Projektdokumentation

Projektgruppe Pimp My Taxi Fleet

SS 2016 - WS 2016/2017

Gruppe:
Torsten de Buhr
Merlin Büsing
Bastian Ehrhardt
Jens Freesemann
Michael Kröger
Christoph Langen
Gérard Paschke
Frederik Preuss

Betreuer:
Tobias Brandt
Michael Brand

31. März 2017

Zusammenfassung

Die weltweite Mobilität von Menschen steht vor tiefgreifenden Veränderungen. Vor allem Klimaschutzaspekte und das Erschöpfen der Öl- und Gasreserven sind wesentliche Kräfte dieser Veränderungen. Langfristig kann die Mobilität in der aktuellen, auf fossilen Kraftstoffen basiertenden Form nicht mehr gewährleistet werden. Erschwerend hinzu kommt, dass die Anzahl der Fahrzeuge auf den Straßen immer weiter steigt. Dies führt zu Staus, kaputten Straßen usw. Als Konsequenz dessen steigt die Nachfrage nach neuen, alternativen Mobilitätskonzepten und -lösungen.

Im Rahmen dieses Bericht wird beschrieben wie sich die Projektgruppe „Pimp my Taxi fleet“ mit einem dieser alternativen Mobilitätskonzepte beschäftigt. Dabei geht es um die Verwaltung einer Flotte von elektronischen und autonom fahrenden Taxis im Umfeld der Stadt Oldenburg. Zu diesem Zweck wurde ein System entwickelt, dass Echtzeitdaten von Taxis und
Kundenaufträgen auswertet und auf Basis dessen eine *intelligente Routen- und Standortplanung* für die Taxis vornimmt.

Das System ist dazu gedacht, die Taxiflotte eines Unternehmen zu überwachen. Dabei optimiert es mithilfe des *DDARP* die Routen der Taxis und kombiniert Aufträge der Kunden um Strecke, Fahrtzeit usw. zu minimieren. Die Daten für diese Planung werden von dem Datenstrommanagementsystem *Odysseus* verarbeitet und kommen von einer abkapselbaren Verkehrssimulation und einer Android-App. Die Berechnung der Route innerhalb von *Odysseus* sowie die Auftragszuweise erfolgt durch *GraphHopper* und *JSprit*.

Weiterhin nimmt das System ein intellgente Standortplanung der Taxis vor, sodass die Strecken zum Kunden möglichst gering sind und diese schnell bedient werden können. Zu diesem Zweck verarbeitet es zum einen *Facebook-Veranstaltungsdaten* und zum anderen überwacht es das aktuelle Auftragsaufkommen und erfasst Zonen mit erhöhtem Auftragsaufkommen und sendet Taxis direkt dort hin.

Zur Überwachung der Taxiflotte und zum anzeigen von Statistiken und Analysen wurde zudem ein *Dashboard* für die Mitarbeiter des Taxiunternehmen entwickelt.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Dieser Abschlussbericht ist das Ergebnis der Projektgruppe **Pimp my Taxi Fleet** an der Carl von Ossietzky Universität Oldenburg, Department für Informatik, Abteilung Informationssysteme. Die Projektgruppe besteht aus acht Personen, begann im April 2016 und wurde im März 2017 abgeschlossen. Im Rahmen dieser Projektgruppe wurde das abteilungsintern entwickelte Datenstrommanagementsystem **Odysseus** zur automatisierten Verwaltung einer autonomen eTaxi Flotte verwendet.

1.1 Motivation

Die Entwicklung moderner Transportmittel ist ein wesentlicher Faktor für die Entwicklung der Menschheit und befindet sich im stetigen Wandel. Wesentliches Hilfsmittel im flächendeckenden Individualverkehr stellt heutzutage für eine breite Masse der Bevölkerung das Automobil dar. Aktuelle Entwicklungen, wie staatliche Subventionen beim Kauf von Elektroautos und Förderungen beim Ausbau der Ladeinfrastruktur, deuten zunächst auf eine Ablösung des Verbrennungsmotors durch einen elektronischen Antrieb hin.[1] Längerfristig lassen sich Ansätze von Technologiekonzernen wie Google, aber auch von traditionellen Automobilherstellern, erkennen, die über Assistenzsysteme den Fahrer[2] unterstützen und mit der Zeit vollständig durch autonom agierende Systeme ersetzen. [3] Unter Berücksichtigung der voraussichtlichen Rahmenbedingungen soll das Konzept der individuellen Fortbewegung ohne eigenes Fahrzeug auf Abruf betrachtet werden.
Für Personen ohne eigenes Auto und dem Bedürfnis nach individueller und komfortabler Fortbewegung waren Taxis lange Zeit die einzige Wahl. Neuere Konzepte wie Online-Angebote über Mitfahrmöglichkeiten, Mietwagen mit Fahrer (*Uber*[4]) oder Carsharing stellen durch günstigere und vorkalkulierte Preise eine Konkurrenz für die herkömmliche Taxibranche dar und erweitern das bestehende Angebot. Ausge-

[1] `https://www.bundesregierung.de/Content/DE/Artikel/2016/05/2016-05-18-elektromobilitaet.html`

[2] Aus Gründen der besseren Lesbarkeit wird in dieser Dokumentation auf die gleichzeitige Verwendung weiblicher und männlicher Sprachformen verzichtet. Für die entsprechenden Begriffe ist sowohl die weibliche als auch die männliche Form gemeint.

[3] `https://www.google.com/selfdrivingcar/`

[4] `https://www.uber.com/de/`

hend von der zukünftigen Entwicklung zu autonom fahrenden Elektrofahrzeugen, wäre eine Dienstleistung denkbar, bei der selbstfahrende Taxis vom Kunden über eine Smartphone App bestellt werden können. Diese autonomen Fahrzeuge können, anders als in der bisherigen Praxis, eine Wegoptimierung vornehmen und mehrere Kunden, die auf einer passenden Strecke liegen, befördern. Dies minimiert nicht nur das Verkehrsaufkommen, sondern auch den Fahrpreis für den Kunden und könnte einen erheblichen Wettbewerbsvorteil erzielen.

1.2 Problembeschreibung und Zielsetzung

Das zuvor beschriebene Konzept einer autonomen Taxiflotte benötigt trotz den autonom agierenden Fahrzeugen eine Verwaltungseinheit und einen Verbindungspunkt zum Kunden. Um einen Beförderungswunsch auf einfache Weise mitzuteilen, könnte ein potentieller Kunde über eine Smartphone App (ähnlich wie myTaxi oder Uber) alle notwendigen Informationen zur gewünschten Fahrt zum Taxiunternehmen übermitteln. Eine Software zur Taxiverwaltung könnte daraufhin automatisiert eine optimierte Zuweisung der eingehenden Aufträge zu einem Taxi und dessen Tourenplanung vornehmen.
Dabei muss die Verwaltungseinheit auf eine Vielzahl von eingehenden Datenströmen, wie der aktuellen Verkehrslage, eingehende Aufträge, Standort und Akkuladung der Taxis in Echtzeit reagieren. Zur internen Verarbeitung der vorliegenden Datenströme soll das DSMS Odysseus zum Einsatz kommen. Basierend auf den eingehenden Daten sollen Aufträge naheliegenden, verfügbaren Taxis zugewiesen werden und das zugewiesene Taxi die Route zunächst zum Kunden, bis hin zum gewählten Beförderungsendpunkt erhalten. Die Routengenerierung kann unter Berücksichtigung der aktuellen Verkehrslage optimiert werden, indem Straßen mit erhöhtem Verkehrsaufkommen vermieden werden. Neben der Auftragszuweisung und Routenplanung ist für autonome Elektroautos ebenso die intelligente Energieverwaltung zwischen den Aufträgen relevant. So können Wartezeiten ohne Kundenauftrag zur Aufladung des Akkus genutzt werden oder anstehende Aufträge an andere verfügbare Taxis umverteilt werden. Besonderes Augenmerk bei der Energieverwaltung ist zudem während der Auftragszuweisung auf die Ladestände der Taxis zu richten, sodass das Taxi nicht nur ausreichend Reichweite für den nächsten Auftrag, sondern auch zur nächstgelegenen Ladestation be-

reithält. Ein weiterer Erfolgsfaktor kann durch die optimierte Standortplanung noch frei verfügbarer Taxis erzielt werden. Im optimalen Fall zeichnet sich dies in einer Standortvorhersage für zukünftige Aufträge ab, sodass noch bevor der Kunde vor Ort den Auftrag aufgibt, ein verfügbares Taxi bereits im Einzugsbereich bereitsteht. Grundsätzliches Ziel dieser Standortplanung ist eine Verringerung der durchschnittlichen Wartezeit der Kunden zwischen Absenden des Auftrags und Ankunft des zugewiesenen Taxis am Abholort. Zur Realisierung der Vorhersage können ortsbezogene Statistiken über vorhergegangene Aufträge, aber auch aktuell eingehende Bedingungen wie Wettervorhersagen oder in Sozialen Netzwerken eingetragene Veranstaltungen einbezogen werden. Da der Projektgruppe zum Zeitpunkt der Softwareentwicklung keine frei verfügbaren autonomen Fahrzeuge mit offenen Schnittstellen zur Verfügung standen, sollte eine Verkehrssimulation für den Raum Oldenburg genutzt werden. Diese liefert der Taxiverwaltung Informationen über die modellierte Verkehrssituation in Echtzeit und simuliert die an die Taxis gesendeten Routen.
Das Ziel der Projektgruppe ist somit die Realisierung der erwähnten Softwarekomponenten zur automatisierten Verwaltung einer autonomen eTaxiflotte. Zur Realisierung soll eine Verkehrssimulation genutzt werden. Aufträge können von einem Auftragsgenerator erzeugt oder händisch über eine Smartphone App eingegeben werden. Die auf dem Prinzip von Datenströmen basierende Taxiverwaltung soll eingehende Aufträge verarbeiten und die errechnete Route an das zugewiesene Taxi in der Verkehrssimulation schicken. Eine visuelles Lagebild sowie detaillierte Informationen zu den Taxis und den zugewiesenen Aufträgen können über ein Dashboard dargestellt werden.

1.3 Aufbau der Dokumentation

Dieser Abschlussbericht soll die Erfahrungen der Projektgruppe Taxi für mögliche Stakeholder vermitteln und das erlangte Wissen im Projekt dokumentieren. Der Bericht führt somit die Ergebnisse sowie die Vorgehensweise der Projektgruppe auf. Hierzu wird zunächst im Kapitel 2 das Projektteam mit seinen einzelnen Aufgabenbereichen und Organisationsform, dem angewandten Vorgehensmodell und der verwendeten Software beschrieben. Für ein besseres Verständnis der Thematik werden anschließend die notwendigen Grundlagen geschaffen. Es folgen im

Kapitel 4 die funktionalen und nicht funktionalen Anforderungen, die zur entwickelnden Softwarelösung erarbeitet wurden. Basierend auf den erhobenen Anforderungen beschreibt der Systementwurf im Kapitel 6 die Software - ausgehend von ersten Konzepten mit Mockups zu Smartphone App und Dashboard, über die Logik der einzelnen Komponenten wie der Routenberechnung und -Optimierung hinweg, bis zur gesamten Systemarchitektur. Es enthält außerdem weitreichende Details zur Implementierung. Dabei wird die Modellierung über verschiedene UML-Diagramme beschrieben, Einblicke in den Quellcode gegeben und anhand der Beschreibung von Testfällen die durchgeführten Qualitätssicherungsmaßnahmen in der Softwareentwicklung geschildert. Abschließend reflektiert ein Fazit im letzten Kapitel 10 die Ergebnisse der Projektgruppe und gibt einen Ausblick über zukünftige Ansatzpunkte einer Weiterentwicklung.

2 Organisation

2.1 Aufgabenbereiche

Im Rahmen der Projektorganisation hat die Projektgruppe wichtige Aufgabenbereiche festgestellt, die den einzelnen Mitgliedern der Gruppe als Rolle zugewiesen wurden. Im Folgenden werden diese Aufgabenbereiche jeweils erläutert.

2.1.1 ScrumMaster

Der ScrumMaster ist verantwortlich für den gesamten Scrum-Prozess im Projekt. Er ist dafür zuständig den Prozess funktionsfähig zu halten und eventuelle Hindernisse oder Probleme zu beseitigen. Dabei ist der ScrumMaster jedoch kein klassischer Projektmanager, sondern nur für den Scrum-Prozess verantwortlich und mit weniger Kompetenzen als ein Projektmanager ausgestattet. Er ist verantwortlich für eine effektive Arbeitsweise des Umsetzungsteams und unterstützt die eigenständige Organisation des Teams. Der ScrumMaster ist zugleich Mediator, Moderator und Teamcoach. Er hat die Aufgabe effektive Meetings zu organisieren und Techniken einzubringen, die die Kreativität des Teams fördern bzw. zum Einsatz bringen. Weiterhin sind ScrumMaster verantwortlich für die Durchführung der Schätzung der einzelnen Tasks und User Stories sowie für das Sprint Planning. Sie tragen die Verantwortung für die Durchführung des Sprint Reviews und der Retrospektive am Ende jedes einzelnen Sprints. Hierbei sollen Erkenntnisse gewonnen werden, wie sich der Teamprozess verbessern und effizienter gestalten lässt. Auch die individuelle Weiterentwicklung der Teammitglieder soll der ScrumMaster unterstützen und ihnen helfen ihr volles Potenzial auszuschöpfen. Die Aufgaben des ScrumMasters im Einzelnen sind die Folgenden:

- Unterstützung des Teams bei der selbstständigen Organisation der Arbeit.
- Stellt sicher, dass das Team produktiv und funktional sein kann.
- Beseitigung von Hindernissen.
- Schutz vor externen Störungen.
- Unterstützt die Zusammenarbeit aller Projektbeteiligten.

- Sicherstellung der Scrumprinzipien und -werte.
- Moderiert Projektsitzungen, Sprint Planning, Sprint Review und Retrospektive.
- Unterstützt beim Entscheidungsfindungsprozess, dem Finden von Problemlösungen und der Klärung von Konflikten.
- Führt, leitet an, berät, coacht und unterstützt das Team, um letztendlich ein selbstorganisiertes, hochproduktives Team zu werden.

Im Rahmen dieser Projektgruppe sind die oben genannten Funktionen und Aufgaben des ScrumMasters nahezu identisch. Dennoch gibt es einige kleine Abweichungen. So übernimmt beispielsweise jede Woche ein anderes Projektgruppenmitglied die Moderation der Sitzung und somit fällt ein Teil der Moderationsaufgabe des ScrumMasters weg. Es ist anzumerken, dass die Moderation und Durchführung von Aufwandsschätzung, Sprint Planning, Sprint Review und Retrospektive weiterhin ausschließlich vom ScrumMaster übernommen wird.

2.1.2 Confluence-Beauftragter

Zu den Aufgaben des Confluence-Beauftragen gehören die Erstellung und Pflege von grundlegenden Seiten auf der Plattform Confluence (siehe Abschnitt "Verwendete Software") zum Austausch von Wissen und Informationen der Projektteilnehmer. Damit eine einfache und schnelle Navigation möglich ist, muss er zudem dafür sorgen, dass stets eine übersichtliche Seitenhierarchie vorhanden ist.

2.1.3 Testbeauftragter

Der Testbeauftragte ist für die Umsetzung der Teststrategie und die Erstellung und Einhaltung der Testpläne zuständig. Hierfür ist vom Testbeauftragten ein Konzept für die Testumgebung und deren Bereitstellung sicherzustellen. Außerdem überwacht der Testbeauftragte den Fortschritt bei der Durchführung der Tests und die Erstellung der Testberichte.

2.1.4 Projektmanager

Üblicherweise fallen in die Aufgabenbereiche des Projektmanagers Tätigkeiten zur Planung, Steuerung und Kontrolle des Projekts. Da in diesem Softwareentwicklungsprojekt ein eigens definiertes Scrum als Vorgehensmodell gewählt wurde, unterscheidet sich das angewandte Projektmanagement von der üblichen Form. Im Gegensatz zum ScrumMaster ist der Projektmanager vielmehr für die mittel bis langfristige Planung des Projekts und die Einhaltung der gesetzten Ziele zuständig. Dabei werden zunächst die Interessen der Stakeholder transparent gemacht und festgehalten. Anschließend können die dokumentierten Anforderungen während der Projektplanung priorisiert werden. Auswahl und Einsatz geeigneter Projektmanagement-Werkzeuge, wie dem Festhalten geeigneter Zielgrößen im Meilensteinplan, wirken unterstützend. Bei der Durchführung von Sprints, beseitigen Projektmanager und ScrumMaster in Kooperation Unklarheiten und Hindernisse.

2.1.5 Dokumentations-Beauftragter

Der Dokumentations-Beauftragte ist für die Projektdokumentation zuständig. Diese wird dabei nicht von ihm alleine erstellt, sondern von allen Mitgliedern der Projektgruppe. Zu den Aufgaben des Dokumentations-Beauftragten gehören die Erstellung einer Dokumentvorlage, die Verwaltung und Qualitätssicherung der Dokumentation sowie die Sicherstellung einer zeitnahen Erstellung. Als Grundlage für die Erstellung der Dokumentation dient das Textverarbeitungsprogramm LaTeX.

2.1.6 Odysseus-Beauftragter

Die Rolle des Odysseus Beauftragten wurde zur Förderung der Integration und des korrekten Einsatzes von Odysseus im Projekt eingeführt. In diesem Rahmen wurde Odysseus initial auf dem Projektserver aufgespielt und als zentrale, lauffähige Testumgebung eingerichtet. Der Odysseus Beauftragte stellt den ersten Ansprechpartner bei auftretenden Fragen zur Implementierung mit Odysseus dar. Um einen Überblick über die anstehenden Aufgaben im Kontext zu wahren, soll der Beauftragte noch auszuführende Aufgaben ermitteln und auf eine zeitlich sinnvolle Umsetzung achten. Dies beinhaltet ebenso die Dokumentation. Änderungen am Umfeld sind zu prüfen und die Lauffähigkeit des Projektes auf dem Server ist sicherzustellen.

2.1.7 Verkehrssimulations-Beauftragter

Die Rolle des Verkehrssimulations-Beauftragten wurde geschaffen, weil die Auswahl einer geeigneten Verkehrssimulation eine wichtige Rolle zur Umsetzung des Projekts spielt. Deshalb hatte der Beauftragte die Aufgabe eine Analyse durchzuführen um Anforderungen, wie zum Beispiel eine geeignete Schnittstelle zum Datenaustausch, zu überprüfen. Nachdem die Simulation ausgewählt und lauffähig gemacht wurde, ist der Beauftragte dafür zuständig eine beratene Funktion einzunehmen, welche Funktionalitäten mit Bezug zur Simulation umgesetzt werden können. Außerdem ist er dafür zuständig, dass die Simulation nahezu immer lauffähig bleibt und überwacht Änderungen an den Szenarien innerhalb der Simulation.

2.2 Vorgehensmodell: Scrum

In diesem Kapitel wird mit *Scrum* das Vorgehensmodell dieser Projektgruppe beschieben. Dazu werden im Kapitel 2.2.1 zunächst relevante Grundlagen gelegt. Danach werden im Kapitel 2.2.2 die wesentlichen Rollen bei diesem Vorgehensmodell erläutert. Im Anschluss daran werden im Kapitel 2.2.3 das *Sprint Planning* und im Kapitel 2.2.4 das *Sprint Review* erläutert. Zu Abschluss werden im Kapitel 2.2.5 allgemein die *Retrospektive* sowie die von dieser Projektgruppe verwendeten Retrospektive-Techniken aufgezeigt.

2.2.1 Grundlagen

Das Vorgehensmodell der agilen Softwareentwicklung des Projekts ist Scrum. Im Scrum-Prozess wird das Projekt iterativ und inkrementell entwickelt. Ein Sprint stellt eine Iteration der Entwicklung dar. Die Gruppe hat sich für dieses Projekt auf Basis von Erfahrungswerten der Teilnehmer aus anderen Projekten für eine Sprintlänge von zwei Wochen entschieden.
Wichtige Elemente bei Scrum sind Rollen, Aktivitäten und Artefakte. Alle Aktivitäten werden im Scrum-Prozess in regelmäßigen Abständen durchgeführt. Bei jeder Sitzung der Gruppe beschreibt jedes Mitglied kurz, was es in der vergangenen Woche getan hat. Diese Aktivität wird Daily Scrum genannt, da sich im Normalfall alle Gruppenmitglieder täglich treffen. In unserem Fall findet ein Treffen jedoch wöchentlich statt. Daher wird diese Aktivität Weekly Scrum genannt.

Vor jedem Sprint, also alle zwei Wochen, findet ein Sprint Planning Meeting (siehe Kapitel 2.2.3) statt. Hier wird festgelegt, welche Features bzw. Tasks in dem folgenden Sprint umgesetzt werden sollen. Dazu wird in diesem Treffen der Aufwand jeder Aufgabe im Backlog geschätzt. Hierzu wurden von den Scrum-Mastern anfangs verschiedene Schätzmethoden vorgestellt. Nach Anwenden von Planning Poker und Magic Estimation in den ersten beiden Sprints entschied sich die Gruppe im weiteren Verlauf des Projekts im wesentlichen Magic Estimation als Schätzmethoden zu verwenden.
Nach jedem Sprint werden in Review und Retrospektive Erfahrungen des abgeschlossenen Sprints ausgetauscht (siehe Kapitel 2.2.4 und 2.2.5).

2.2.2 Rollen

Im Scrum-Prozess werden im wesentlichen drei Rollen definiert, nämlich die des Product Owners, Scrum Masters und Entwicklungsteams. Im Verlauf dieses Kapitels werden die einzelnen Rollen näher erklärt.

Product Owner Bei dem *Product Owner* handelt es sich prinzipiell um eine einzige Person, die die Anforderungen, Features festlegt und priorisiert. Dabei legt er zusammen mit dem Umsetzungsteams Tasks im Backlog an und priorisiert diese. Weiterhin ist der Product Owner für die Abnahme des Zwischenprodukts am Ende jedes Sprints verantwortlich. Dabei kommt der Product Owner häufig vom Auftraggeber und stellt eine Art Brücke zwischen Auftraggeber und Umsetzungsteam dar. In der Projektgruppe ist diese Rolle nicht so klar zu definieren, fällt aber am ehesten den zwei Betreuern zu, die großen Anteil daran tragen welche Features umgesetzt werden.

Scrum Master siehe 2.1.1

Umsetzungsteam Das Umsetzungsteam kümmert sich um die eigentlich Implementierung der Software. Dabei benutzt es die durch den Product Owner definierten Anforderungen und Tasks aus dem Backlog. Das Umsetzungsteam organsiert seine Arbeit im wesentlichen selbst und schätzt am Anfang jedes Sprint den Aufwand für die Tasks und wie viele Tasks es in dem kommenden Sprint schaffen kann.

Dies geschieht im sogenannten *Sprint Planning* (siehe Kapitel 2.2.3). In unserer Projektgruppe sind alle Teilnehmer auch Teil des Umsetzungsteams.
Es ist festzustellen, dass durch das universitäre Umfeld in dem die Projektgruppe stattfindet nicht alle Rollen vollständig Scrum-konform ausgefüllt werden. Allerdings sind die Rollen relativ nahe an dem normalen Scrumprozess.

