert_mysqldata.php test_data.sql

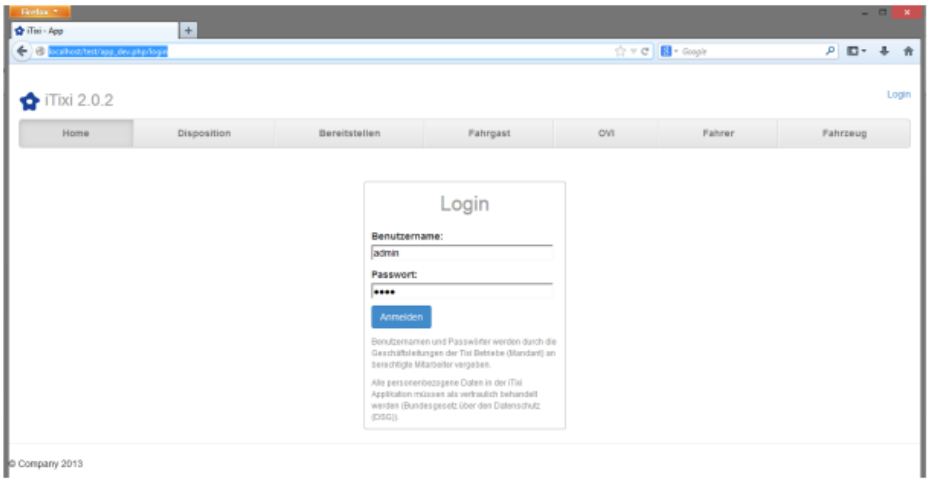
Or use MySQL Editor (like Workbench) to import the test_data.sql into the tixi database

Notice:
project:build-fulltext will only succeed if at least MySQL version 5.6 is installed

8 Test Website

If all succeeded, we should be able to start the site

http://localhost/test/app_dev.php/login



19.5.2 Technologie-Deployment

iTixi

Technologie Deployment

Definition

Carmelo Schumacher, Reto Schelbert
12. Juni 2014

1 Dokumentinformationen

1.1 Änderungsgeschichte

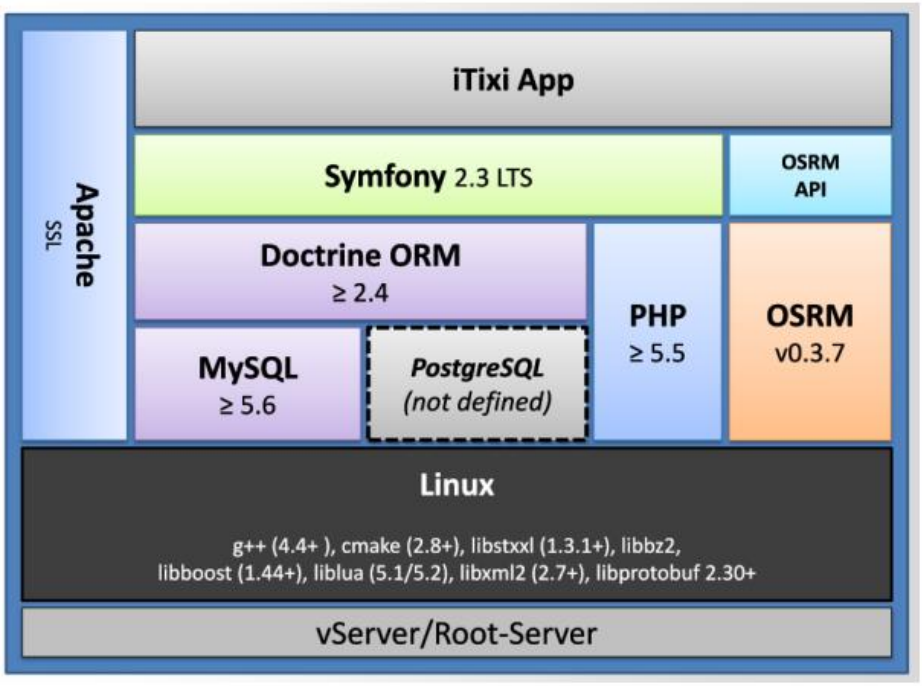
Datum	Version	Änderung	Autor
19.04.2014	1.0	Erste Version des Dokuments	RS

1.2 Inhaltsverzeichnis

1 Dokumentinformationen 1
1.1 Änderungsgeschichte 1
1.2 Inhaltsverzeichnis 1
2 Hardware und Software 2
2.1 Betriebssystem 3
2.1.1 Distribution 3
2.1.2 Libraries 3
2.2 OSRM 4
2.2.1 Installation OSRM auf Debian 7 4
2.3 Apache 7
2.4 MySQL 7
2.5 PHP 7

2 Hardware und Software

Vorläufig sieht der benötigte Technology-Stack folgendermassen aus:



Nachfolgend werden die benötigten Komponenten im Detail aufgeführt.

2.1 Betriebssystem

Um alle Komponenten auf einer Instanz zu betreiben eignet sich nur ein Linux / Unix Derivat als Betriebssystem. Die ist vor allem durch die Open Street Routing Machine **OSRM** bedingt, ein Windows OS scheidet ganz aus.

2.1.1 Distribution

Alle Komponenten sind momentan erfolgreich im Einsatz unter Debian 7 Wheezy (http://tixi.seiout.ch)

Es lässt sich ein Linux System mit Kernel > 3.2 empfehlen. Debian 7 Wheezy oder Ubuntu 12.04 mit Long Term Support **LTS**, oder aktuelle Versionen von CentOS/RHEL sowie Fedora sind sicher geeignet.