2.2.3 Sprint Planning

Das *Sprint Planning* findet vor jedem neuen Sprint statt. Dort werden in vom Product Owner gestellten Anforderungen in Arbeitspakete, die sogenannten Tasks, zerlegt. Diese Tasks werden anschließend durch das Umsetzungsteam unter Nutzung von „Story Points“ geschätzt um festzustellen wie aufwändig die einzelnen Arbeitspakete sind. Weiterhin wird durch das Umsetzungsteam festgelegt wie viele Tasks bzw. „Story Points“ es im nächsten Sprint schaffen kann. Geleitet wird das Sprint Planning Meeting durch den Scrum Master.
Im Rahmen dieser Projektgruppe wurden folgende zwei Schätzverfahren verwendet um die „Story Points“ für die einzelnen Tasks durch die Gruppe zu schätzen:

Planning Poker Beim Planning Poker erhält jeder Teilnehmer zehn Karten mit folgenden Werten darauf: 0, 0.5, 1, 2, 3, 5, 8, 13, 20 und „?“. Diese Werte symbolisieren die „Story Points“ und das Fragezeichen symboliert, dass eine Schätzung nicht möglich ist. Nun werden die einzelnen Tasks durch den Scrum Master vorgelesen. Bei jedem Task wählt jeder Teilnehmer eine Karte und legt diese vor sich auf den Tisch. Anschließend werden die einzelnen Schätzungen diskutiert und ein Kompromiss für die Schätzung gefunden. Dieses Vorgehen wird so lange wiederholt, bis keine ungeschätzten Tasks mehr im Backlog sind.

Magic Estimation Für das Schätzverfahren *Magic Estimationen* wird ein Kartensatz der „Planning Poker“-Karten auf einem Tisch ausgelegt. Weiterhin werden alle Tasks und User Stories aus dem Backlog ausgeruckt und an die Teammitglieder verteilt. Bei dieser Art des Planning soll nicht gesprochen werden. Alle Mitglieder legen ihre Tasks zu den jeweiligen „Planning Poker“-Karten mit denen sie die Tasks schätzen wollen. Danach gucken sie sich die von den anderen Mitglieder gelegten Tasks an und können diese gegebenenfalls verschieben. Sollte auffallen, dass ein Task immer wieder verschoben wird, kann im Anschluss an das Verfahren separat

über die Schätzung dieses Tasks diskutiert werden. Auch dieses Vorgehen wird so lange wiederholt, bis keine ungeschätzten Tasks mehr im Backlog sind.

2.2.4 Sprint Review

Das Sprint Review findet am Ende jedes Sprints statt, dabei wird dem Product Owner bzw. im Fall der Projektgruppe den Betreuern, das im Sprint entwickelte Softwareprodukt vorgestellt. Dieser entscheidet nun ob die vorher definierten Anforderungen und Kriterien erfüllt sind und gibt gegebenfalls Ideen für Verbesserungen und neue Tasks.
Im Rahmen dieser Projektgruppe wurde das Sprint Review in der Regel auf zwei Arten durchgeführt. Zum einen wurden mittels Beamer die sich im Sprint befindetenden Tasks über Jira angezeigt. So wurde besprochen was in den einzelnen bereits abgeschlossenen Tasks getan wurde und wo eventuelle Verbesserungen notwendig sind. Die zweite Möglichkeit waren Live-Demos einzelner Komponenten. Dies erfolgte in der Regeln auch über den Beamer.

2.2.5 Retrospektive und Retrospektive-Techniken

Am Ende jedes Sprints wird durch die Scrum Master und das Umsetzungsteam eine sogenannte *Retrospektive* durchgeführt. Hierbei wird durch das Umsetzungsteam aufgezeigt was gut bzw schlecht lief im letzten Sprint. Dabei werden Problem und Schwachstellen erkannt und Lösungen erarbeitet. Aus der Retrospektive soll das Team lernen und so die festgestellten Probleme abstellen und effizienter arbeiten. Im Rahmen der Projektgruppe wurden, zu einer möglichst vielfältigen und abwechselungsreichen Gestaltung der Sprint-Retrospektiven, verschiedene Retrospektive-Techniken angewandt. Alle verwendeten Techniken werden im Verlauf diese Kapitels einzeln beschrieben. Dabei handelt es sich um die Folgenden:

Stafish – Wheel (Einfache Form) Bei der einfachen Form der Starfish-Technik wird ein Kreis auf eine Tafel, ein Whiteboard o. ä. gezeichnet. Dieser Kreis wird anschließend in drei gleich große Segmente unterteilt. Danach werden diese Segmente mit mit den Termen „Start doing“, „Stop doing“ und „Continue doing“ beschriftet. Anschließend werden die folgenden Fragen zu den aufgeführten Termen vom Scrum-Master an das Team gestellt:

- **Start doing:** Womit können wir **anfangen** um den Team-Prozess und den Projektfortschritte zu beschleunigen?
- **Stop doing:** Womit können wir **aufhören**, dass den Team-Prozess und den Projektfortschritte aktuell behindert oder verlangsamt?
- **Continue doing:** Womit können wir **fortfahren**, dass aktuell den Team-Prozess und den Projektfortschritte fördert?

Im Anschluss daran haben die Teammitglieder mehrere Minuten Zeit um sich Antworten zu überlegen und diese auf kleine Zettel zu schreiben. Abschließend werden diese Zettel durch die Mitglieder in die entsprechenden Segmente an der Tafel geheftet und so gesammelt.
Der Aufbau der Starfish-Technik und ein beispielhaftes Ergebnis einer Retrospektive unserer Gruppe lässt sich Abbildung 1 entnehmen.

Stafish – Wheel (Erweiterte Form) Die Starfish-Technik in der erweiterten Form ist der einfachen Starfish-Technik prinzipiell sehr ähnlich. Zu Beginn wird ebenfalls ein Kreis an eine Tafel gezeichnet. Dieser wird allerdings nicht in drei, sondern in fünf Segmente unterteilt. Beschriftet werden diese Bereiche mit „Start doing“, „Stop doing“, „Keep doing“, „More of“ und „Less of“. Zu den einzelnen Bereichen stellt der Scrum-Master folgende Fragen:

- **Start doing:** Womit können wir **anfangen** um den Team-Prozess und den Projektfortschritte zu beschleunigen?
- **Stop doing:** Womit können wir **aufhören**, dass den Team-Prozess und den Projektfortschritte aktuell behindert oder verlangsamt?
- **Keep doing:** Womit können wir **fortfahren**, dass aktuell den Team-Prozess und den Projektfortschritte fördert?
- **More of:** Was hilft aktuell dem Team-Prozess und dem Projektfortschritte und wovon können wir **mehr machen**?
- **Less of:** Was behindert aktuell den Team-Prozess und den Projektfortschritte und wovon können wir **weniger machen**?

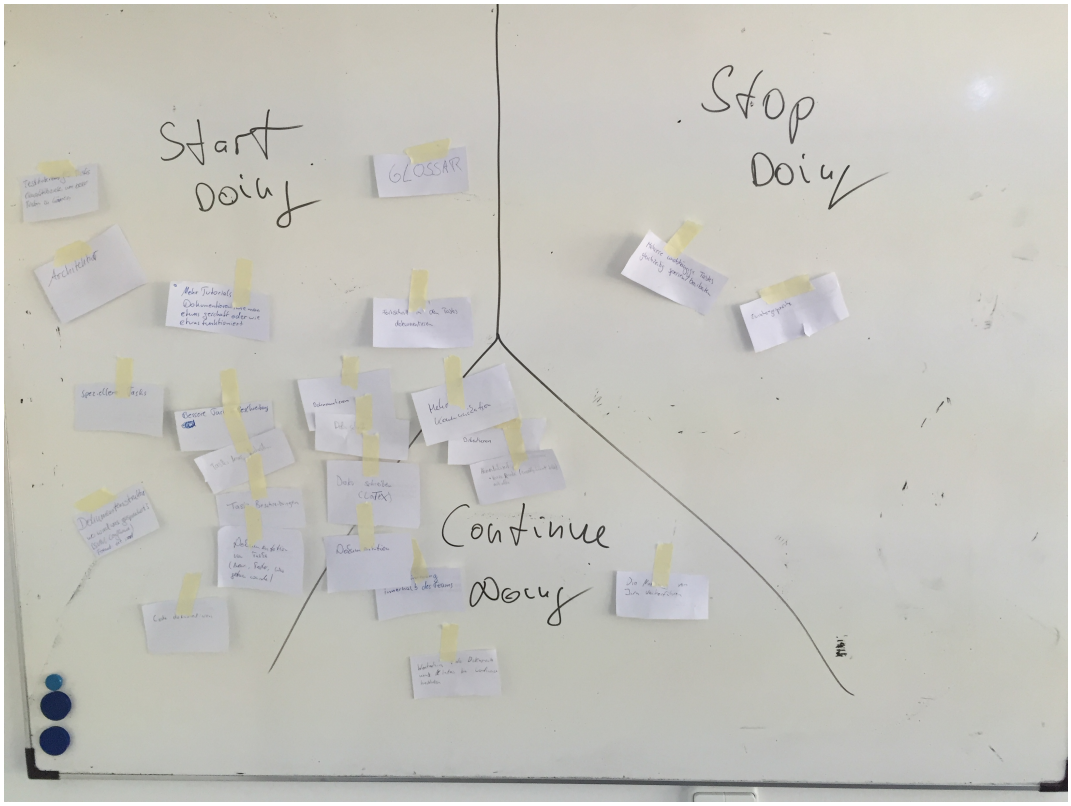


Abbildung 1: Beispiel für die einfache Form der Starfish-Technik (Bild der Ergebnisse des ersten Sprints)

Im Anschluss daran haben die Teammitglieder mehrere Minuten Zeit um sich Antworten zu überlegen und diese auf kleine Zettel zu schreiben. Abschließend werden diese Zettel durch die Mitglieder zu den entsprechenden Punkten an der Tafel geheftet und so gesammelt.
Der Aufbau der Starfish-Technik und eine beispielhaftes Ergebnis einer Retrospektive unserer Gruppe lässt sich Bild 2 entnehmen.

Sailboat Beim Sailboat-Verfahren wird zuerst ein Segelboot auf eine Tafel gemalt. Anschließend werden die folgenden Elemente hinzugefügt: Wind, ein Anker, Felsen im Meer bzw. ein Piratenschiff (In der Regel wird nur eines der Elemente in der Retrospektive verwendet. Welches gewählt wird, bestimmen die Präferenzen des Scrum-Masters.) und eine Insel. Richtig angeordnet sehen die Elemente wie in Abbildung 3 aus. Danach erklärt der Scrum-Master dem Team die einzelnen

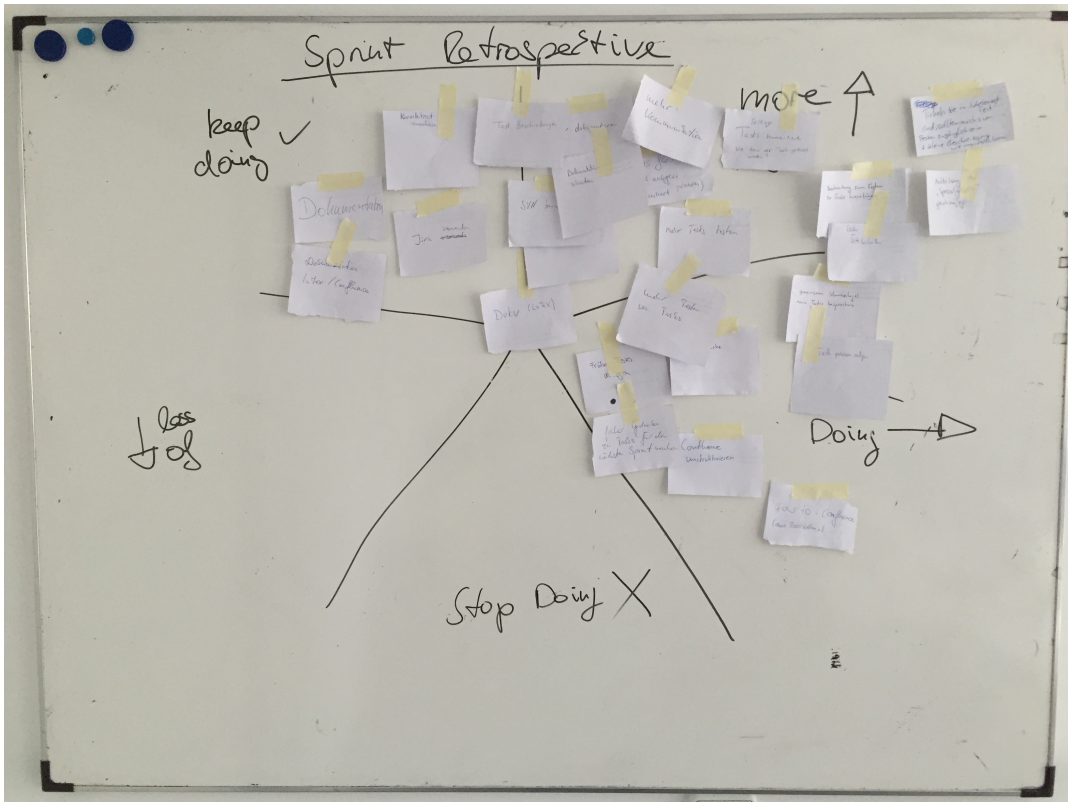


Abbildung 2: Beispiel für die erweiterte Form der Starfish-Technik (Bild der Ergebnisse des zweiten Sprints)

Metaphern. Dabei stehen die Metaphern für Folgendes:

- **Insel:** Die Insel steht für das abgeschlossene fertige Produkt, also den Projektabschluss und die Erreichung des Ziels.
- **Segelboot:** Das Segelboot stellt das Projekt bzw. den aktuellen Projektfortschritt dar. Es befindet sich entfernt von der Insel, da das Produkt noch nicht ferig ist. Weiterhin besitzt das Segelboot einen Anker.
- **Anker:** Der Anker ist direkt mit dem Segelboot verbunden. Er steht für Dinge die den Team-Prozess und den Projektfortschritt zurückhalten und behindern.
- **Wind:** Der Wind treibt das Segelboot an und steht für alle Gelegenheiten und positiven Aspekte, die aktuell den Projektfortschritt und Team-Prozess

vorantreiben.

- **Piratenschiff:** Das Piratenschiff ist in der Lage das Segelboot zum Kentern zu bringen und steht für Risiken, die das Projekt im weiteren Projektverlauf zum Scheitern bringen können.
- **Felsen:** Die Felsen können das Segelschiff versenken und stehen ebenfalls für Risiken, die das Projekt im weiteren Projektverlauf zum Scheitern bringen können.

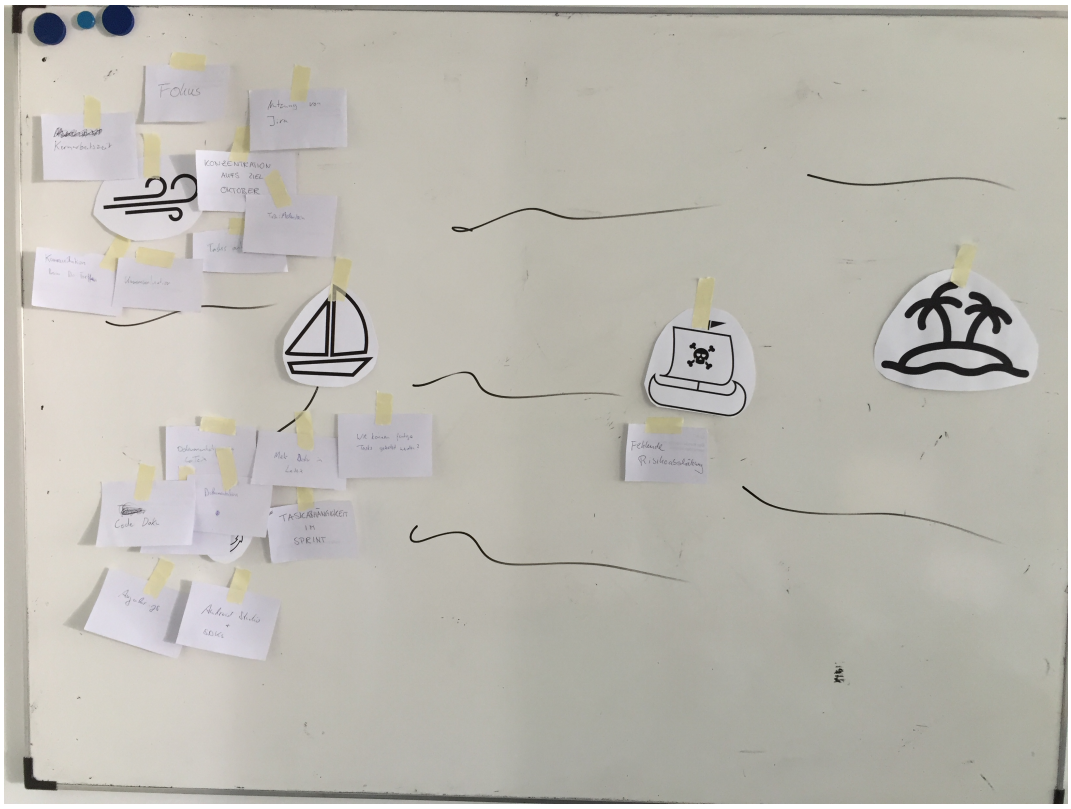


Abbildung 3: Beispiel für die Sailboat-Technik (Bild der Ergebnisse des dritten Sprints)

Der Scrum-Master stellt dem Team nun folgende Fragen:

- Was sind günstige **Winde**, die uns helfen das Segelboot zur Insel zu treiben?
- Welche **Anker** halten uns zurück?

- Welche **Felsen** oder **Piraten** können unser Schiff versenken (oder zumindest beschädigen)?

Die Teammitglieder haben mehrere Minuten Zeit um die Fragen auf kleinen Zetteln zu beantworten. Anschließend werden die Zettel an die Tafel zu den entsprechenden Metapher geklebt. Abschließend werden Zettel mit ähnlichen Aussagen gruppiert.

Mad Glad Sad Bei der „Mad Glad Sad"-Methode wird eine Tafel, ein Whiteboard o. ä. in drei gleich Große Bereiche unterteilt. Überschrieben werden die einzelnen Bereiche mit den Begriffen *Mad*, *Glad* und *Sad*. Anschließend erklärt der Scrum-Master die einzelnen Überschriften. Diese haben die folgenden Bedeutungen:

- **Mad** – Frustrationen, Dinge die das Team genervt haben oder viel Zeit verschwendet haben.
- **Sad** – Enttäuschungen und Dinge, die nicht so gut wie erhofft funktioniert haben.
- **Glad** – Freuden und Dinge, die das Team glücklich gemacht haben.

Anschließend haben die Teammitglieder einige Minuten Zeit ihre Ideen zu den Überschriften auf kleinen Zetteln zu formulieren. Im Anschluss daran kleben die Mitglieder ihre Ideen in die Spalte der jeweiligen Überschrift an der Tafel. Danach werden zusammengehörende Zettel gruppiert und die wesentlichen Ideen vom Scrum-Master nochmal angesprochen. Ein Beispiel für diese Technik ist in Abbildung 4 dargestellt.

Team Craft Für die Team Craft Technik wird eine Rakete auf eine Tafel gemalt. Links und rechts neben die Rakete werden jeweils ein Pfeil nach oben und ein Pfeil nach unten gezeichnet. Beschriftet werden die zwei Pfeile nach oben mit *Enablers* und *Driving Forces*, die zwei Pfeile nach unten werden mit *Blockers* sowie *Impediments* beschriftet. Mit Hilfe dieser Analogie versucht das Team *Driving Forces und Enablers* zu identifizieren, die dem Projekt helfen das Ziel zu erreichen. Genauso sollen allerdings auch *Blockers und Impediments* identifiziert werden, die das Projekt gefähren bzw. den Fortschritt verzögern. Der Scrum-Master gibt den

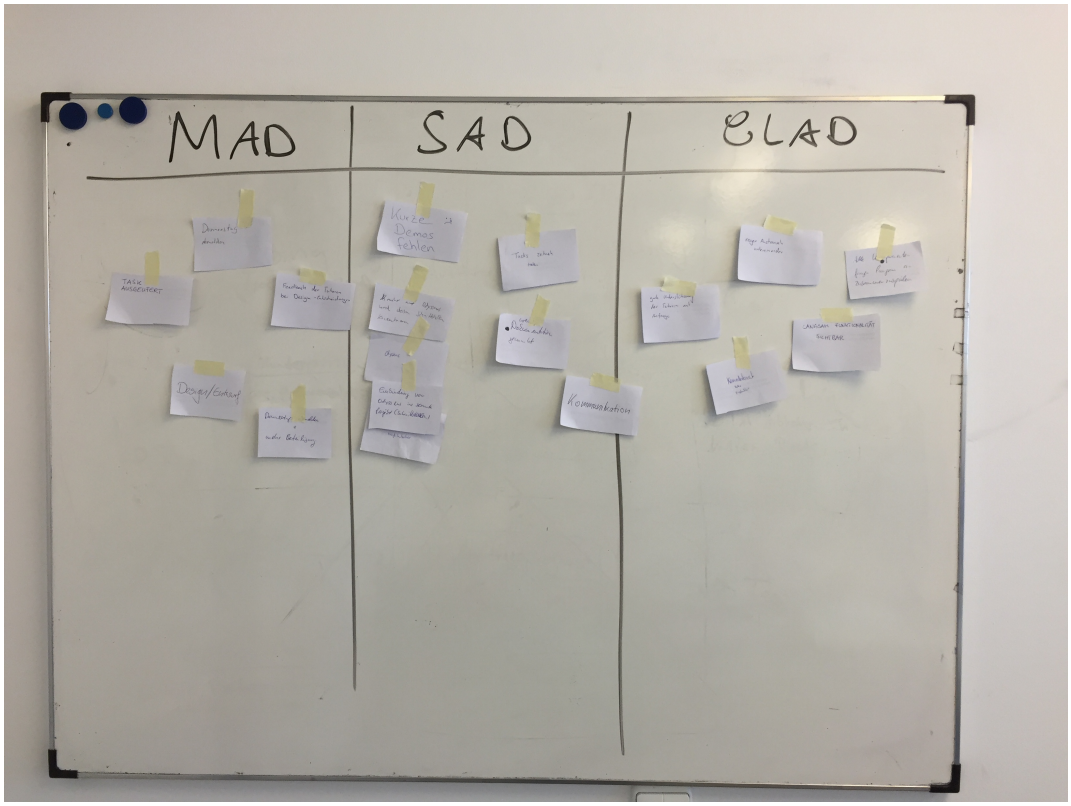


Abbildung 4: Beispiel für die „Mad Glad Sad“-Technik (Bild der Ergebnisse des siebten Sprints)

einzelnen Teammitglieder kleine Zettel, auf denen diese ihre Ideen notieren um sie anschließend an der Tafel zu sammeln. Ein Beispiel wie die Umsetzung dieser Technik aussehen könnte, lässt sich Abbildung 5 entnehmen.

Focus! Bei dieser Retrospektivetechnik geht es vor allem darum, dem Team zu helfen sich auf die relevanten nächsten Schritte im Projekt zu fokussieren bzw. diese zu identifizieren. Dazu werden zunächst drei Bereiche an die Tafel gemalt und kreisförmig angeordnet. Diese Bereiche erhalten anschließend die folgenden Überschriften: *Like*, *Dislike* und *Focus*. Der Scrum-Master erklärt die Bedeutungen der einzelnen Überschriten. Dabei handelt es sich um folgende Bedeutungen:

- **Like** – Unter dieser Überschrift sollen die Teammitglieder Dinge notieren, die ihnen im letzten Sprint gefallen haben oder die positiv für sie und das Team waren.

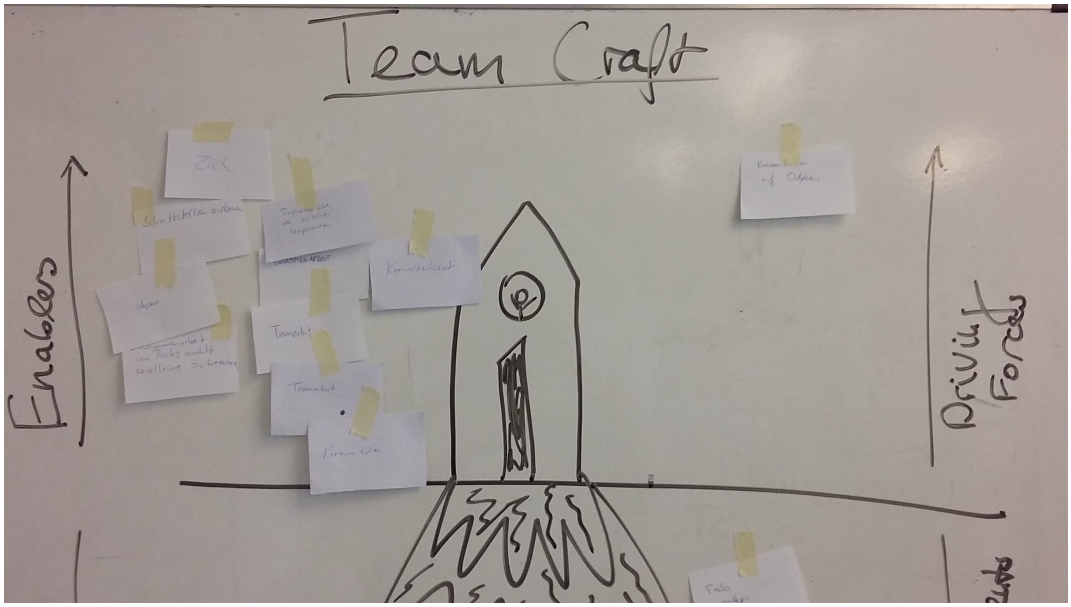


Abbildung 5: Beispiel für die „Team Craft“-Technik (Bild der Ergebnisse des neunten Sprints)

- **Dislike** – Unter dieser Überschrift sollen die Teammitglieder Dinge notieren, die ihnen im letzten Sprint nicht gefallen haben oder die negativ für sie und das Team waren.
- **Focus** – Hier sollen die Aufgaben, Features und ähnliche Dinge von den Teammitgliedern genannt werden, die sie in näherer Zukunft für besonders relevant erachten.

Nach der Erläuterung der Überschriften haben die Teammitglieder einige Minuten Zeit ihre Ideen zu den Begriffen auf kleinen Zetteln zu formulieren. Anschließend heftet jedes Teammitglieder seine Zetteln an die dafür vorgesehene Stelle an der Tafel und erläutert dabei kurz seine Idee. Zum Abschluss greift der Scrum-Master die wesentlichen Punkte auf und bespricht diese mit dem Team. Anhand des Bereichs *Focus* können der Scrum-Master und das erkennen auf welche Aufgaben der Fokus des Teams in den nächsten Sprints gelegt werden sollte. Ein Beispiel für diese Technik lässt sich Abbildung 6 entnehmen. Hierbei handelt es sich gleichzeitig um die Ergebnisse der Retrospektive des elften Sprints unserer Projektgruppe.

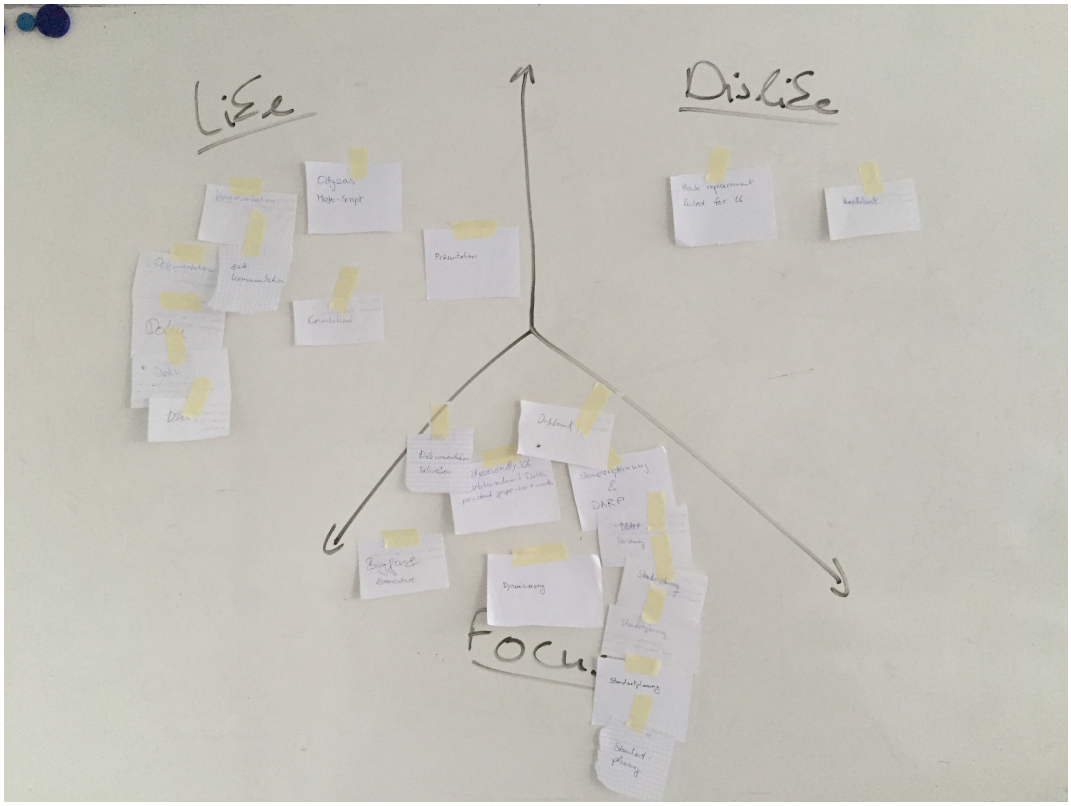


Abbildung 6: Beispiel für die „Focus“-Technik (Bild der Ergebnisse des elften Sprints)

2.2.6 Abschließende Retrospektive

In der letzten Sitzung der Projektgruppe wurde eine abschließende Retrospektive in Form einer offenen Diskussion durchgeführt. Teilnehmer waren neben den Gruppenmitgliedern auch die Tutoren. Folgende Punkte wurden dabei angesprochen:

- Es sollte mehr Wert auf die Kernarbeitszeiten gelegt werden. Teilweise wurde Abwesenheit bemängelt. Grundsätzlich war das Feedback hier jedoch positiv.
- Bei zukünftigen Projekten sollte stets darauf geachtet werden, dass immer ein auslieferbares Produkt zur Verfügung steht. Dies war aufgrund zahlreicher Bugs im Projekt nicht immer gegeben.
- Die Einarbeitung in Odysseus gestaltete sich anfangs schwierig. Eine Lösung der Problematik wurde nicht erarbeitet. Die Tutoren wurden aber involviert

und über die einzelnen Probleme in Kenntnis gesetzt. Hilfreich wäre laut der Gruppe eine bessere Dokumentation und eventuell die Vorstellung der Odysseus-Folien aus der Veranstaltung Informationssysteme 3.

- Es wurde angemerkt, dass Elemente eines automatisierten Entwicklungsprozesse durchaus sinnvoll sein können. So zum Beispiel könnte ein Test-Server dazu dienen die Testabdeckung des Codes darzustellen.
- Die Art und Weise wie sich die Tutoren eingebracht haben, wurde als gut empfunden. Auch wenn mehr Zurückhaltung häufig empfohlen wird, fühlte sich die Gruppe gut geleitet.
- Generell müssten Retrospektiven mehr aufarbeitet werden, damit die gezogenen Schlüsse auch zur Besserung führen. Eine Aufnahme in die Tagesordnung könnte hier helfen.
- Sitzungen und Live-Demos hätten besser vorbereitet werden sollen.

2.3 Testkonzept

Der *IEEE 829 Standard for Software Test Documentation* gibt Empfehlungen für die Inhalte und Dokumentation eines Testkonzeptes. Folgende Auswahl, die auf die PG angepasst wurde, sollte in einem Testkonzept enthalten sein:

- Vorgehensweise
- Ablaufplan
- Regeln zur Auswahl der Testfälle
- Testumgebung
- Dokumentation

2.3.1 Vorgehensweise

Im laufenden Sprint werden Testfälle für neue Features entworfen und die entsprechenden Tasks dafür angelegt. Im darauf folgenden Sprint sollen nun die entworfenen Testfälle implementiert und in der Testfallmappe dokumentiert werden.

Erfolgreich durchgeführte Testfälle werden anschließend automatisiert und in eine TestSuite übertragen. Nicht erfolgreiche Testfälle sollen möglichst noch im laufenden Sprint korrigiert werden. Die Einhaltung dieses Verfahrens führt zu einem kontinuierlichen Ablauf, da vor jedem Sprint-Review Testfälle für den nächsten Sprint angelegt werden.

2.3.2 Qualitätsmerkmale

Zu den wichtigen Qualitätsmerkmalen gehören:

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Effizienz
- Änderbarkeit
- Übertragbarkeit

2.3.3 Testphasen

- Komponententests
- Integrationstests
- Abschlusstest

2.3.4 Dokumentation

Die Dokumentation in der Testfallmappe soll folgende Punkte enthalten: Testobjekt, Verantwortlichkeiten, getestetes Qualitätsmerkmal gemäß ISO 9126, Erfolgskriterien und Testabbruch-/Wideraufnahmekriterien.

2.3.5 Code-Reviews

Es sollen mehrere Code-Reviews in kleineren Gruppen während der Kernarbeitszeit durchgeführt werden und mindestens ein großes Code-Review mit der gesamten Gruppe. Dabei soll der Programmcode direkt korrigiert werden und die Verbesserungen dokumentiert werden.

2.3.6 Merkmale für das Code-Review

Beim Code-Review soll vor allem auf eine verständliche Namensgebung (Klassen, Variablen, Methoden, Funktionen), verständliche Kommentare, Schnittstellenspezifikationen und Einhaltung von anderen Konventionen (z. B. Reihenfolge der Methoden) geachtet werden. Die Ergebnsise des Code-Reviews sind zu dokumentieren.

2.3.7 Frameworks

Verwendert wird u. a. das JUnit Testframework zur Implementierung und Ausführung automatisierter Einzeltests.

2.3.8 Konventionen

Bei Testklassen soll der Postfix **Test** (z. B. UpdateGraphTest) und bei Testmethoden der Präfix **test** (z. B. testSetSpeed) angefügt werden.

2.4 Verwendete Software

2.4.1 Odysseus

Das Oldenburger DynaQuest Datastream Query System (Odysseus) ist 2007 aus einem Forschungsprojekt der Abteilung Informationssysteme am Department für Informatik der Universität Oldenburg entstanden. Die Grundidee hinter Odysseus war die Entwicklung eines flexiblen Frameworks zur effizienten Verarbeitung von Datenströmen, das zunächst universell einsetzbar ist, jedoch bei Bedarf auch für unberücksichtigten Einsatzgebieten vollkommen angepasst werden kann. Als Datenstrommanagementsystem verfügt es dabei über einen weit größeren Umfang als Systeme die nur Ereignisdatenströme verarbeiten können (Complex Event Processing, CEP). Nach einer Einbindung von Datenstromquellen und -Senken, kann

über Anfragen eine Verarbeitung von eingehenden Daten verschiedener Typen definiert werden. Ermittelte Ergebnisse werden daraufhin fortlaufend automatische an die definierte Anwendung übermittelt. Für Anfragen können Sprachen wie CQL oder Procedural Query Language (PQL), die für den Umgang mit Datenströmen optimiert wurden, wie folgt verwendet werden.
Auch eine Integration neuer Operatoren ist möglich. Bei der Verarbeitung von Anfragen nimmt Odysseus eigenständig Optimierungen vor, wobei gleiche Verarbeitungsschritte verschiedener Anfragen zusammengefasst und so Ressourcen gespart werden können. Über die unterstützte Mehrbenutzerfähigkeit ist es ebenso möglich, einen Odysseus Server für mehrere verschiedener Anwendungen parallel zu verwenden. Aber auch eine Verteilung der Anfragen auf mehrere Instanzen im Netzwerk ist über ein nachinstallierbares Peer 2 Peer Plug-In möglich. Je nach Anwendungsgebiet können verschiedenste Features über Plug-Ins den Funktionsumfang von Odysseus sinnvoll erweitern oder bestehende Standardkomponenten ersetzten. So können bspw. Erweiterungen für zusätzliche Datentypen, Zugriffsprotokolle und komplexe Mustererkennung, aber auch Test- und Auswertungstools zur Stabilisierung der Systemlast geladen werden. Für die Verbindung mit Datenquellen und -Senken kann eine Auswahl an bereits implementierte Adapter-Schnittstellen verwendet werden. Eine Einbindung von Sozialen Netzwerken wie Twitter und Facebook wurde bereits implementiert und erweitert die Einsatzmöglichkeiten und Anwendungsgebiete des Systems. Werden neben der breiten Auswahl an Features bspw. noch weitere Protokolle oder Übertragungsmechanismen benötigt, um spezielle Sensoren zu unterstützen, können diese selbst in Java entwickelt und nachträglich eingebunden werden. Für die Steuerung und Ausführung von Odysseus wurde mit Odysseus Script eine eigene Sprache entwickelt. Ein Zugriff kann per Webservice, Konsole oder Java erfolgen.

2.4.2 Subversion

Subversion (SVN) ist eine Software zur Versionskontrolle. Die Nutzung von SVN als zentrales Projektarchiv (Repository) wurde der Projektgruppe vorgegeben, da nur diese Software von Odysseus unterstützt wird. SVN besitzt einige Besonderheiten, die es von anderen Versionskontrollen-Softwarelösungen unterscheidet. Bei Änderungen einzelner Dateien und dem anschließenden Commit dieser

Änderungen wird eine neue Version des vollständigen Projektarchivs statt nur der Dateien erstellt und dieser eine neue Revisionsnummer zugeordnet. Bei Commits, Checkouts und Updates speichert SVN eine Kopie der geänderten Dateien im .svn-Verzeichnis, wodurch Änderungen auch ohne Netzwerkzugriff eingesehen werden können. Commits geschehen in SVN atomar - sie werden entweder vollständig oder gar nicht ins Projektarchiv gespeichert.

2.4.3 GIT

Ebenso wie Subversion ist Git eine freie Software zur Versionsverwaltung von Dateien. Im Gegensatz zu einem zentralen Repository mit Änderungshistorie, das Subversion einsetzt, setzt Git auf ein verteiltes System. Dabei besitzt jeder Nutzer eine lokale Kopie des gesamten Repositories, inklusive der einzelnen Veränderungen. Commits (Festhalten von Veränderungen) können somit auch ohne Verbindung zum Server lokal hinterlegt und zu einem späteren Zeitpunkt synchronisiert werden. Ein Ausfall oder Verlust des zentralen Servers, der weiterhin zur Synchronisation genutzt wird, führt somit nicht zum Verlust der Änderungshistorie. Die Projektgruppe ist im laufenden Projekt von SVN auf Git umgestiegen, da die Veröffentlichung neuer Odysseus Releases unter SVN eingestellt wurde. Als Plattform wurde der Gruppe Bitbucket zur Verfügung gestellt. Die Veränderungshistorie konnte vom Beauftragten vollständig aus SVN übernommen werden. Im Workflow wurde nach mehreren Absprachen die Möglichkeit integriert, in getrennten Branches zu arbeiten. Dieses Vorgehen stellt sicher, dass zu jedem Zeitpunkt eine lauffähige Version unserer Software vorliegt.

2.4.4 GIT-Workflow

Um einen stabilen Code zu gewährleisten, hat die Projektgruppe sich bemüht mit einem GIT-Workflow zu entwickeln. Hierbei haben wir uns dazu entschieden den Feature-Branch-Workflow zu verfolgen. Dabei wurden die Features, in gekapselten Feature-Branches entwickelt. Oft wurden dabei mehrere Tasks in einem Feature-Branch bearbeitet. Wenn Bug-Tasks in *JIRA* erstellt wurden, wurden hierfür spezielle Bugfix-Branches erstellt. Die Abkapselung ermöglicht es mehreren Projektmitgliedern an bestimmten Tasks in einem Feature-Branch zu arbeiten, ohne die Haupt-Codebasis zu beeinflussen. Es wurde darauf geachtet, dass

die Feature-Branches nur in den Master-Branch gemergt wurden, wenn diese keine Fehler im Code aufgewiesen haben, damit der Master-Branch niemals fehlerhaften Code beinhaltete und somit zu jeder Zeit ein lauffähiges Produkt bereitstand. Hierzu konnten Pull-Requests wirksam eingesetzt werden, die die Möglichkeit geboten haben Branches durch weitere Projektmitglieder innerhalb eines CodeReviews abzusegnen, bevor Sie in das allgemeine Projekt integriert wurden.
Der Master-Branch repräsentiert die offizielle Projekt-Historie. Bei der Erstellung von Branches wurde angestrebt möglichst aussagekräftige Bezeichnungen für die Feature-Branches zu treffen, damit diese den Tasks oder Sprints eindeutig zugeordnet werden konnten und es klar wurde, welchen Zweck welcher Feature-Branch hatte.
Damit auch mehrere Projektmitglieder an einem Feature-Branch arbeiten konnten, konnten die Entwicklungsfortschritte aus dem lokalen Git-Repository der einzelnen Projektmitgliedern in ein zentrales Git-Repository gepusht werden. Das Speichern der Feature-Branches in einem zentralen Git-Repository beeinflussten den Master-Branch ebenso wenig, wie in einem lokalen Git-Repository. So wurde es uns gleichzeitig bequem ermöglicht Backups für unsere Commits im zentralen Repository zu erstellen. Das zentrale Git-Repository konnte mit BitBucket verwaltet werden. Innerhalb von Bitbucket wurden die Pull-Requests durchgeführt, um die Feature- bzw. Bugfix-Branches in den Master-Branch zu mergen. Bitbucket wurde mit unserem Projektmanagement-Tool *JIRA* synchronisiert, sodass Task-Angaben in unseren Commit-Messages mit Tasks aus *JIRA* verknüpft wurden und wir somit möglichst jeden Commit einem Task zuweisen konnten.
Die lokale Verwaltung der lokalen GIT-Repositorys wurden von den Mitglieder mit mehreren Tools durchgeführt. Hierzu gehörten:

- SourceTree von Atlassian
- Über die Konsole
- EGIT das standard geliefert Eclipse-Plugin

2.4.5 Confluence

Confluence ist eine Java-basierte Plattform zur Kommunikation sowie zum Austausch von Daten und Wissen innerhalb der Projektgruppe. Ein elementarer Be-

standteil von Confluence ist das Dashboard. Innerhalb eines Arbeitsbereichs dient es der Navigation in der von der Projektgruppe angelegten Seitenhierarchie und dem Schnellzugriff auf Verknüpfungen. Weiterhin bietet Confluence unterschiedliche Werkzeuge wie die Suche, Änderungsanzeige und Favoritenangabe. Bei der Erstellung von Seiten wird nicht nur die Eingabe von Text und dessen Formatierung ermöglicht, sondern auch die Einbindung von Listen, Bildern und Dateien wie PDF-Dokumente oder PowerPoint-Präsentationen. Weitere Features von Confluence sind z. B. die Einbindung verschiedener Schnittstellen wie die Makroerstellung sowie eine Drag-and-Drop-Funktionalität.

2.4.6 JIRA

JIRA ist eine Java-basierte Projektmanagementsoftware für die Softwareentwicklung. Es bietet den Nutzern unter anderem Werkzeuge für die Fehlerbehebung, das Anforderungsmanagement und die Statusverfolgung. Wie in Confluence ist auch in JIRA ein Dashboard ein elementarer Bestandteil. Auf diesem Dachboard können Nutzer Verlinkungen und Auflistungen eines ausgewählten Arbeitsbereichs erreichen. Eine wesentliche Funktion von JIRA ist die Erstellung und Verwaltung von Tickets. Tickets besitzen unterschiedliche Informationen wie eine Beschreibung, ein Typ (z. B. Task oder User Story), eine Priorität, Kommentare und Details über den Ersteller des Tickets. Weiterhin können Anhänge zum Ticket und selbst definierte Felder zum Ticket hinzugefügt werden. Unterschiedliche Informationen sowie der Status der Tickets können bearbeitet bzw. geändert werden. Eine weitere Funktion von JIRA ist die Unterstützung von agilen Vorgehensmodellen wie SCRUM. So kann das Projekt mit Hilfe von Sprints und Epen geplant, strukturiert und verwaltet werden. In einem Backlog werden alle für das Projekt erstellten User Stories und Tasks aufgelistet. In der Ansicht Äktive Sprints"werden alle dem aktuellen Sprint zugewiesenen Tasks nach Status aufgeführt und können per Drag&Drop von Nutzern zu einem anderen Status verschoben werden. In der Ansicht "Berichte"können Informationen zum Projekt wie z. B. Burndown-Diagramme angezeigt werden.

2.4.7 SUMO

Die Abkürzung Sumo steht für SSimulation of Urban MObilityünd ist ein kostenloses, quelloffenes Open Source Projekt, welches hauptsächlich vom Deutschen Zentrum für Luft- und Raumfahrt entwickelt wird. Es ist in C++ implementiert und verwendet ein mikroskopisches Fahrzeugfolgemodell, welches mit Hilfe von Sensormessdaten eines Autobahnabschnittes kalibriert wurde. Die Fahrzeuge werden raumkontinuierlich sowie zeitdiskret mit einem Zeitschritt von einer Sekunde simuliert. In der Verkehrssimulation werden aus soziodemographischen statistischen Daten der simulierten Welt Tagespläne ermittelt, die dann mit Hilfe von simulierten Agenten ausgeführt werden. In Sumo besteht die Möglichkeit multimodalen Verkehr zu modellieren, was bedeutet, dass verschiedene Verkehrsteilnehmer und Fahrzeugtypen simuliert werden können. Allerdings sind andere Verkehrsteilnehmertypen als Autos nur durch Schätzung der Reisedauer, also ohne die Simulation des zurückgelegten Weges implementiert. Somit kann eine Interaktion zwischen den verschiedenen Verkehrsteilnehmertypen nicht berücksichtigt werden. SUMO wird seit 2001 entwickelt und erweitert. So stellt es Funktionen zur Generierung von Graphen aus Kartenmaterial im Shapefile- und OpenStreetMap-Format (OSM) zur Verfügung. Außerdem stehen zum Beispiel Erweiterungen zur Verfuğung, die es ermöglichen die Simulation Online zu steuern oder die Anbindung an ein Simulations-Framework erlauben. Weiterhin ist es über eine Schnittstelle möglich den Status der Simulation jederzeit abzufragen, wozu auch die Position von einzelnen Verkehrsteilnehmern gehört.