Ältere Distributionen können grosse Probleme bereiten mit neueren Versionen von Libraries wie LUA.

Auch werden nicht immer aktuelle Versionen von den Hersteller-Repositories angeboten.

PHP 5.5 auf Debian Wheezy muss z.B. manuell kompiliert oder über 3rd-Party Repositories bezogen werden da aktuell nur die Version 1.

2.1.2 Libraries

Die aufgeführten Libraries beziehen sich hauptsächlich auf OSRM.

Alle Libraries und Dependencies zum LAMP Stack mit Apache, MySQL, PHP werden vorausgesetzt.

2.2 OSRM

http://project-osrm.org/

Für Installation/Build auf verschiedenen Linux Distributionen siehe:
https://github.com/DennisOSRM/Project-OSRM/wiki/Building%20OSRM

Benötigte Compiler :

- g++ 4.4+

Third Party Dependencies:

- Boost 1.44+
- libxml2 2.7+
- cmake 2.8+
- stxxl 1.3.1+
- libprotobuf 2.30+
- lua 5.1/5.2
- bz2
- xml2

2.2.1 Installation OSRM auf Debian 7

Auf Debian 7 Wheezy wurde eine Installation von OSRM vorbereitet.

2.2.1.1 Dependencies

Dependencies installieren

```
apt-get install git libboost-dev gcc g++ cmake libstxxl-dev \
libxml2-dev libbz2-dev zlib1g-dev libzip-dev libboost-filesystem-dev \
libboost-thread-dev libboost-system-dev libboost-regex-dev libboost-program-options-dev \
libboost-iostreams-dev libgomp1 libpng-dev libprotoc7 libprotobuf-dev protobuf-compiler \
liblua5.1-0-dev libluabind-dev pkg-config libosmpbf-dev
```

LUA wurde in Version 5.1 und 5.2 installiert

```
aptitude install lua5.1 liblua5.1 liblua5.1-dev
aptitude install lua5.2 liblua5.2 liblua5.2-dev libluabind-dev
```

Zu C++ BOOST muss das Root-Verzeichnis der Libraries bekannt gemacht werden

```
declare -x DBOOST_ROOT=/usr/include/boost
```

2.2.1.2 OSRM Kompilieren

Version herunterladen und entpacken

```
wget https://github.com/DennisOSRM/Project-OSRM/archive/v0.3.7.zip
unzip v0.3.7.zip
```

Die stxxl Config wurde erstellt um spezifisch ein large data set anzulegen

.stxxl

```
disk=/tmp/stxxl,2000,syscall
```

Im entpackten OSRM Directory ein build anlegen und das Kompilieren starten

```
mkdir -p build
cd build
cmake ..
make
```

Wurde alles erfolgreich kompiliert, müssen noch Symlinks zum entsprechenden Routing Profile angelegt werden

```
ln -s ../profiles/car.lua profile.lua
```

es kann u.U. nötig sein auch noch die Profile Library zu linken

```
ln -s ../profiles/lib/
```

Danach zu den kompilierten OSRM Dateien im build Ordner jeweils einen Link erstellen ins Root-Dir von OSRM

```
ln -s build/osrm-extract
ln -s build/osrm-prepare
ln -s build/osrm-routed
```

2.2.1.3 OSRM OSM Extraction

OSRM bedient sich aus OpenStreetMap Data Extracts um seine Routen zu berechnen. Dafür werden .OSM Files benötigt, welche dann mit osrm-extract auf die benötigten Datenwerte ausgelesen werden. Dieses Prozess ist Arbeitsspeicher-instensiv und kann nur schon für die OSM Schweiz (http://download.geofabrik.de/europe/switzerland-latest.osm.pbf) gerne mal 4GB benötigen.

Entsprechend viel Memory oder Swap-Speicher sollte deshalb vorhanden sein.

Um die benötigten Daten aus OSM gering zu halten um etwas Memory einzusparen, kann OSMCONVERT verwendet werden. Siehe: http://wiki.openstreetmap.org/wiki/DE:Osmconvert#Download

Installation per:

```
wget -O - http://m.m.i24.cc/osmconvert.c | cc -x c - -lz -O3 -o osmconvert
```

Folgendermassen kann man nun aktuelle OSM Dateien herunterladen und für OSRM präparieren:

```
/ROOT-DIR von OSRM/
cd build

#Aktuelle OSM herunterladen
wget http://download.geofabrik.de/europe/switzerland-latest.osm.pbf

#OSM convertieren
./osmconvert switzerland-latest.osm.pbf -o=swiss.o5m
./osmconvert swiss.o5m -o=swiss.osm.pbf

#OSM extrahieren
./osrm-extract build/swiss.osm.pbf
./osrm-prepare build/swiss.osrm

#Start OSRM
./osrm-routed
```

2.3 Apache

Eine spezifische Version wird nicht benötigt.

Apache sollte SSL fähig ausgestattet sein um HTTPS Verbindungen mit einem vertrauenswürdig ausgestellten Zertifikat zu unterstützen.

2.4 MySQL

MySQL muss in der Version 5.6 oder höher vorliegen.

Erst ab dieser Version ist Fulltext-Search für InnoDB vorhanden
http://dev.mysql.com/doc/refman/5.1/en/fulltext-search.html

2.5 PHP

PHP muss in der Version 5.5 oder höher vorliegen.

BCrypt Support und OPCache ab dieser Version bieten grosse Vorteile und werden genutzt.