2.4.8 TraCi

TraCi steht für "Traffic Control Interfaceünd ist eine Erweiterung, die es ermöglicht die Verkehrssimulation interaktiv zu steuern. Es kann als Schnittstelle verwendet werden, weil es zum einen möglich ist Befehle an die Verkehrssimulation zu senden und zum anderen können zum Beispiel Statusdaten der Fahrzeuge wie Position oder Route abgefragt werden. Es gibt verschiedene Versionen von TraCi. Die vom Deutschen Zentrum für Luft- und Raumfahrt entwickelte Variante ist in Python implementiert und unterstützt alle Befehle, welche nach eigenen Angaben täglich getestet werden. Eine andere Version ist Traci4J, welche eine Java Implementierung von TraCi ist. Diese Version wurde von Dritten geschrieben und unterstützt

nur einen Bruchteil der TraCi Funktionen. Eine weitere Version ist TraaS, welche ebenfalls eine Java Implementierung von TraCi ist. Im Gegensatz zu Traci4J wurde dabei der Code automatisch aus den Python Skripten generiert, weshalb theoretisch alle Funktionen zur Verfügung stehen.

2.4.9 GraphHopper

In unserem Projekt verwenden wir die von der GraphHopper GmbH entwickelte und frei verfügbare Routing-Engine als Java-Klassenbibliothek. Diese dient dazu eine Route zwischen zwei oder mehreren Punkten auf einer Straßenkarte zu ermitteln. Zunächst dient GraphHopper dazu die Kosten einer Strecke zwischen zwei Punkten für JSprit zu ermitteln. Es werden für alle Kanten des Graphen in JSprit die realen Kosten (Fahrzeit) der Route ermittelt und zurückgegeben. Hier muss beachtet werden, dass die Strecke zwischen Punkt A und B ungleich der Strecke zwischen Punkt B und A ist (Verkehrsführung durch Einbahnstraßen oder unterschiedliche Verkehrslage auf der jeweiligen Fahrbahnseite). Nachdem JSprit eine optimale Reihenfolge der Wegpunkte für eine Fahrzeug berechnet hat, wird diese wieder an GraphHopper übergeben und GraphHopper berechnet die tatsächliche Route auf der Straßenkarte und übergibt diese abschließend an die Verkehrssimulation. GraphHopper verwendet in unserem Projekt eine OSM-Karte von Oldenburg aus dem ein Graph erzeugt wurde. Der Graph besteht aus Straßen(-abschnitten), welche als Kanten mit Kreuzungen/Abbiegungen als Knoten verbunden sind.

2.4.10 JSprit

JSprit ist eine Java-Klassenbibliothek, mit der u. a. das VRP und - mit entsprechenden Anpassungen - auch das DARP gelöst werden kann. Die Aufträge werden als Wegpunkte übergeben und JSprit berechnet eine optimale Route zwischen diesen Wegpunkten, d.h. eine optimale Reihenfolge der Wegpunkte. Dieses geschieht indem JSprit die Wegpunkte als Punkte eines Graphens und die Kosten zwischen zwei Punkten als Kante behandelt. Mit einer Meta-Heuristik berechnet JSprit nun die optimale Route zwischen den einzelnen Punkten. Das Problem ist, dass JSprit die Punkte nur auf einem simplen Koordinatensystem darstellt und keinen Bezug zu einer Straßenkarte hat. Somit wird auch für die Kosten die einfache Entfernung zwischen den Punkten im Koordinatensystem verwendet. Um JSprit für unser

Projekt nutzen zu können, müssen wir die Kostenfunktion anpassen und mittels GraphHopper die reale Fahrzeit zwischen den einzelnen Punkten des Graphens ermitteln.

2.4.11 Firebase Cloud Messaging

Bei Firebase Cloud Messaging handelt es sich um eine cross-platfrom messaging Lösung die es ermöglicht Nachrichten und Benachrichtigungen für iOS, Android und Web Apps zu versenden. Das Versenden der Nachrichten führt dabei zu keinen Kosten. Firebase Cloud Messaging ist der Nachfolger von Google Cloud Messaging und wurde bzw. vom Unternehmen Google entwickelt. Bei Firebase Cloud Messaging wird zunächst eine Nachricht an den Google Server gesendet, der daraufhin entweder alle Apps oder nur die App auf einem bestimmten Gerät benachrichtigt. In unserem Projekt wird Firebase Cloud Messaging dazu benutzt die App des Kunden zu informieren sobald seine Route berechnet ist bzw. eine Routenaktualisierung verfügbar ist.

2.4.12 LaTeX

Für die Dokumentation dieses Projekts wird das Textverarbeitungsprogramm LaTeX verwendet. Zur Verwendung von LaTeX werden eine TeX-Distribution (z. B. MiKTEX) sowie ein TeX-Editor (z. B. TexMaker) benötigt. Ein Vorteil von LaTeX ist die Möglichkeit die Struktur des Dokuments in einzelne Dateien unterteilen zu können. Bei großen Dokumenten erleichtert dies die Übersicht und ermöglicht eine Arbeit von verschiedenen Nutzern an unterschiedlichen Abschnitten parallel ohne Versionskonflikte. Ein weiterer Vorteil ist die Möglichkeit nicht nur Bilder, sondern auch andere Dateien wie z. B. PDF-Dokumente in das Dokument einzubinden. LaTeX ermöglicht weiterhin das Erstellen von Grundgerüsten für Dokumentvorlagen, die universell eingesetzt werden können.

2.4.13 Jython

Jython ist eine Java-Implementierung von Python und ermöglicht die Ausführung von Python Code innerhalb von Java. Es wurde benutzt um das Python Skript zum ansprechen der SUMO Schnittstelle innerhalb des Java Projekts ausführen zu können.

2.5 Zusammenfassung

Im vorausgegangenen Kapitel wurde die Organisation beschrieben. Wichtige Bestandteile dieser sind die Aufgabenbereiche der Gruppenmitglieder. Zusätzlich entwickelten sich im Laufe des Projektes weitere Kernkompetenzen wie Entwicklung der App oder Entwicklung mit GraphHopper und JSprit.
Das Vorgehensmodell des Projektes ist Scrum. Wichtige Aktivitäten hierbei sind die Sprintplanung, die Sprintreviews und die Sprintretrospektiven.
Im Anschluss wurde die verwendete Software beschrieben. JIRA und Confluence dienen zur Unterstützung des Vorgehensmodells. Odysseus, SUMO, GraphHopper und JSprit sind die zentralen Komponenten des Systems. Odysseus dient zur Datenverarbeitung. SUMO simuliert den Verkehr. GraphHopper und JSprit werden zum Routing und zur Routenberechnung verwendet.

3 Grundlagen

Der folgende Abschnitt stellt die wesentlichen Themen vor, die für das weitere Verständnis der Dokumentation wichtig sind. Diese Grundlagen wurden in Seminararbeiten, die während des Projekts von einzelnen Teilnehmern angefertigt worden sind, erarbeitet.

3.1 Datenstrommanagementsysteme

Ein Datenstrommanagementsystem (DSMS, engl. data stream management system) ist für die Verarbeitung und Verwaltung von Datenströmen ausgelegt.
Ein Datenstrom (engl. Data Stream) bezeichnet einen kontinuierlichen Fluss an Echtzeit-Datensätzen eines beliebigen jedoch gleichbleibenden Typs, der eine unendliche Länge aufweisen kann [13]. Er kann durch eine über die Zeit variierende Datenrate qualifiziert werden, die den Datendurchsatz, also die Datensätze pro Zeiteinheit angibt. Datenströme können bspw. von Sensoren in autonomen Elektrofahrzeugen erzeugt werden, um einem zentralen Leitsystem aktiv und ununterbrochen Statusinformationen (aktuelle Position, Geschwindigkeit, Zielort, Reichweite) mitzuteilen. Ein einzelner Datensatz kann dabei u.a. als Tupel betrachtet werden, der ähnlich wie ein relationaler Datenbankeintrag aus einer Reihe von Attributen mit zugehörigen Werten besteht. Auch wenn relationale Modelle durch eine gut definierte Semantik weit verbreitet sind, unterstützen nicht alle Datenquellen dieses Konzept. Auch dynamische Graphen können über Datenströme abgebildet und analysiert werden. Dabei kann ein Datenstrom aus verschiedenen Bestandteilen, wie einer Sequenz von Kanten, einer Matrix oder Liste, die den Graphen angibt oder einer Aktualisierung der Kantenwerte bestehen [9].
Datenstrommanagementsysteme können mit Datenbankmanagementsystemen verglichen werden. Die besonderen Eigenschaften, denen Datenströme unterliegen, verwehren hingegen DBMS eine Verarbeitung. Wesentliche Unterschiede zwischen DBMS und DSMS sind somit die zugrundeliegende Datenbasis, aber auch die Art und Weise der gestellten Anfragen (Query) [13]. Ein DBMS ist auf die persistente Speicherung von sich selten verändernden Datensätzen und vielen, spontan gestellten und teilweise komplexen Benutzeranfragen ausgelegt. Dem gegenüber steht ein DSMS, das eine Vielzahl von eingehenden Datenströmen, mit unendlicher Länge, zeitnah verarbeiten kann, diese aber nicht speichern muss [5].

3.1.1 Architektur von DSMS

Beim Stellen einer Anfrage wird zunächst der logische Anfrageplan optimiert und zu einem physischen Ausführungsplan überführt (Physical Query Execution Plan, QEP). Logische Operatoren enthalten gegenüber physischen Operatoren keine konkrete Implementierung, sondern beschreiben den Umgang mit den Daten, ähnlich wie ein algebraischer Operator. So kann ein logischer Operator durch verschiedene physische Operatoren umgesetzt werden[5]. Ein QEP enthält die Aufrufe der implementierten Operatoren sowie die gepufferten Schlangen (Queues) für Operator Ein- und Ausgaben. In QEP enthaltene Synopsis Strukturen stellen Algorithmen und Datenstrukturen bereit, die eine kurzfristige Speicherung von Zwischenständen ermöglicht, indem ein Datenstrom oder ein Teil eines Datenstroms zusammengefasst wird. Bei zu hoher Systemauslastung können bspw. Load Shedder Elemente unberücksichtigt lassen. Für weitere Optimierungszwecke werden häufig QEPs zu einem großen Plan zusammengefasst, um bereits für anderen Anfragen errechnete Zwischenschritte wiederzuverwenden [9].

3.2 Verkehrssimulationssysteme

Bei der Analyse von Sachverhalten kommt es oft zu einem nicht vertretbaren Aufwand in der Realität. Dieser entsteht beispielsweise, wenn das Realsystem noch nicht existiert, Experimente am System zu teuer oder gefährlich wären oder es dadurch sogar zerstört werden würde [14]. Deshalb werden Simulationen verwendet, welche auf die Wirklichkeit übertragbare Erkenntnisse liefern sollen [4]. In diesen Simulationen wird das Realsystem durch ein Abbild, welches auch Modell genannt wird, dargestellt. Diese Modelle sind Vereinfachungen der Realität, die zwar die Komplexität reduzieren, aber dennoch genügend Informationen beinhalten, um damit Rückschlüsse auf die Realwelt ziehen zu können [14].
Eine spezielle Art einer Computersimulation ist eine agentenbasierte Simulation, die häufig in Verkehrssimulationen zum Einsatz kommt. Ein System kann eine große Anzahl von verschiedenen Subsystemen enthalten, die miteinander interagieren und sich gegenseitig beeinflussen. Um dabei ein Gesamtverhalten vorherzusagen ist es notwendig, jedes einzelne Subsystem zu simulieren. Dies kann mit

[5] `http://wiki.odysseus.informatik.unioldenburg.de/display/ODYSSEUS/The+Odysseus+Operator+Framework,01.05.16`

Hilfe von agentenbasierten Simulationen umgesetzt werden [4]. Ein Agent ist ein Computersystem, das sich in einer Umgebung befindet und die Fähigkeit besitzt, autonom in dieser Umgebung Aktionen durchzuführen, um vorher definierte Ziele zu erreichen [17]. Die in der Simulation enthaltenen Subsysteme werden dabei als Agenten eingesetzt, wobei eine Simulation mit einer Vielzahl von Agenten auch Multiagentensystem bezeichnet wird. Ein Beispiel für ein Multiagentensystem ist der Straßenverkehr, in dem jeder Verkehrsteilnehmer als Agent modelliert wird, der mit anderen Agenten interagiert. Insgesamt entsteht ein komplexes Zusammenspiel, das sogenannte Emergenzen entstehen lässt. Ein negatives Beispiel für eine Emergenz ist ein Stau. Allerdings gibt es auch positive Beispiele, wie zum Beispiel die Findung der kürzesten Route bei den Ameisen, die zur Entwicklung des sogenannten Ameisenalgorithmus geführt hat [4].
Für die Simulation von Verkehr wurden im letzten Jahrhundert sogenannte Verkehrssimulationssysteme entwickelt. Dies begann bereits in den 1930er Jahren, weil Prognosemethoden für Verkehrsaufkommen sowie Modelle zur Optimierung der Verkehrsleittechnik verlangt wurden. Aufgrund der geringen Leistung von Computersystemen sind diese Simulationen makroskopisch ausgelegt, das heißt sie berechnen das Verhalten einer Gruppe. Erst in den letzten beiden Jahrzehnten ist es zur Entwicklung von mikroskopischen Simulationen gekommen [4].

3.2.1 Makroskopische und mikroskopische Simulationen

Bei makroskopischen Verkehrssimulationen werden Kenngrößen wie der Verkehrsfluss oder die Verkehrsdichte betrachtet. Bei diesem Verkehrsmodell ist es beispielsweise nicht relevant die Geschwindigkeit von jedem simulierten Individuum zu berechnen. Deshalb wird nicht jeder einzelne Verkehrsteilnehmer simuliert, sondern es wird angenommen, dass sich alle Teilnehmer in der gleichen Durchschnittsgeschwindigkeit fortbewegen. Auch die Routenplanung wird nicht detailliert betrachtet, weil Details wie einzelne Spuren vernachlässigt werden. Wichtige Kenngrößen sind die Fahrzeugkapazität der Straße und wie schnell sich die Fahrzeuge durchschnittlich bewegen [2].
Das mikroskopische Modell erlaubt die Simulation von einzelnen Verkehrsteilnehmern mit detaillierten Eigenschaften wie zum Beispiel der Geschwindigkeit, Größe und Beschleunigung. Auch die Routenplanung kann detailliert dargestellt wer-

den, indem Spuren, Ampeln und Kreisverkehre simuliert werden. Durch die eingeschränkte Leistungsfähigkeit der Computer wurden mikroskopische Verkehrsmodelle vor allem erst ab den 1990er Jahren entwickelt und auch heute noch haben sie Grenzen. Beispielsweise ist die simulierte Welt deutlich kleiner als bei makroskopischen Verkehrssimulationen [2].
Bei einigen Simulationssoftwaresystemen verschwimmt die Grenze zwischen makroskopischen und mikroskopischen Verkehrsmodellen, weil Eigenschaften aus beiden Verkehrsmodellen implementiert sind (Mesoskopische Simulationen). Der Verkehr kann beispielsweise in einer Stadt mikroskopisch untersucht werden und der Zu- und Abfluss aus angrenzenden Regionen wird makroskopisch simuliert. In einer Stadt gibt es zudem eine Reihe von unterschiedlichen Straßentypen. Auf Fernstraßen wie zum Beispiel einer Autobahn kommt in der Regel ein makroskopisches Modell zum Einsatz, während im Straßenverkehr mit komplexen Kreuzungen und Ampelschaltungen ein mikroskopisches Modell eingesetzt wird [4].

3.2.2 Fahrzeugfolgemodelle

Bei diesen Modellen wird das Verhalten der Fahrzeuge in Relation zu den vorausfahrenden Fahrzeugen in ihrer Spur simuliert. Das Ziel besteht darin, die Verhaltensweise der Fahrzeuge so realistisch wie möglich abzubilden und beispielsweise sprunghafte Geschwindigkeitsveränderungen zu vermeiden [4]. Dabei sollte der Mindestabstand zum vorausfahrenden Fahrzeug proportional zur Geschwindigkeit sein, was bedeutet, dass die Zeitlücke zwischen den beiden Fahrzeugen einem vorgegebenen Zielwert entsprechen muss. Abhängig von den Aktionsmöglichkeiten der Fahrzeuge wird zwischen sogenannten Modellen der Längsdynamik, bei denen Beschleunigung und Bremsverhalten modelliert werden, und Modellen der Querdynamik, bei denen Spurwechsel modelliert werden, unterschieden. In komplexen Verkehrssimulationen werden die Eigenschaften beider Modelle verknüpft um eine möglichst realistische Simulation zu ermöglichen [15].

3.3 Dial-a-Ride Problem

Das Dial-a-Ride Problem (kurz: DARP) tritt unter anderem bei Fahrdiensten für den Personentransport auf. Der Transport erfolgt mit einem Fahrzeug oder eine Fahrzeugflotte. Die Transportleistung wird als Dienstleistung für mehrere Kunden

angeboten, die sich (temporär) ein Fahrzeug teilen. Die Kunden bestimmen dafür ihren Abholpunkt, den Ablieferpunkt, sowie Zeitfenster für die Abholung und Ablieferung. Es kann auch vorkommen, dass Kunden entweder nur das Zeitfenster für die Abholung oder die Ablieferung vorgeben. Diese Daten werden von den Kunden als Anfrage an den Fahrdienst übermittelt. Der Fahrdienst verfügt über ein Fahrzeug oder eine Flotte von Fahrzeugen, um die Kundenanfragen zu bedienen. Zur Bedienung der Kundenanfrage schickt der Fahrdienst ein Fahrzeug aus dem eigenen Depot zum Abholpunkt des Kunden und fährt den Kunden zu seinem Ablieferpunkt. Das Fahrzeug muss dabei innerhalb des vorgegebenen Zeitfensters am Abhol- bzw. Ablieferpunkt eintreffen und nach der Fahrt wieder in das Depot zurückkehren. Zudem kann der Fahrdienst prüfen, ob die Bündelung von Kundenanfragen möglich ist, um Fahrten verschiedener Kunden zusammenzulegen [3]. Das DARP enthält Restriktionen bezüglich der Zeitfenster an den Abhol- bzw. Ablieferpunkten und der Fahrzeiten. Damit ist es eine verallgemeinerte Form des Vehicle routing problems with pickups and deliveries (kurz: VRPPD) und gehört damit zu den schweren Planungsproblemen. Beim VRPPD wird nach optimalen Routen gesucht, auf denen Fahrzeuge aus dem Depot heraus alle Wegpunkte (Kundenanfragen) der Reihe nach durchlaufen und anschließend wieder in das Depot zurückkehren, ohne die Beladerestriktionen eines Fahrzeuges zu verletzen. Es wird zwischen Abhol- und Ablieferpunkten unterschieden. Abholpunkte repräsentieren Kunden, die eine Lieferung vom Abholpunkt zum Depot erwarten, während Ablieferpunkte Lieferungen aus dem Depot erwarten. In der Praxis tritt ein solches Planungsproblem unter anderem bei Paketzustelldienstleistern auf [8].

3.3.1 Varianten

Das Dial-a-Ride Problem kann in unterschiedlichen Ausprägungen vorliegen. So kann zwischen einem homogenen beziehungsweise heterogenen Fuhrpark unterschieden werden. Ein homogener Fuhrpark besteht nur aus gleichen Fahrzeugen. Ein heterogener Fuhrpark besteht nicht nur aus gleichen Fahrzeugen. Ein homogener Fuhrpark eines Fahrdienstes bestünde aus nur einem Fahrzeugmodell, wie beispielsweise einem fünftürigen Fahrzeug für vier Fahrgäste, mit dem alle Kundenanfragen bedient werden [16].
Des Weiteren wird zwischen einer statischen und dynamischen Umgebung un-

terschieden. Beim statischen Dial-a-Ride Problem sind die Kundenanfragen im Vorfeld bekannt. Die Routenberechnung erfolgt anhand dieser Daten und wird anschließend nicht mehr verändert. Bei einer dynamischen Umgebung wird von einem Dynamic Dial-a-Ride Problem (DDARP) gesprochen.

3.4 Dynamic Dial-a-Ride Problem

Beim DDARP werden die Kundenanfragen während der Laufzeit in bestehende Routen eingeplant. Anfänglich berechnete Routen können so aufgrund von neuen Ereignissen verändert werden. Dies ist für eine intelligente Planung von Routen für Fahrdienste die vielversprechendere Herangehensweise, da die reale Umgebung eines Fahrdienstes ebenfalls dynamisch ist [16].
Das Ziel besteht daraus die optimale Route zu finden, damit kosteneffizient so viele Personen wie möglich mit möglichst wenig Fahrzeugen befördert werden. Eine wichtige Bedingung dabei ist die Dienstleistungsqualität bzw. das Zufriedenstellen der beförderten Personen [16]. Es muss also ein Gleichgewicht zwischen Kosteneffizienz und Kundenzufriedenheit gefunden werden, damit die Kunden keine unangemessene Zeitspanne bis zur Beförderung zu ihrem Ziel warten müssen [3].

3.5 Spatio-temporal Prediction

Bei der Spatio-temporal Prediction wird die Frage untersucht, zu welchem Zeitpunkt an welchem Ort ein Ereignis geschieht. Einen seiner Ursprünge hat die Spatio-temporal Prediction in der Bekämpfung von Verbrechen. So wurden z. B. schon Mitte des 19. Jahrhunderts in Frankreich statistische Daten erhoben, um Regionen mit erhöhten Verbrechensraten zu ermitteln. Jedoch wurde dies zu der damaligen Zeit noch manuell und ohne Zeitbezug getan. In den 1970ern wurden dann in den USA frühe Versuche unternommen, Verbrechen auf Karten mit Hilfe von digitalen Prozessen abzubilden. Dies war jedoch durch die technologischen Möglichkeiten stark eingeschränkt. So gab es Einschränkungen bei den Daten, digitale Adressen konnten nicht einfach als Punkte auf einer Karte umgewandelt werden und viele Polizei-Datenbanken waren einfach unorganisiert. Mit der technologischen Weiterentwicklung sind auch die Einschränkungen verschwunden und durch wachsendes Interesse an der Verbrechenskartographie ist ein ganzes Forschungsfeld der Verbrechensvorhersage entstanden.

In der Arbeit von Moreira-Matias et al. wird die Vorhersage der örtlichen Verteilung von möglichen Taxi-Passagieren für einen kurzfristigen Zeitraum unter Verwendung von Datenströmen in der Stadt Porto, Portugal untersucht. Heutzutage sind die meisten Taxis mit Technologien wie z. B. GPS ausgestattet, die eine neue Quelle zur Lieferung einer Menge von spatio-temporalen Daten sind. Das aus der Auswertung dieser Daten gewonnene Wissen, wo ein Bedarf für Taxis besteht, kann ein Vorteil für den Fahrer sowie das Unternehmen sein und letztendlich zur Erhöhung des Profits beitragen. Eine intelligente Verteilung von Fahrzeugen auf verschiedene Taxi-Stände reduziert die Wartezeit für die Fahrzeuge und macht die Anfahrt wesentlich profitabler. Die Auswahl des Taxi-Stands hängt von vier Variablen ab: 1) Der erwartete Erlös für eine Fahrt in einer bestimmten Zeit; 2) die Entfernung/Kosten-Beziehung für jeden Stand; 3) die Anzahl von Taxis, die bereits warten und 4) die Passagier-Nachfrage über die Zeit hinweg für den jeweiligen Stand [7].

3.6 Elektromobilität

Von vielen Menschen und Institutionen wird die Elektromobilität als die Zukunftstechnologie im Bereich der Mobilität angesehen. Die Elektromobilität sei der einzige Weg, unsere aktuelle Mobilität langfristig aufrechtzuerhalten, nachdem die fossilen Ressourcen unseres Planeten erschöpft sind. Außerdem können Elektrofahrzeuge einen wichtigen Beitrag zum Umweltschutz und zum Aufhalten des Klimawandels leisten, da sie – zumindest wenn die Energie aus regenerativen Quellen gewonnen wird - emissionsfrei sind. Gerade für Fahrzeuge, die dauerhaft auf Kurzstrecken im Einsatz sind, wie z. B. Taxis oder Carsharingfahrzeuge, wird Elektromobilität zunehmend interessanter [11].
Es werden verschiedene Definitionen der Elektromobilität verwendet, bei denen jedoch überwiegend ein gemeinsamer Konsens gefunden wird. In der Regel beinhalten diese Definitionen alle Typen von Fahrzeugen für den Personen- und Güterverkehr, "die mit elektrischer Energie angetrieben werden" [12]. Dabei ist nicht einheitlich festgelegt, welche Fahrzeuge als Elektrofahrzeuge bezeichnet werden, aus welchem Energiespeicher die elektrische Energie kommt oder zu welchem Anteil die vom Fahrzeug genutzte Energie elektrisch sein muss [1]. Die Bundesregierung definiert die Elektromobilität folgendermaßen:

"Elektromobilität [...] umfasst all jene Fahrzeuge, die von einem Elektromotor angetrieben werden und ihre Energie überwiegend aus dem Stromnetz beziehen, also extern aufladbar sind. Dazu gehören rein elektrisch betriebene Fahrzeuge (BEV), eine Kombination von E-Motor und kleinem Verbrennungsmotor (Range Extender, REEV) und am Stromnetz aufladbare Hybridfahrzeuge (PHEV)."

3.6.1 Ladestrategien

Bei den folgenden zwei Strategien, der Langsamladung und der Schnellladung, handelt es sich streng genommen um zwei Lademodi für Elektrofahrzeuge. Da diese jedoch entscheidende strategische Bedeutung für die Nutzung von Elektrofahrzeugen besitzen, werden sie im Rahmen dieser Projektgruppe auch als Strategien angesehen.

Langsamladung: Die Langsamladung erfolgt durch Wechselstrom (AC) und kann theoretisch an jeder Haushaltssteckdose durchgeführt werden. Da die üblichen Haushaltssteckdosen nicht auf das Erbringen der zum Laden benötigten Leistungen über einen langen Zeitraum ausgelegt sind, empfehlen Automobilhersteller die Installation und das Laden des Fahrzeugs durch eine sogenannte Wallbox.

Schnellladung: Die Schnellladung kann durch Wechsel- (AC) oder Gleichstrom (DC) erfolgen. Bei der Schnellladung des Akkus mit Wechselstrom kann das im Auto verbaute Ladegerät benutzt werden und es sind dabei Ladezeiten von unter einer Stunde für eine Vollladung zu realisieren [12]. In der Praxis ist jedoch das Laden mit Gleichstrom weiter verbreitet, da sich hiermit die Ladedauer auf unter 20 Minuten verkürzen lässt.

Wechselakku: Bei dieser Strategie sollen die langen Ladezeiten des Fahrzeugs verhindert werden, indem der Akku nicht mehr im Fahrzeug, sondern außerhalb und unabhängig davon geladen wird. Statt zu laden, fährt das Fahrzeug in eine speziell zu diesem Zweck konzipierte Akku-Wechselstation. Dort wird der komplette leere Akku gegen einen anderen vollen Akku getauscht. Anschließend wird der leere Akkumulator dann in der Station geladen und in ein anderes Auto eingesetzt [12]. Der Akkutausch beansprucht ca. fünf Minuten. Durch diese Strategie ist

es möglich, das Elektrofahrzeug in einer ähnlichen Zeit betriebsbereit zu machen, wie durch das Betanken eines konventionellen PKW mit Verbrennungsmotor.

Intelligentes Laden (Vehicle to Grid): Durch moderne Informationstechnologie wird es möglich, nicht mehr nur den Startzeitpunkt für das Laden des Elektrofahrzeugs festzulegen, sondern auch den Zeitpunkt zu dem der Akku des Fahrzeugs vollgeladen sein muss. Der Energieanbieter kann den Ladevorgang optimieren, indem er ihn (bei genügend Zeitreserven) so steuert, dass vorrangig dann geladen wird, wenn ausreichend oder zu viel Strom im Netz angeboten wird und dieser daher kostengünstig ist [12]. Um das Netz zu entlasten könnte das Laden des Fahrzeugs bei Spitzenlast unterbrochen werden oder der Akku als Energiespeicher dienen und bei Bedarf Energie zurück ins Netz speisen.

Induktives Laden: Beim induktiven Laden wird das Elektrofahrzeug kontaktlos geladen, d. h. es ist keinerlei Kabel als Verbindung zwischen dem Fahrzeug und dem Ladegerät notwendig. Das induktive Laden bietet dem Nutzer mehr Komfort als andere Ladearten, da dieser keine Kabelverbindung mehr herstellen bzw. das Fahrzeug überhaupt nicht mehr verlassen muss. Für den Ladevorgang sind im Prinzip zwei Spulen notwendig, eine davon im Fahrzeug und die andere im Ladepunkt - beispielsweise eine Garage oder ein Parkplatz. Das Laden funktioniert durch ein magnetisches Wechselfeld. Aussagen über Ladezeiten usw. lassen sich derzeit noch nicht treffen.

Wasserstoff tanken (Brennstoffzelle): Diese Ladestrategie ist nur für Brennstoffzellenfahrzeuge (FCHEV) möglich. Diese erzeugen aus Wasserstoff elektrische Energie durch Elektrolyse mittels einer Brennstoffzelle. Der dafür benötigte Wasserstoff wird in hochkomprimierter Form in einem Hockdrucktank mitgeführt. Das Betanken eines Brennstoffzellenfahrzeugs dauert ähnlich lange wie das Betanken eines Benzin- oder Dieselfahrzeugs

Einsammeln der Fahrzeuge mit einer Zugmaschine: Diese Ladestrategie ist auf die Nutzung beim Carsharing mit Kleinwagen optimiert. Sie soll es dem Kunden ermöglichen, das von ihm gebrauchte Elektrofahrzeug an jeder beliebigen Stelle abzustellen. Die Fahrzeuge müssen nicht an einem bestimmten Ladepunkt abgestellt werden. Die leeren Fahrzeuge werden von einem Zugfahrzeug eingesam-

melt, hintereinander gehängt und zu einem Ladepunkt transportiert, an dem sie anschließend geladen werden.

3.6.2 Der digitale Taximarkt

Die Digitalisierung veränderte in den letzten Jahren maßgeblich den Taxi-Markt. Die urbane Mobilität wird durch Ridesharing-Plattformen sowie Taxi- und Mietwagenvermittlungen unterstützt, in denen die Ressourcen und Kosten geteilt werden können. Durch die Digitalisierung wurde eine technische Grundlage geschaffen, um einfacher Fahrgemeinschaften oder andere Fahrdienste zu finden und Vertrauen durch soziale Informationen und sichere Zahlungsmöglichkeiten zu schaffen. Die Fahrdienstvermittlungs-Apps machen das Taxi-Gewerbe transparenter und Kontrollen durch Finanzbehörden werden wesentlich einfacher, da durch elektronische Zahlungsmethoden Steuerhinterziehung und Schwarzarbeit leichter festgestellt werden können [6]. Auf internationaler Ebene konkurrieren die Fahrdienstvermittler Uber und Didi Kuaidi vor allem in China um Marktanteile mit Hilfe von großen Investitionen. China gilt als attraktivster zu erschließender Markt für die neuen Taxidienste. Uber konnte innerhalb des Jahres 2015 den Marktanteil auf dem chinesischen Markt von einem Prozent auf 35 Prozent steigern. In den USA wird der Markt der Fahrdienstvermittler von Lyft und Uber heiß umkämpft, wobei sich Uber als Marktführer in den USA weiter auf dem Vormarsch befindet, um Martkanteile für sich zu gewinnen. In Deutschland nutzen immer mehr Taxi-Fahrer den Doppelfunk, der seit einem Urteil des Oberlandesgericht Frankfurt in Deutschland von Taxizentralen nicht mehr verboten werden darf. So können Taxi-Fahrer über mehrere Alternativen, wie Taxi-Zentrale oder Taxi-App, Fahraufträge annehmen. Zur Folge der wachsenden Marktanteile für Taxi-Apps und Ridesharing-Plattformen haben sich die europäischen Taxizentralen zusammengeschlossen, um verlorene Martktanteile zurückzugewinnen und der Konkurrenz von mytaxi, Uber und Co. zu begegnen. Ziel dieses Zusammenschlusses ist ein europaweites Taxinetzwerk, das sich derzeit aus den Apps taxi.eu, eCab und Taxi Deutschland kombiniert. Als weitere Alternative will die Taxi-Genossenschaft Taxi Deutschland künftig die WhatsApp Chat-Funktion nutzen, um einen neuen Bestellservice für Taxis bereitzustellen. Das sogenannte "WhatsApp Taxi" soll als Ergänzung für die Taxi-Deutschland-App dienen.

4 Anforderungen

4.1 Funktionale Anforderungen

In diesem Abschnitt werden die funktionalen Anforderungen, die das System erfüllen soll, unterteilt nach einzelnen Systemkomponenten aufgelistet.

4.1.1 App

In diesem Abschnitt werden alle funktionalen Anforderungen an die App dieser Projektgruppe beschrieben. Weiterhin werden diese Anforderungen in die Bereiche *Allgemein*, *Profil*, *Registrierung*, *Login*, *Zieleingabe* und *Ordern des Taxis unterteilt*. Im Folgenden werden die Anforderungen zu jedem Bereich einzelt beschrieben.

Allgemein:

- Der Kunde soll über die App ein Taxi zu sich rufen können.
- Der Kunde soll seinen eigenen Standort tracken können.

Profil:

- Der Kunde soll die Möglichkeit haben sein Passwort zu ändern.
- Der Kunde soll die Möglichkeit haben sein persönlichen Daten (z. B. Name, Username usw.) verändern zu können.

Registrierung:

- Der Kunde soll sich nach dem Starten der App registrieren können.
- Der Kunde soll Namen, Vornamen und E-Mail eingeben können.
- Der Kunde soll sich mit einem Usernamen registrieren können.

Login:

- Der Kunde soll sich nach der Registrierung oder nach dem Starten der App mit seinen Nutzerdaten einloggen können.
- Der Kunde soll ein neues Passwort anfordern können.
- Der Kunde soll die Möglichkeit haben eingeloggt zu bleiben.

Zieleingabe:

- Der Kunde soll sein gewünschtes Ziel eingeben können.
- Der Kunde soll bei der Eingabe seines Ziels Adressvervollständigungsvorschläge von der App erhalten.
- Der Kunde soll sein gewünschtes Ziel auf einer Karte markieren können.
- Der Kunde soll seinen gewünschten Abholort eingeben können.
- Der Kunde soll seinen gewünschten Abholort auf einer Karte markieren können.
- Der Kunde soll einen gewünschten Fahrzeugtyp angeben können.
- Der Kunde soll angeben können, ob er alleine fahren möchte
- Der Kunde soll eine gewünschte Abholungs- und Ankunftszeit angeben können.
- Die App soll eine Karte mit der berechneten Route anzeigen.
- Die App soll die Entfernung zum Ziel und den geschätzten Preis anzeigen.
- Der Kunde soll einen Abholort als Favoriten speichern können.
- Der Kunde soll einen Zielort als Favoriten speichern können.

Ordern eines Taxis:

- Nachdem der Kunde das Taxi bestellt hat, soll die App die aktuelle Position des Taxis auf einer Karte anzeigen.
- Die App soll die aktuelle Zeit bis zur Ankunft des Taxis anzeigen.
- Wird während der Wartezeit ein anderes Taxi zum Kunden gesendet, soll die App die Position und Ankunftszeit des neuen Taxis anzeigen.
- Der Kunde soll einen Auftrag stornieren können.
- Der Kunde soll einzelne Fahrten bewerten können.
- Der Kunde soll angeben können ob er einen Rollstuhl transportiert haben möchte.
- Der Kunde soll angeben können ob er einen Fahrrad transportiert haben möchte.
- Der Kunde soll angeben können ob er zusätzliches Gepäck transportiert haben möchte.
- Der Kunde soll die voraussichtlich gefahrene Route zwischen Abhol- und Zielort angezeigt bekommen.
- Der Kunde soll den voraussichtlichen Preis seiner Fahrt angezeigt bekommen.
- Der Kunde soll die voraussichtliche Streckenlänge angezeigt bekommen.
- Der Kunde soll die voraussichtliche Fahrtdauer angezeigt bekommen.
- Der Kunde soll eine Benachrichtigung bekommen wenn das Taxi am Abholort eintrifft.
- Der Kunde soll zu bestimmten Zeitpunkten bevor das Taxi eintrifft Benachrichtigungen erhalten (Bsp: Ihr Taxi ist in 5 Minuten am Abholort).
- Der Kunde soll angezeigt bekommen, wie lange die Fahrt inklusive Anfahrt des Taxis zum Abholort voraussichtlich dauert.

4.1.2 Dashboard

In diesem Abschnitt werden alle funktionalen Anforderungen an das Dashboard dieser Projektgruppe beschrieben. Weiterhin werden diese Anforderungen in die Bereiche *Allgemein*, *Analyse*, *Taxiflotte*, *Aufträge*, *Events* und *Karte*. Im Folgenden werden die Anforderungen zu jedem Bereich einzelt beschrieben.

Allgemein

- Das Dashboard soll Informationen über Aufträge, Taxis, Wetter, Events und Ladestationen von Odysseus erhalten.
- Der Administrator des Dashboards soll neue Mitarbeiter registrieren können.
- Der Mitarbeiter der Taxizentrale soll sich über das Dashboard einloggen können.
- Der Mitarbeiter der Taxizentrale will sich die Position jedes Taxis einzeln auf einer Karte anzeigen lassen können
- Der Mitarbeiter der Taxizentrale soll sich auf dem Dashboard einloggen können.
- Der Mitarbeiter der Taxizentrale soll in der Lage sein sich Fahrzeuge, Events, Aufträge und Ladestationen zusammen anzeigen lassen zu können.
- Der Mitarbeiter der Taxizentrale soll in der Lage sein sich Fahrzeuge, Events, Aufträge und Ladestationen einzeln anzeigen lassen zu können.

Analyse

- Das Dashboard soll Informationen aus vergangen Aufträgen aufbereitet darstellen können.
- Der Mitarbeiter der Taxizentrale will sich auf der Analyseseite Informationen über die Auslastung der Taxis anzeigen lassen können.
- Der Mitarbeiter der Taxizentrale will sich auf der Analyseseite Informationen über die Anzahl der Passagiere pro Taxifahrt anzeigen lassen können.

- Der Mitarbeiter der Taxiflotte möchte Daten in Form von unterschiedlichen Diagrammtypen visualisieren können.

Taxiflotte

- Das Dashboard soll Informationen über alle Taxis bereitstellen.
- Der Mitarbeiter der Taxizentrale will Informationen über die Kapazität eines Taxis vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über die Position eines Taxis vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über das Modell eines Taxis vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über die Ladung eines Taxis vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über die gefahrene Gesamtstrecke eines Taxis vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über die gefahrene Gesamtstrecke eines Taxis an einem Tag vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über den aktuell bearbeiteten Auftrag eines Taxis vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale soll die Taxiflotte bearbeiten können, d.h. Eigenschaften der Taxis verändern und neue Fahrzeuge hinzufügen können.

Aufträge

- Das Dashboard soll Informationen über aktuelle Aufträge bereitstellen.
- Der Mitarbeiter der Taxizentrale will Informationen über den Auftraggeber eines Auftrags vom Dashboard bereitstellen bekommen.

- Der Mitarbeiter der Taxizentrale will Informationen über den Status eines Auftrags vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über die Personenanzahl eines Auftrags vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über das Datum eines Auftrags vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über die Kosten eines Auftrags vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über den Abholort eines Auftrags vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über den Zielort eines Auftrags vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über die Route eines Auftrags vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über die Streckenlänge eines Auftrags vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über Dauer eines Auftrags vom Dashboard bereitstellen bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über vergangene Aufträge vom Dashboard bereitstellen bekommen.

Events

- Das Dashboard soll Informationen über aktuelle Events bereitstellen.
- Der Mitarbeiter der Taxizentrale will Informationen über den Veranstaltungsort eines Events vom Dashboard bereitgestellt bekommen.
- Der Mitarbeiter der Taxizentrale will Informationen über den Veranstaltungstermin eines Events vom Dashboard bereitgestellt bekommen.

- Der Mitarbeiter der Taxizentrale will Informationen über die erwartete Teilnehmerzahl eines Events vom Dashboard bereitgestellt bekommen.
- Der Mitarbeiter der Taxizentrale soll manuell Events anlegen können.
- Das Dashboard soll Informationen über vergangene Events bereitstellen.
- Das Dashboard soll Informationen über Events von Facebook bekommen und bereitstellen.

Karte

- Das Dashboard soll eine Karte enthalten, in der Auftrage, Taxis, Events und Ladestationen visualisiert werden können.
- Die Karte soll Routen von Taxis visualisieren können.
- Die Karte soll den Standort von Events in Oldenburg und Umgebung anzeigen.
- Die Karte soll den Standort von Taxis in Oldenburg und Umgebung anzeigen.
- Die Karte soll den Standort von Aufträgen in Oldenburg und Umgebung anzeigen.
- Die Karte soll voreingestellt sein auf den Bereich Oldenburg und Umgebung.

4.1.3 GraphHopper

In diesem Abschnitt werden alle funktionalen Anforderungen an die Routing-Engine von GraphHopper beschrieben.

Allgemein

- GraphHopper muss aus der Oldenburg-Karte einen Graphen erzeugen.
- GraphHopper muss eine Route zwischen zwei Punkten berechnen.
- GraphHopper muss die Kosten der Route zwischen zwei Punkten an JSprit übermitteln.

- GraphHopper muss die optimale Reihenfolge der Wegpunkte von JSprit empfangen.
- GraphHopper muss eine Route zwischen mehr als zwei Punkten berechnen.
- GraphHopper muss GPS-Koordinaten zu Straßen(-abschnitten) zuordnen.
- GraphHopper muss die Route an die Verkehrssimulation senden.
- GraphHopper muss die Route eines Passagiers bereitstellen.
- GraphHopper muss die Route eines Taxis bereitstellen.
- GraphHopper muss den Ankunftszeitpunkt des Taxis beim Passagier bereitstellen.
- GraphHopper muss den Name des zugeordneten Taxis für den Passagier bereitstellen.
- GraphHopper muss Informationen über gesperrte Straßen von Odysseus erhalten.
- GraphHopper muss den Graphen bei gesperrten Straßen aktualisieren.
- GraphHopper muss Informationen über eine neue Geschwindigkeit auf einer Straße von Odysseus empfangen.
- GraphHopper muss den Graphen bei einer neuen Geschwindigkeit auf einer Straße aktualisieren.
- GraphHopper muss stornierte Aufträge bei der Routenberechnung einbeziehen.
- GraphHopper muss neue Aufträge bei der Routenberechnung einbeziehen.
- GraphHopper muss einen veränderten Ablieferpunkt emfpangen.
- GraphHopper muss den Graphen bei veränderten Abholpunkten aktualisieren.

- GraphHopper muss den Graphen bei veränderten Ablieferpunkten aktualisieren.
- GraphHopper muss bei einem aktualisierten Graphen die Routenberechnung durchführen.

4.1.4 JSprit

In diesem Abschnitt werden alle funktionalen Anforderungen an JSprit beschrieben.

Allgemein

- Jsprit muss die Fahrzeugflotte anlegen.
- JSprit muss die Aufträge für die Optimierung anlegen.
- JSprit muss die Kosten auf einer Kante zw. zwei Wegpunkten von GraphHopper empfangen.
- JSprit muss eine asym. Kostenmatrix mit den Kosten auf allen Kanten aufbauen.
- JSprit muss die Kostenmatrix zu dem Optimierungsproblem hinzufuegen.
- JSprit muss das Optimierungsproblem (DARP) aufbauen.
- JSprit muss das Optimierungsproblem (DARP) lösen.
- Jsprit muss Auftragsinformationen bereit stellen.
- Jsprit muss die optimale Reihenfolge der Wegpunkte an GraphHopper übermitteln.
- JSprit muss neue Kosten auf einer Kante empfangen.
- JSprit muss die Kostenmatrix aktualisieren.
- JSprit muss stornierte Aufträge in der Optimierung berücksichtigen.
- JSprit muss neue Aufträge in der Optimierung berücksichtigen.

- JSprit muss eine veränderte Zahl an Fahrgästen bei der Optimierung berücksichtigen.
- JSprit muss die Anzahl der Fahrzeuge in der Fahrzeugflotte mit deren Sitzplätzen empfangen.

4.1.5 SUMO

In diesem Abschnitt werden alle funktionalen Anforderungen an SUMO und TraCI beziehungsweise dem Java, Jython und Python Projekt, was zur Kommunikation aufgebaut wurde.

Obligatorisch

- Sumo soll Informationen über den Standort der Taxis liefern.
- Sumo soll Informationen über Staus liefern.
- Sumo soll die Reisezeit auf einer Kante zurück geben.
- Sumo soll ein realistisches Verkehrsaufkommen simulieren.
- Sumo soll eine GPS-Position zu einer Straße umwandeln können.
- Sumo soll Haltepunkte für ein Taxi definieren können.
- Sumo soll die Route eines Fahrzeuges verändern können.
- Sumo soll neue Fahrzeuge hinzufügen können.
- Sumo soll Fahrzeuge entfernen können.
- Sumo soll Statuswerte der Fahrzeuge verändern können.

Optional

- Sumo soll die aktuelle Ladung der Batterie eines Fahrzeuges liefern.
- Sumo soll den elektrischen Verbrauch eines Fahrzeuges liefern.
- Sumo soll Ladesäulen zur Verfügung stellen.

4.2 Nicht-Funktionale Anforderungen

Die Projektgruppe orientiert sich bei den nicht-funktionalen Anforderungen an die internationale Norm ISO/IEC 9126. Diese Norm umfasst neben den sechs übergeordneten Qualitätsmerkmalen Benutzbarkeit, Änderbarkeit, Effizienz, Zuverlässigkeit, Übertragbarkeit und Funktionalität weitere 26 untergliederte Teilmerkmale. Damit ist ISO/IEC 9126 grundsätzlich auf jede Art von Software anwendbar.
Eine für die Projektgruppe wichtige nicht-funktionale Anforderung ist die Modularität des Systems. Während innerhalb des Projekts nur die Routenberechnung für simulierte Fahrzeuge erfolgt, soll die Verkehrssimulations-Software austauschbar sein.

5 Evaluationen zum Systementwurf

Im Verlauf der Projektgruppe wurden zu bestimmten Problemstellungen und Lösungsalternativen Evaluationen durchgeführt. Diese wurdem im Anschluss der Gruppe vorgestellt, die auf Basis dessen eine gemeinsame Entscheidung über das weitere Vorgehen getroffen hat. Im Verlauf dieses Kapitels werden die von der Projektgruppe durchgeführten Evaluationen im einzelnen erläutert.
Zunächst wird im Kapitel 5.1 die Evaluation zur Umsetzung der App beschrieben. Im Kapitel 5.2 folgt eine Evaluation zur Umsetzung der Facebook-Schnittstelle zum Erhalten der Facebook-Veranstaltungen. Danach wird in den Kapiteln 5.3 und 5.4 die normale sowie die reaktive Standortplanung evaluiert. Weiterhin erfolgen in den Kapiteln 5.5 und 5.6 Evaluationen zum Aufbau und zur Umsetzung des Dashboard. Abschließend wird im Kapitel 5.7 evaluiert welche Verkehrssimulation diese Projektgruppe an Odysseus anbinden möchte.

5.1 Evaluation App

Zu Beginn der Projektgruppe wurden Überlegungen angestellt, welchem Zweck die App dienen soll. Dazu wurde festgehalten, dass ein Kunde über die App in der Lage sein soll ein Taxi zu bestellen. Weiterhin soll der Kunde jedoch auch noch zusätzliche Informationen bereitsgestellt bekommen, wie beispielsweise die voraussichtlich gefahrene Route, den voraussichtlichen Preis, die aktuelle Position des Taxis usw.
Daraufhin wurden prinzipiell folgende Schritte zur Bestellung eines Taxis und zur Erstellung eines Auftrags festgelegt:

- Der Kunde ruft App auf
- Der Kunde loggt sich ein (sofern der Kunde noch keinen Account hat, muss er sich zunächst registrieren
- Der Kunde gibt sein Ziel ein
- Die App sendet die GPS-Daten des Kunden und des Ziels an Odysseus
- Odysseus sendet dei GPS-Daten an SUMO und fragt ein freies, sich in der Nähe befindliches Taxi an

- SUMO sendet die GPS-Daten des Taxis an Odysseus
- Odysseus berechnet Route, Entfernung, Preis und Wartezeit
- Odysseus sendet Daten an die App
- Der Kunde bestätigt den Auftrag
- Die App sendet die Bestätigung an Odysseus
- Odysseus sendet die neue Route an SUMO
- SUMO schickt das Taxi los und sendet die GPS-Daten des Taxis kontinuierlich an Odysseus
- Odysseus leitet die GPS-Daten des Taxis kontinuierlich an die App weiter

Zur Umsetzung der App entstand die Frage, in welcher Form diese umgesetzt werden soll. Dazu gab es prinzipiell die Möglichkeiten der Entwicklung einer nativen, hybriden oder Web-App. In dieser Evaluation wurden Frameworks und Technologien gesucht durch die eine Entwicklung der verschiedenen Appformen möglich ist.

5.1.1 Web-App

Die Web-App wird in eine Server- und eine Clientseite unterteilt und für beide Seiten wurden verschiedene Frameworks gesucht.
Dabei wurden folgenden serverseitigen Frameworks gefunden:

Node.js

- JavaScript-basierte Plattform für asynchrone Events in Netzwerkanwendungen
- Ereignisgesteuerte Architektur
- Ermöglicht die Verarbeitung eintreffender Datenströme und Erzeugung ausgehender Datenströme
- Ermöglicht eine große Zahl gleichzeitig bestehender Netzwerkverbindungen

Express.js

- Erweiterung von Node.js
- Bietet Werkzeuge zur einfachen Entwicklung von Webanwendungen
- Ermöglicht einfache und schnelle Erstellung von APIs durch HTTP-Dienstprogrammmethoden und Middlewarefunktion
- Wird oft in Kombination mit MongoDB, AngularJS und node.js (MEAN) verwendet

MongoDB

- Schemafreie und dokumentenorientierte NoSQL-Datenbank

Socket.io

- Framework für den Austausch von Informationen zwischen Client und Server in Echtzeit
- Nutzt WebSockets, ein auf TCP basiertes Netzwerkprotokoll, das parallel zur HTTP-Anfrage des Browsers eine weitere Verbindung zum Server herstellt, die für die gesamte Dauer des Seitenbesuchs permanent bestehen bleibt
- Setzt auf node.js auf

Weiterhin wurden noch die folgenden clientseitigen Frameworks gefunden:

AngularJS

- JavaScript-Framework für die Erstellung von Singlepage-Anwendungen nach dem Model-View-View-Model-Muster

Bootstrap

- CSS-Framework
- Bietet HTML- und CSS-basierte Gestaltungsvorlagen für Oberflächenelemente

5.1.2 Hybride App

Auch für hybride Apps wurden mögliche Framework evaluiert. Dabei entstanden folgende Ergebnisse:

Ionic

- Frontend-Framework mit eigenem Kommandozeilentool
- Einfache Entwicklung dank SASS, HTML und CSS
- Ermöglicht Integration von AngularJS

Intel XDX

- Bietet Tools für Entwicklung, Emulation, Testing und Debugging sowie auf dem Open-Source-JavaScript-Framework von Intel basierte „Ready-to-Use“-App-Templates
- Unterstützt Bootstrap 3, jQuery mobile und das Topcoat-UI-Framework

5.1.3 Native App

Bei dieser Form der App stellte sich eigentlich nur die Entscheidung zwischen einer Appentwicklung für iOS oder Android. Dazu wurdem im wesentlichen zwei Dinge in Betracht gezogen. Zum einen ist die Entwicklung einer iOS-App nur auf Geräten der Firma Apple möglich, zum anderen hatten bereits mehrere Gruppenmitglieder Erfahrungen mit der Entwicklung von Android-Apps.

5.1.4 Webdienste

Weiterhin wurden verschiedene Webdienste evaluiert, die bei der Umsetzung behilflich sein können. Es entstanden folgende Ergebnisse:

Google Maps Geolocation API

- Gibt den Standort und den Präzisionsradius von einem mobilen Client basierend auf Daten von Mobilfunkmasten und WiFi-Knoten zurück

Google Maps Android API v2

- Ermöglicht das Hinzufügen von Karten basierend auf Google Maps-Daten zur App
- Ermöglicht die Anzeige von Taxis durch Marker und Routen

Firebase Cloud Messaging

- Webdienst zur Implementierung von Push Notifications

5.2 Evaluation zur Facebook-Schnittstelle

Im Rahmen dieser Projektgruppe wollen wir eine Standortplanung der Taxis auf Basis von in Oldenburg stattfindenden Veranstaltungen durchführen. Die dazu notwendigen Informationen, wie beispielsweise Teilnehmnerzahl, Veranstaltungsort, Veranstaltungszeitraum usw., sollen von Facebook bezogen werden. Dazu muss eine Schnittstelle zu Facebook implementiert werden, über die diese Informationen beschafft werden können. Im Rahmen dieser Evaluation wurden die folgenden Möglichkeiten identifiziert:

Graph API

- Basis für das Lesen von Daten aus Facebook ist die Graph API: https://developers.facebook.com/docs/graph-api/overview
- Für die Verwendung der Graph API wird ein Access Token benötigt, das durch Login eines Facebook Accounts angefordert werden kann: https://developers.facebook.com/tools/explorer/
- Facebook Schnittstelle in Odysseus stellt TransportHandler und ProtocolHandler
- Bisher werden nur die Felder „Message“, „ID“ und „timestamp“ gelesen – Quellcode müsste um Felder für Veranstaltungsdaten erweitert werden
- Problem: Es können mit einer Anfrage nicht alle Events innerhalb des Radius einer Koordinate wiedergegeben werden
- Möglichkeit: Suche nach Stichwort „Oldenburg“ in metadata von Events?

facebook-events-by-location [6]

- Auf Express.js basierender Webservice, mit dem Events nach Locations ausgegeben werden
- Suche nach Events erfolgt in drei Schritten:
  - Search for places in the radius of the passed coordinate and distance
  - Use the places to query for their events in parallel
  - Unify, filter and sort the results from the parallel calls and return them to the client
- Ausgabe im JSON-Format

Firebase Cloud Messaging

- Webdienst zur Implementierung von Push Notifications

5.3 Evaluation Standortplanung

Im Rahmen unserer Projektgruppe müssen besondere Bedingungen in die Wahl der Standorte einbezogen werden. Da wir komplett auf Elektrofahrzeuge setzen, ist es sinnvoll den Standort des Taxis möglichst immer an einer Ladesäule zu setzen. Dies gestaltet sich in der Realität allerdings schwierig, da es in Oldenburg nur ungefähr 15 Ladesäulen gibt und somit eine effiziente Standortplanung unmöglich ist. Unsere Projektgruppe plant eine Standortplanung auf Basis von in Oldenburg stattfindenden Events. Dazu sollen vor allem die Teilnehmerzahl und der Ort des Events Berücksichtigung finden. Zur Ermittlung des idealen Standorts des Taxis ließen sich klassische Verfahren zur Berechnung des optimalen Standorts verwenden, allerdings wird durch diese z. B. der Punkt mit der kürzesten Entfernung zu allen Events berechnet. Dies kann als langfristiges Nebenziel gesehen werden, ist aber zunächst nicht zielführend, da der Kunden wieder ein Taxi bestellen und einige Minuten darauf warten müsste. Der Kunde möchte jede wahrscheinlich von dem Event kommen und direkt vor der Location in ein Taxi steigen. Dieser Ansatz

[6] `https://www.npmjs.com/package/facebookevents-by-location`

lässt sich in einer zunächst noch sehr einfachen Variante verfolgen, indem die Teilnehmeranzahl überprüft wird und ab bestimmten Schwellwerten ein Taxi direkt vor dem Event abgestellt wird. Zum Beispiel könnte man festlegen, dass aber 100 Teilnehmern ein Taxi abgestellt wird, ab 300 zwei Taxis usw. Um die Informationen über die Events zu erhalten soll eine Schnittstelle von Odysseus zu Facebook entwickelt werden, über die die Informationen zu Facebookevents abgefragt werden können.
Ein weiterer Ansatz ist, dass Odysseus dauerhaft die eintreffenden Aufträge überwacht und wenn in einem definierten Bereich in einem bestimmten Zeitraum viele Aufträge eingehen, dann automatisch ein Taxi dahin geschickt wird. Beispielsweise könnte festgelegt werden, dass wenn 5 Aufträge in 30 Minuten in einem Umkreis von 500 Metern eingehen, dass dann automatisch ein Taxi in diesen Bereich geschickt wird. (siehe Kapitel 5.4)

5.4 Evaluation reaktive Standortplanung

Bei der reaktiven Standortplanung soll auf ein erhöhtes Auftragsaufkommen reagiert werden. Dazu sollen die einkommenden Aufträge in einem definierten Bereich in einer vordefinierten Zeitpanne ausgewertet werden. So soll erkannt werden, wo aktuell Events oder Menschenansammlungen sind, um drauf zu reagieren. So kann ein Taxi in dem Bereich mit dem erhöhten Auftragsaufkommen platziert werden und die Strecke und Wartezeit des Kunden minimiert werden. Dazu soll Oldenburg in ein Raster mit gleich großen Teilen unterteilt werden (siehe Abbildung unten). Wird nun in einer bestimmten Zeitspanne ein bestimmter Grenzwert an Aufträgen in diesem Bereich überschritten, so wird ein Taxi in den Bereich geschickt. Ein Vorschlag hier sind 5 Aufträge in 15 Minuten. Als ein weiterer Schritt kann dieser Grenzwert dynamisch an das aktuelle Auftragsaufkommen angepasst werden und somit die Wahrscheinlichkeit auf zufälliges Auslösen eines des Bereichs verringert werden.
Das Rastermodell über Oldenburg bietet allerdings auch Nachteile, so werden Aufträge die knapp außerhalb eines Kastens liegen nicht mit einbezogen so kann keine optimale Standortplanung der Taxiflotte gewährleistet werden. Deshalb wäre langfristig ein Modell zu überlegen bei dem die Bereiche dynamisch zu den Aufträgen berechnet werden.

5.5 Evaluation Spring (Java Framework)

Das Spring Framework ist ein quelloffenes Framework für die Java. Ziel ist die Vereinfachung der Entwicklung von Java/JavaEE Anwendungen. Spring bietet ein breites Spektrum an Funktionalitäten, sowie eine Menge Schnittstellen zu weiteren Systemen. Es gibt sehr viele Tutorials, Dokumentationen und eine breite Community. Hierbei werden Programmierparadigmen genutzt die den Code voneinander lösen und leichter testbar machen. Es gibt vordefinierte Templates, sodass der Code z.B. bei JDBC (Java Database Connector) für das Exception Handling, creating connection, creating statement, committing transaction, closing connection etc. deutlich kürzer ausfällt.
Das Dashboard, das mit dem JavaScript-Framework AngularJS entwickelt wurde, ist derzeit in ein SpringMVC Projekt integriert und kann bisher nur auf einen Tomcat 8 Server ausgeführt werden.
Leider hat es enorm viel Zeit benötigt ein lauffähiges Projekt zu erstellen. Allerdings verspricht das Spring sehr viele Vorteile. Wir erhalten als Produkt ein Maven-Projekt, womit wir unser Projekt standardisiert erstellen und verwalten können. Spring bietet eine große Unterstützung zur Erstellung von REST Webservices in Verbindung mit AngularJS als auch von Android-Anwendung. Als Anreiz sich mit dem Spring Framework auseinanderzusetzen, kann man sich z.B. die Stellenangebote verschiedener Software-Agenturen anschauen, bei denen Kenntnisse in Spring eine Voraussetzung darstellen und somit eine Einarbeitung sehr interessant sein kann. Nach Gesprächen mit der letzten Projektgruppe wurde Spring schon in Verbindung mit Odysseus entwickelt, sodass das Projekt mit „Sehr Gut“ abgeschlossen wurde. Das Web bietet eine große Anzahl von Tutorials, Beispielen auf Github, Gute Dokumentation sowie eine große Community.

5.6 Evaluation Diagramm im Dashboard

Für das Management des Taxiunternehmens soll eine Webseite Informationen zu Aufträgen, Taxis, Wetter, Events, Fahrgäste und Ladestationen bereitstellen. Hierbei sollen passende Darstellungsformen für die jeweilige Thematik gewählt werden. Neben der klassischen Auflistung, dessen Elemente jeweils zu einer Detailansicht in Textform führen, soll eine Karte und verschiedene Diagrammen die Informationen aufbereiten.

Zur Visualisierung wird die von Google zur verfügung gestellte Chart API verwendet. Vordefinierte Diagramme decken alle wünschenswerten Diagramm Typen ab und können bei Bedarf angepasst werden. Die Definition und die Übergabe der zu Grunde liegenden Daten findet in JavaScript statt. Der Einsatz der Google Charts API in AngularJS ist folglich ohne großen Aufwand möglich.
Zunächst werden die für eine nähere Auswahl sinnvoll erscheinende Diagramme aufgeführt:

- Column/Bar Chart

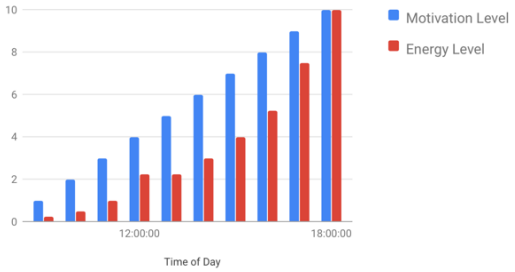


Abbildung 7: Beispiel für ein Column/Bar Chart

- Line Chart

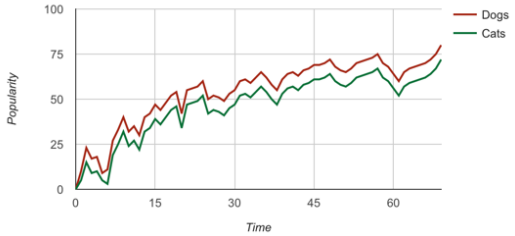


Abbildung 8: Beispiel für ein Line Chart

- Area Chart

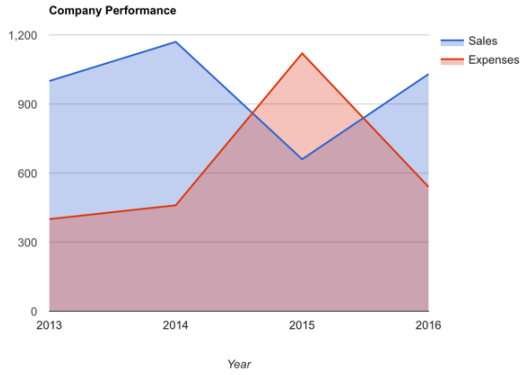


Abbildung 9: Beispiel für ein Area Chart

- Stepped Area Chart

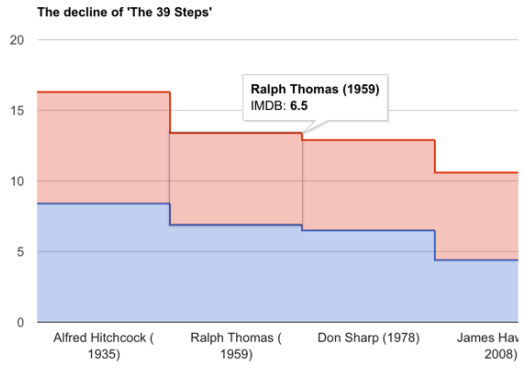


Abbildung 10: Beispiel für ein Stepped Area Chart

- Pie Chart

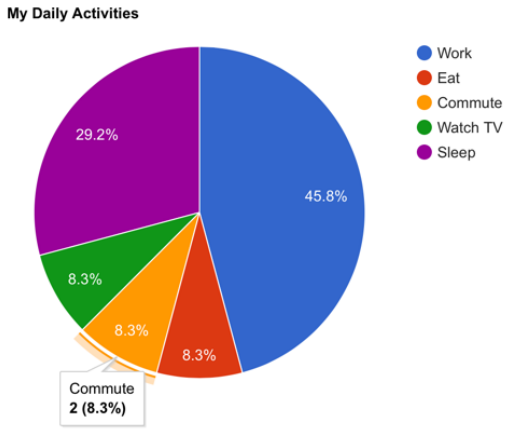


Abbildung 11: Beispiel für ein Pie Chart

- Timeline Chart

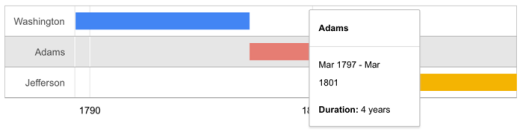


Abbildung 12: Beispiel für ein Timeline Chart

Basierend auf den verfügbaren Daten der einzelnen Sachverhalte, sollen angemessene Diagramm Typen den jeweiligen Sachverhalt visuell aufbereitet und gut verständlich darstellen. Diese Überlegungen werden im folgenden in Abhängigkeit der Dashboard Seite ausgeführt.

- Aufträge
- Kunden
- Fahrzeuge
- Externe Einflüsse: Events, Wetter, Verkehrsereignisse
- Ladesäulen
- Historie

Bei Zeitbezug sollte eine oder mehrere geeignete Größen gewählt werden (Stunde, Tag, Woche, Monat).

- aktueller Wert (aktuelle Personenanzahl je Taxi)
- Durchschnitt/Kumuliert in einem festen Zeitraum (durchschnittliche Personenanzahl in einer Stunde)
- Werte nach Zeit (Personenanzahl je Taxi im Zeitverlauf)

5.6.1 Aufträge

Wesentliche Informationen für die Auftragsverwaltung sind Auftragsanzahl und Qualitätskriterien. Die Anzahl der Aufträge und die mit den Aufträgen verbundenen Beförderungskilometer können in einer Line Chart über die Zeit aufgetragen werden. Auch eine Darstellung der Personenanzahl pro Auftrag wäre denkbar, jedoch zunächst nicht wesentlich. In einer weiteren Grafik kann zur Qualitätssicherung die Wartezeit der bevorstehenden Aufträge je Taxi angezeigt werden. Alternativ kann ebenso die durchschnittliche Wartezeit in einer Area Chart dargestellt werden. Sofern eine Kostenrechnung realisiert wird, kann ein Area Chart Umsatz und Gewinn im Zeitverlauf abbilden. Denkbar wäre ebenso die Darstellung eines variablen Fahrpreises als Line Chart, der sich Abhängig von Angebot und Nachfrage verhält.
Da Aufträge ebenso einen mengenmäßigen Ortsbezug aufweisen, wäre eine Darstellung in einer Heat Map sinnvoll. Um zusätzlich einen Zeitbezug zu realisieren, könnte die Darstellung im Heat Map Overlayer von einer horizontalen Scrollbar abhängen.

5.6.2 Kunden

Grafische Auswertungsmöglichkeiten zu Kunden sollen nicht abgebildet werden. Die Auflistung von Kunden soll ausschließlich dessen Verwaltung dienen.

5.6.3 Fahrzeuge

Eine Visualisierung wesentlicher Informationen zu den eingesetzten Fahrzeugen kann in kumulierter und Einzelwertung vorgenommen werden. So können Übersichtsdiagramme schnell einen Eindruck über die aktuelle Situation aller Fahrzeuge

bieten. Grafiken mit Informationen zu einzelnen Fahrzeugen hingegen liefern einen detaillierten Einblick. Verfügbare Werte über die Taxis die u. a. in Diagrammen Verwendung finden können sind die Folgenden:

- Taxi ID
- Position
- Akkuladung
- aktuelle Personenanzahl im Fahrzeug
- Status

5.6.4 Ladestationen

Ladestationen bieten derzeit neben Standort, Anschlussart und Maximale Spannung wenig relevante Informationen für die Taxiverwaltung, die einer grafischen Würdigung bedürfen. Neben der Auflistung und Kennzeichnung der Stationen auf der Karte könnte die Belegung der Ladestationen mit den zu verwaltenden Taxis in einer Timeline veranschaulicht werden. Der Ladeverlauf wird hingegen bereits im Zusammenhang mit den Fahrzeugen als Akkuladung über die Zeit an anderer Stelle erfasst. Sofern die Strompreise pro Säule zukünftig öffentlich verfügbar sind, könnte der Preisverlauf pro Säule in einem Line Chart von Interesse sein.

5.6.5 Externe Einflüsse

Externe Einflüsse unter die bspw. Wetterlage, Verkehrsereignisse und Veranstaltungen fallen, können aktiv die Nachfrage an Aufträgen und das Fahrverhalten der Taxi beeinflussen. Die erfassten Einflüsse sollen neben einer Auflistung ebenso in grafischer Form dargestellt werden.
Die Darstellung der Wettervorhersage ist auf Webseiten von Wetterdiensten meist sehr einheitlich getroffen. Die aktuellen Bedingungen, wie Sonnenschein, Wolken, Nebel, Regen oder Gewitter, sind in Piktogrammen vorzufinden. Weitere Werte wie Temperatur, Regenwahrscheinlichkeit oder Niederschlagsmenge sind neben kleinen Icons aufgeführt.

Grundsätzlich sollte eine Beschränkung auf für die Fahrzeugverwaltung relevante Daten stattfinden. Die größte Relevanz wäre dabei max./min. Temperatur, Sonnenstunden, Niederschlagswahrscheinlichkeit und -menge zuzuschreiben. Aus der Kombination dieser Daten wird das Verhalten der Kundenmasse statistisch vorhersehbar. (Link zum Kapitel Einflussbasierte Vorhersage) Die für diesen Anwendungsfall interessantere Darstellung von max. und min. Temperaturen und Niederschlagsmengen in einem kombinierten Diagrammen (Combo Chart = Line Chart und Column Chart) ist auch in längerfristigen Wettervorhersagen vorzufinden. Es soll lediglich eine Ergänzung um die Sonnenstunden pro Tag vorgenommen werden. Auch denkbar wäre eine Darstellung der verschiedenen Temperaturzonen auf einer Karte (Heatmap). Da der Aktionsradius der Taxiverwaltung jedoch zunächst nicht größer als Oldenburg und Umgebung ist, kann auf eine gebietsabhängige Darstellungsform des Wetters verzichtet werden.
Ereignisse zur Verkehrslage und Veranstaltungen haben ebenso einen Ortsbezug, sind aber gegenüber der Wetterlage im betrachteten Bereich viel feiner verteilt. Eine Darstellung von Straßensperrungen und Veranstaltungsorten ist daher in jedem Fall in einer Karte vorzunehmen. Dabei können für verschiedene Verkehrsereignisse eindeutige Symbole verwendet werden. Weitere Informationen wie Zeitraum und Quelle sollten zur Wahrung der Übersichtlichkeit einer Auflistung zu entnehmen sein und nicht innerhalb der Karte platziert werden. Für Veranstaltungen kann hingegen auch innerhalb der Karte eine grobe Einschätzung der Besucherzahl von Interesse sein. Eine mögliche Darstellungsweise wäre ein zur Personenanzahl äquivalenter halbtransparenter Radius um den Veranstaltungsort.
Sofern eine Grafik die vorhandenen Verkehrsereignisse abbilden soll, könnte dies mit einer Line Chart und der Anzahl der jeweiligen Ereignisse im zeitlichen Verlauf erfolgen. Gleiches gilt für Veranstaltungen, wobei die die Besucherzahlen deutlich aussagekräftiger ist und in einer Combo Chart abgebildet werden kann.
Um den prozentualen Anteil der Arten von Verkehrsereignissen oder Besucherzahlen nach Veranstaltung abzubilden, eignet sich ein Pie Chart.

5.7 Evaluation Verkehrssimulation

Zur Bestimmung der Verkehrssimulation, die in diesem Projekt Verwendung finden soll, wurde von einer Kleingruppe eine Nutzwertanaylse zur bestehenden Auswahl

an Verkehrssimulationen durchgeführt. Die in Betracht gezogenen Simulationen sind *SUMO (Simulation of Urban MObility)*, *MATSim*, *Mobility Testbed* und *Gama*.
In die Betrachtung der Verkehrssimulationen wurden folgende Anforderungen mit jeweils zugehörigen Gewichtungen einbezogen:

- Anpassbarkeit – 0,3
- Benutzerfreundlichkeit – 0,05
- Funktionsumfang – 0,1
- Schnittstellen – 0,2
- Importfunktion – 0,1
- Dynamische Steuerung – 0,15
- Dokumentation – 0,1

Die Ergebnisse der Nutzwertanalyse lassen sich Abbildung 13 entnehmen. Dabei wird durch die Farbe der einzelnen Spalten symbolisiert, wie gut sich die jeweilige Simulation für die Anforderungen unserer Projektgruppe eignet. Dabei steht Grün für gut, Gelb für mittel und Rot für schlecht. Am besten eignet sich folglich *SUMO* für unsere Zwecke, Mobility Testbed ist noch in Ordnung und Gama sowie MATSim eignen sich nicht. Auf Basis dieser Nutzwertanalyse fiel die entgültige Entscheidung der Gruppe für den Einsatz von **SUMO**.

Anforderungen	Gewichtung	Sumo	Gesamt	MobilityTestbed	Gesamt	Gama	Gesamt	MATSim	Gesamt
Anpassbarkeit	0,3	2	0,6	3	0,9	2	0,6	2	0,6
Benutzerfreundlichkeit	0,05	3	0,15	2	0,1	2	0,1	1	0,05
Funktionsumfang	0,1	2	0,2	2	0,2	2	0,2	2	0,2
Schnittstellen	0,2	3	0,6	2	0,4	1	0,2	1	0,2
Importfunktion	0,1	2	0,2	1	0,1	3	0,3	2	0,2
Dynamische Steuerung	0,15	3	0,45	1	0,15	1	0,15	2	0,3
Dokumentation	0,1	3	0,3	2	0,2	3	0,3	3	0,3
Gesamt	1	18	2,5	13	2,05	14	1,85	13	1,85

Abbildung 13: Nutzwertanalyse zu den Verkehrssimulationen

6 Systementwurf und Implementierung

6.1 Einleitung

6.1.1 Zweck des Systems

Der Zweck das Systems besteht darin, eine optimierte Routen- und Standortplanung für Taxis zu ermöglichen. Übliche Taxi-Unternehmen beachten bei der Abholung der Kunden häufig nicht die Nähe der Kunden untereinander, d. h. jeder Auftrag eines Kunden wird weitestgehend unabhängig von anderen bearbeitet. Unser System hingegen soll mit Hilfe des Datenstrommanagementsystems Odysseus mehrere Kundenaufträge kombinieren können. Dadurch soll es einerseits zu Kosteneinsparungen auf Seiten des Taxiunternehmens kommen und andererseits dem Kunden helfen Geld zu sparen, wenn dieser bereit ist, mit anderen Kunden in einem Taxi zu fahren. Die Optimierung erfolgt unter anderem durch das Lösen des Dial-a-Ride-Problems (siehe Kapitel 6.4).
Zusätzlich zur Optimierung der Routen soll das System dazu genutzt werden können, eine gezielte Standortplanung vorzunehmen. D. h. Taxis sollen an Orten platziert werden, an denen höheres Aufkommen von Aufträgen prognostiziert wird. Dadurch soll der Kunde schneller an sein Ziel gelangen.
Des Weiteren simuliert das System die enthaltenen Taxis als Elektrofahrzeuge. D. h. das System arbeitet unter Betrachtung der Reichweitenproblematik von Elektrofahrzeugen und bezieht mit ein, dass diese regelmäßig geladen werden müssen.

6.1.2 Entwurfsziele

Neben den oben beschriebenen Anforderungen haben wir den Entwurf betreffende Ziele und Qualitätskriterien festgehalten. Diese unterscheiden sich je nach Komponente des Projekts. Besonders wichtig für das System ist, dass die Komponente SUMO ersetzbar ist, um einen Einsatz in der Praxis einfacher zu ermöglichen. Das System soll also insgesamt *modular* aufgebaut sein. Im Vordergrund des Projektes steht die *Funktionalität* des Systems. Die Komponenten GraphHopper und JSprit sind nötig, um einen Mehrwert für das System zu generieren. Daher liegt das Hauptaugenmerk darauf, diese Komponenten sinnvoll zu nutzen. Die Qualitätskriterien der Komponenten des Dashboards und der App beziehen sich vor allem auf die *Gebrauchstauglichkeit*. So soll das Dashboard als Überwachungswerk-

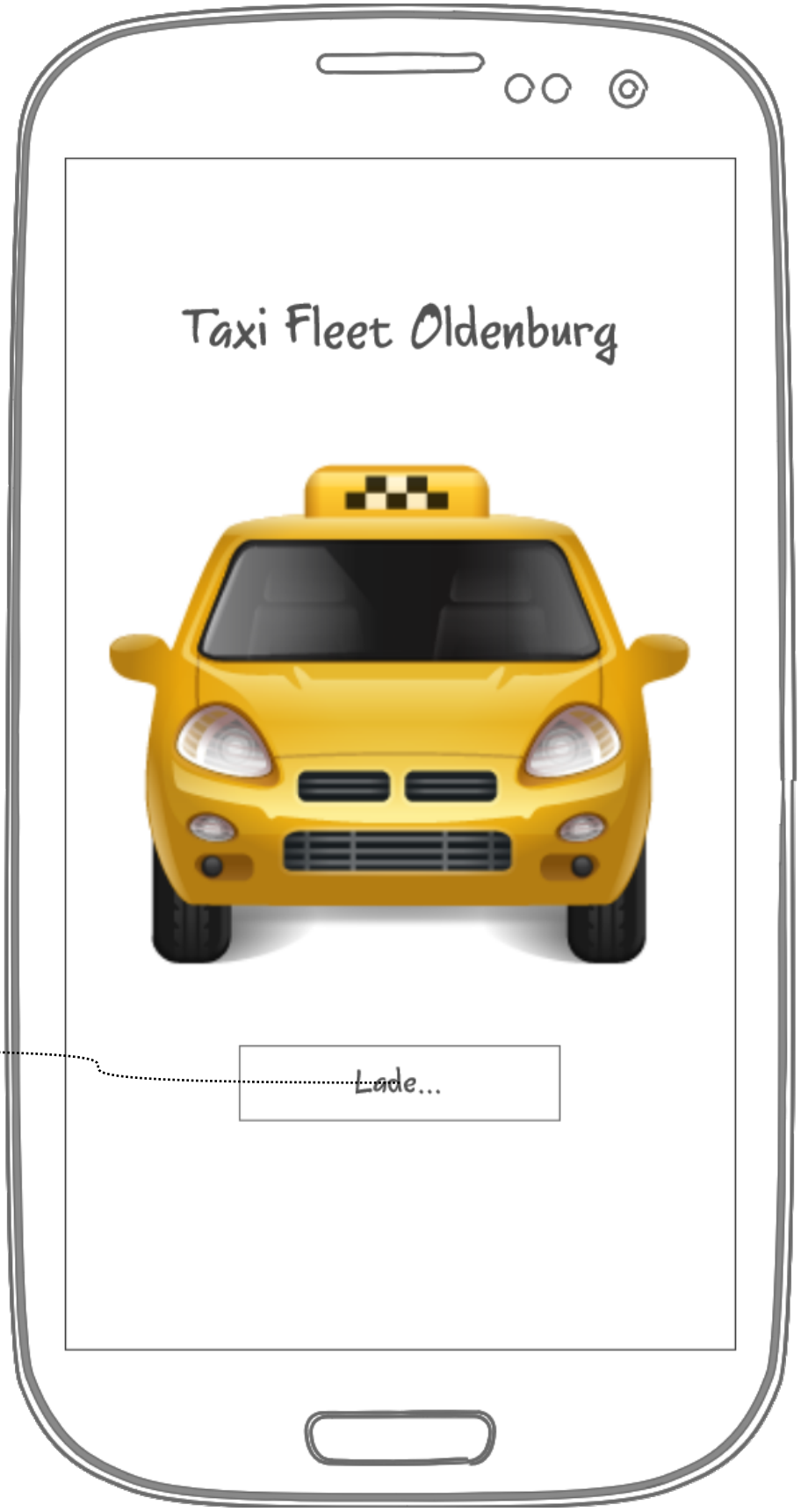
zeug darauf ausgelegt sein, dass der Mitarbeiter des Taxiunternehmens das Produkt gut bedienen kann. Auch für die App gilt, dass sie gebrauchstauglich sein soll. Der Fokus des Systems soll letztendlich aber nicht darauf liegen, die perfekte Gebrauchstauglichkeit zu erzielen. Neben den beschriebenen Kriterien soll das System darauf ausgelegt sein, das Projekt nach Abschluss weiterzuentwickeln. Daher wird Wert auf die *Lesbarkeit* und *Erweiterbarkeit* des Systems gelegt.

6.1.3 Mock-Up App/Dashboard

Im Folgenden werden mit dem kostenlosen Tool NinjaMock[7] erstellte Mock-Ups für die Benutzeroberflächen der App sowie des Dashboards aufgeführt.

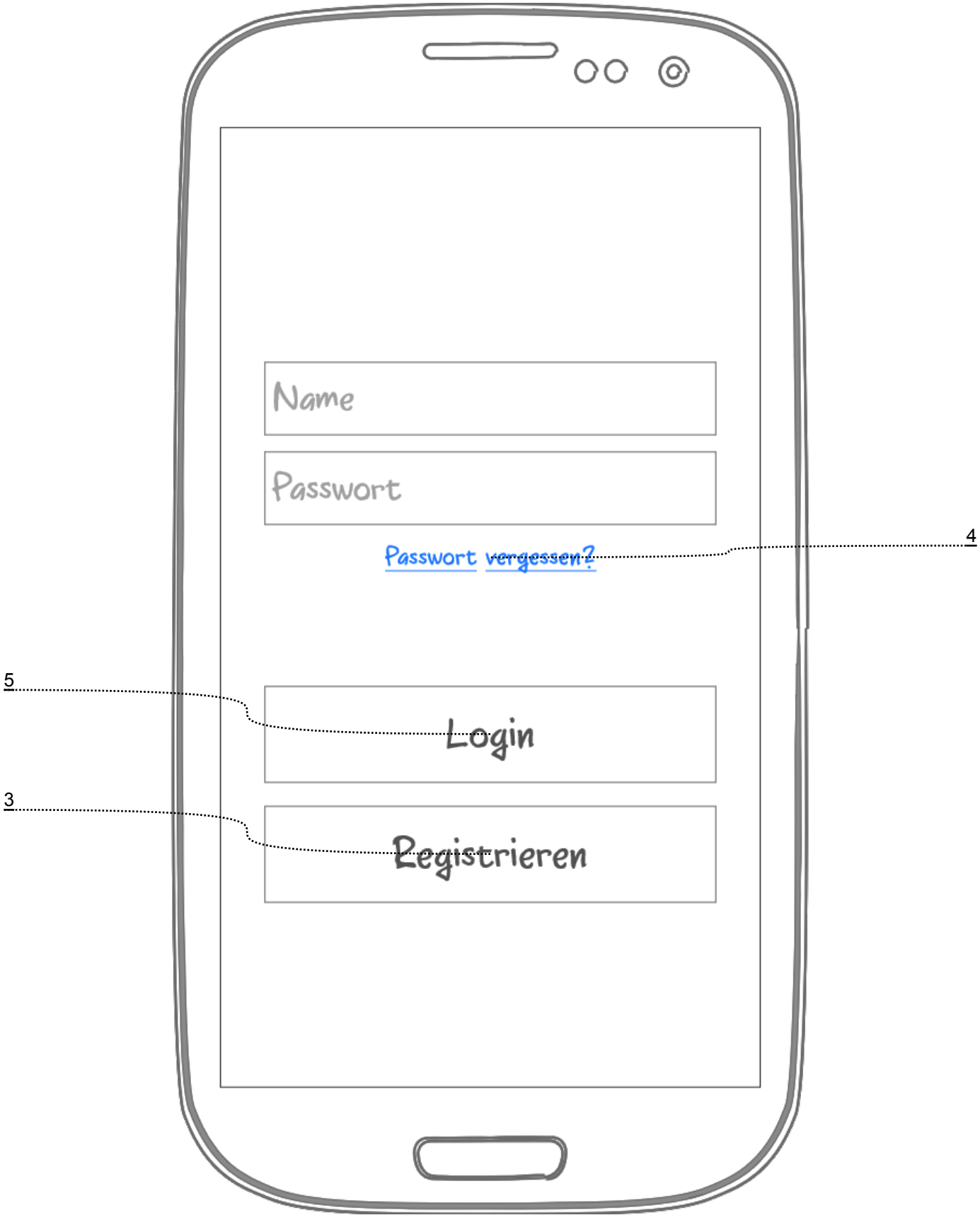
[7] `https://ninjamock.com/`

1 - Ladebildschirm



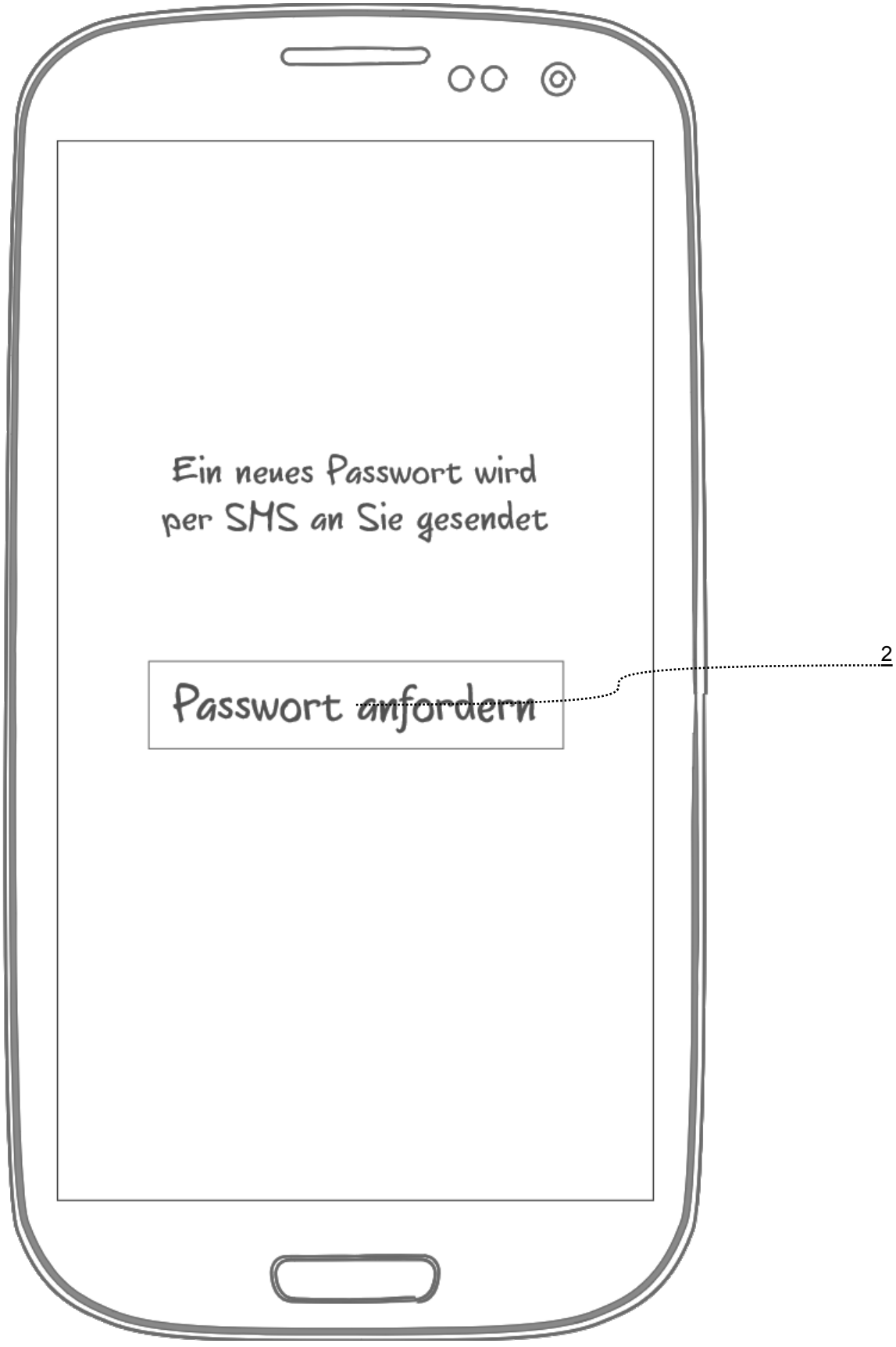
2

2 - Login

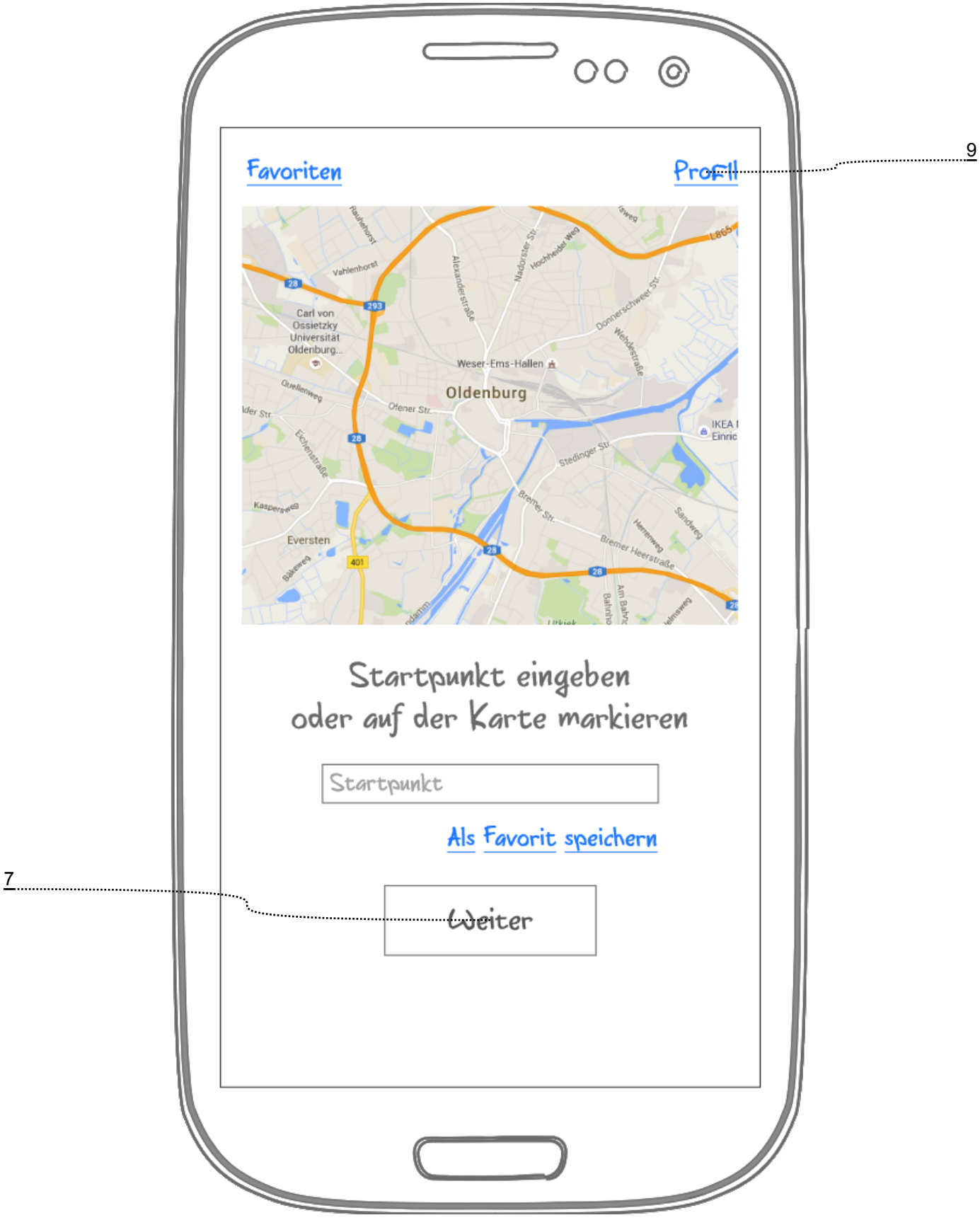


3 - Registrierung

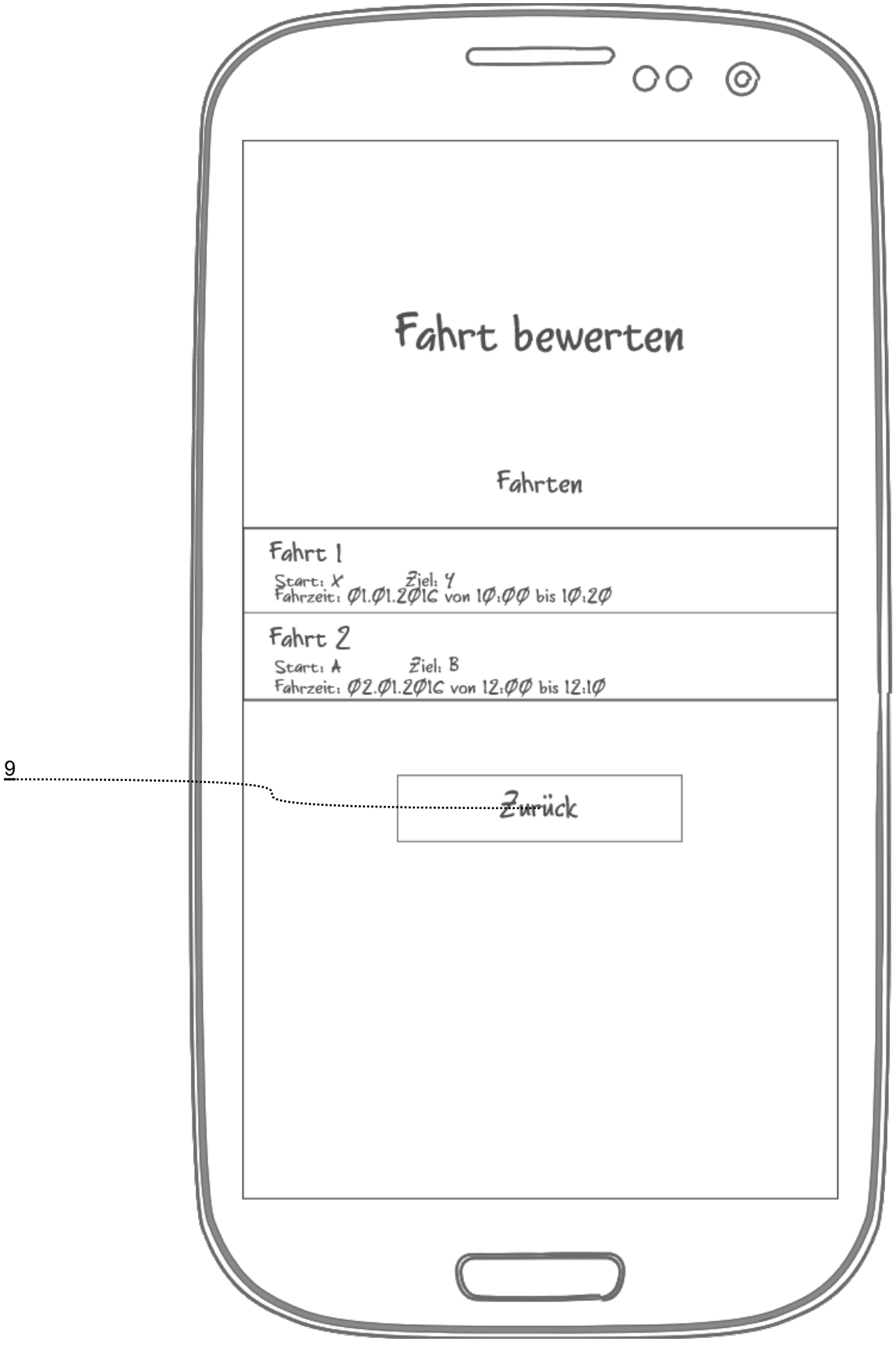
4 - Passwort vergessen



5 - Starteingabe

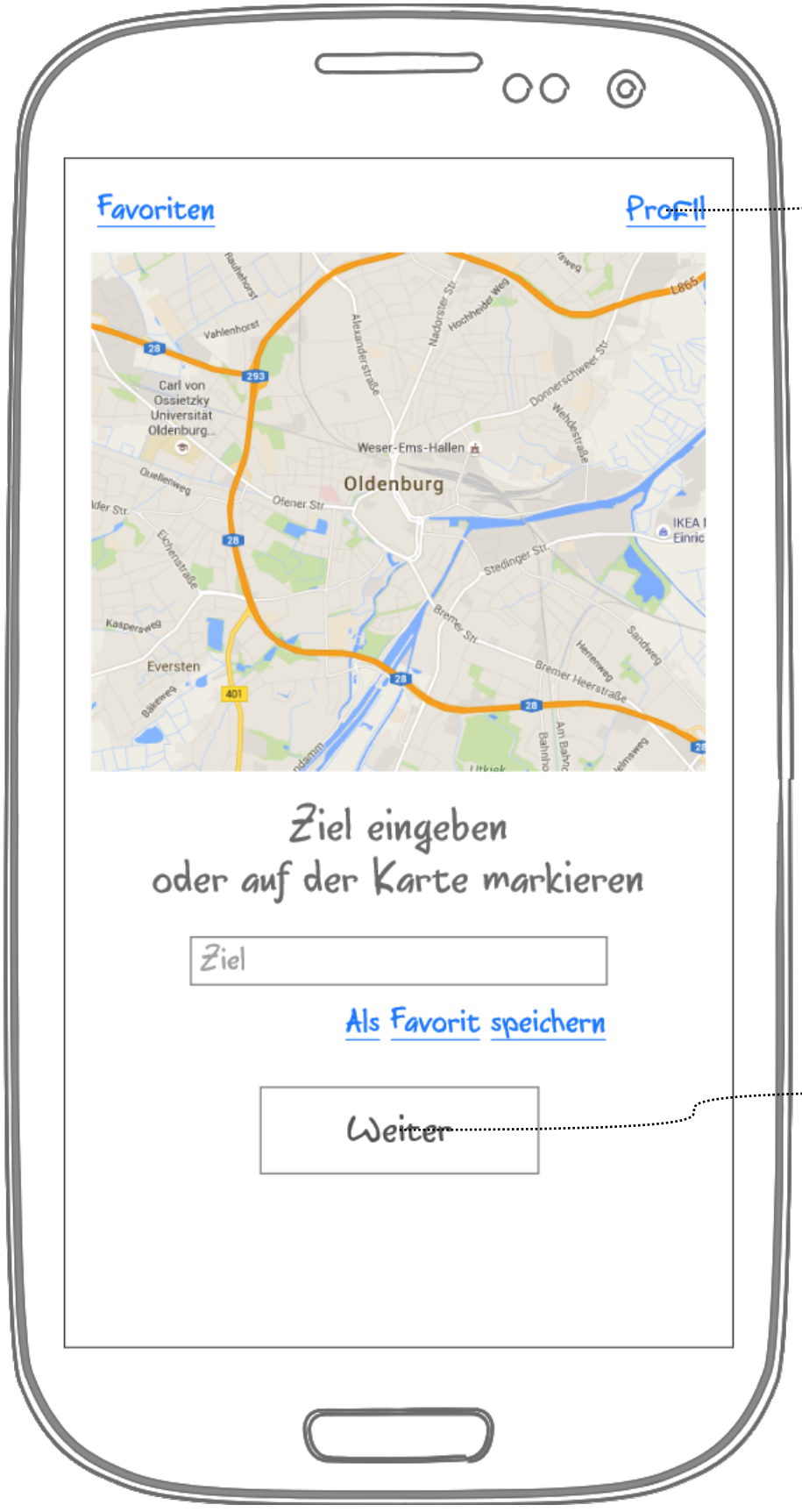


6 - Fahrt bewerten



9

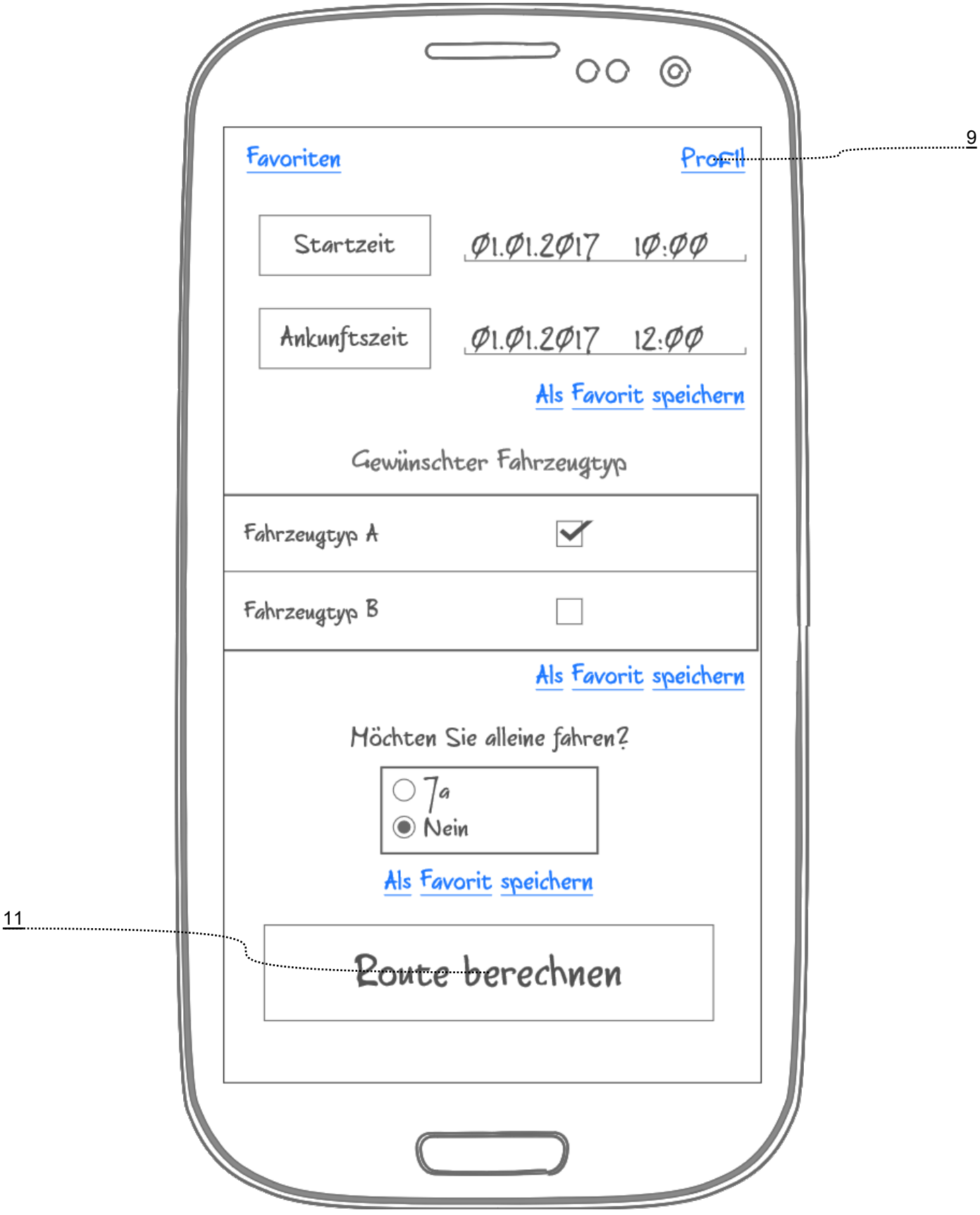
7 - Zieleingabe



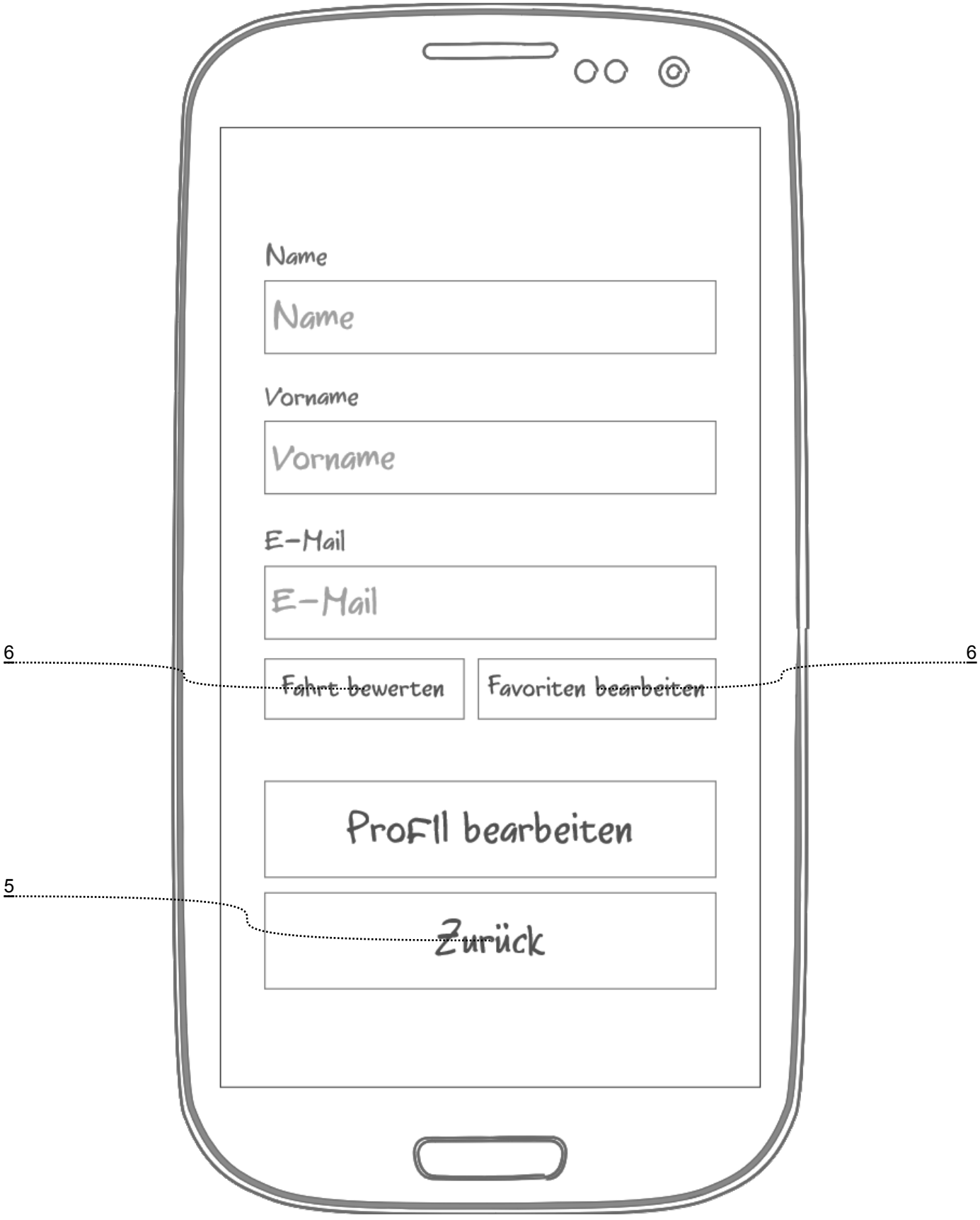
9

8

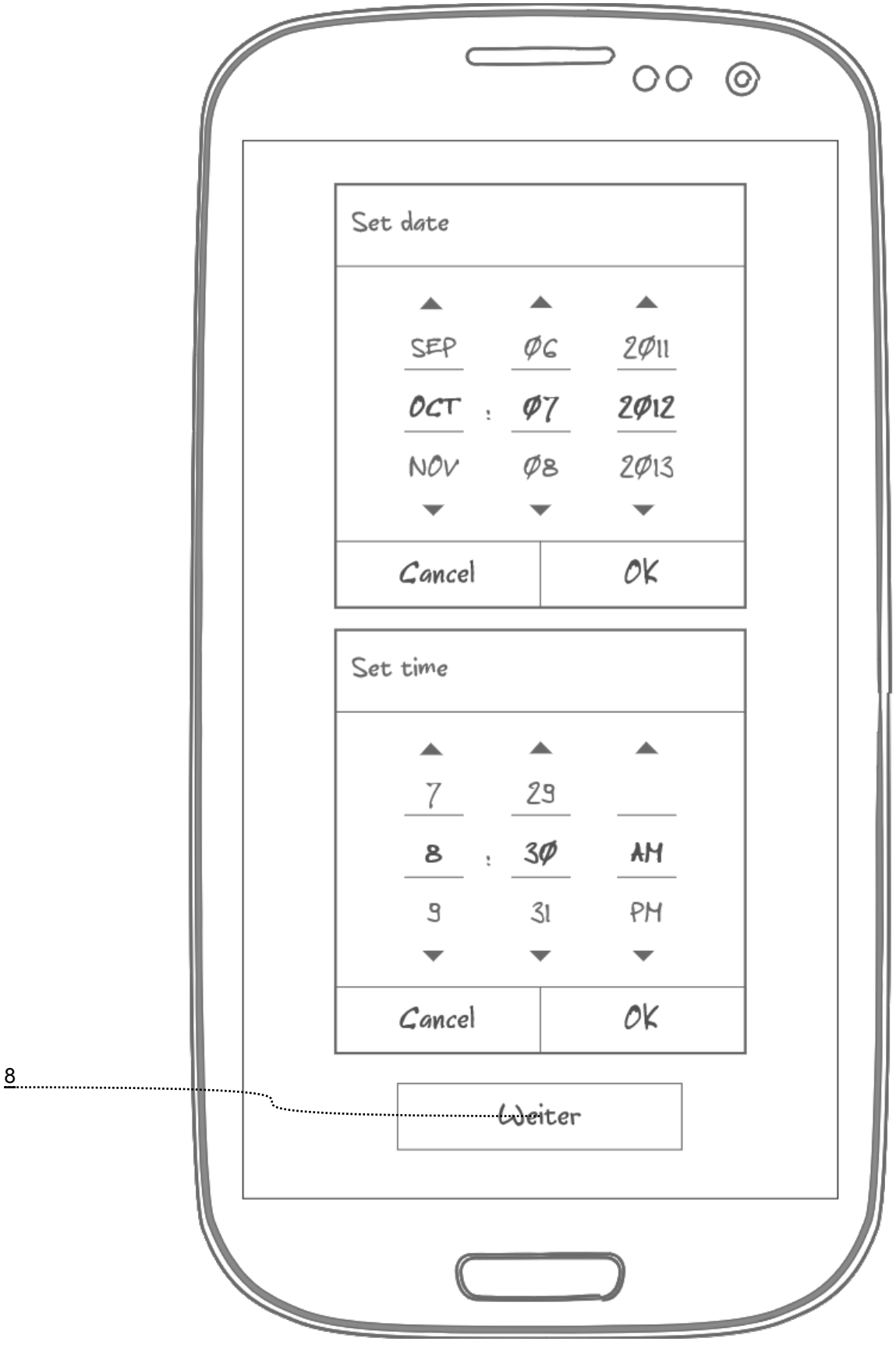
8 - Weitere Informationen



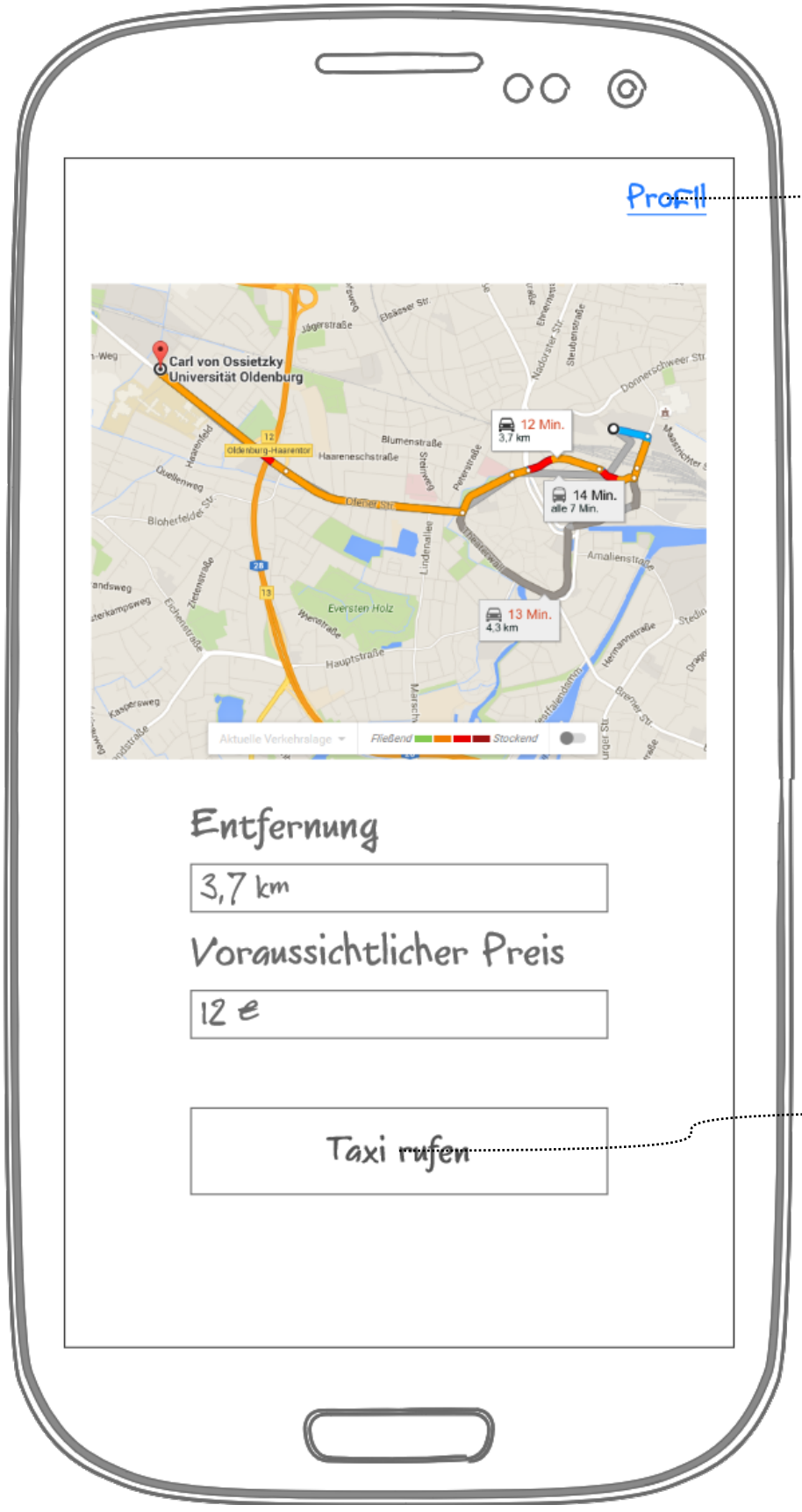
9 - Profil


Name
Name
Vorname
Vorname
E-Mail
E-Mail
Fahrt bewerten
Favoriten bearbeiten
Profil bearbeiten
Zurück
6
6
5

10 - Zeitangabe



11 - Information



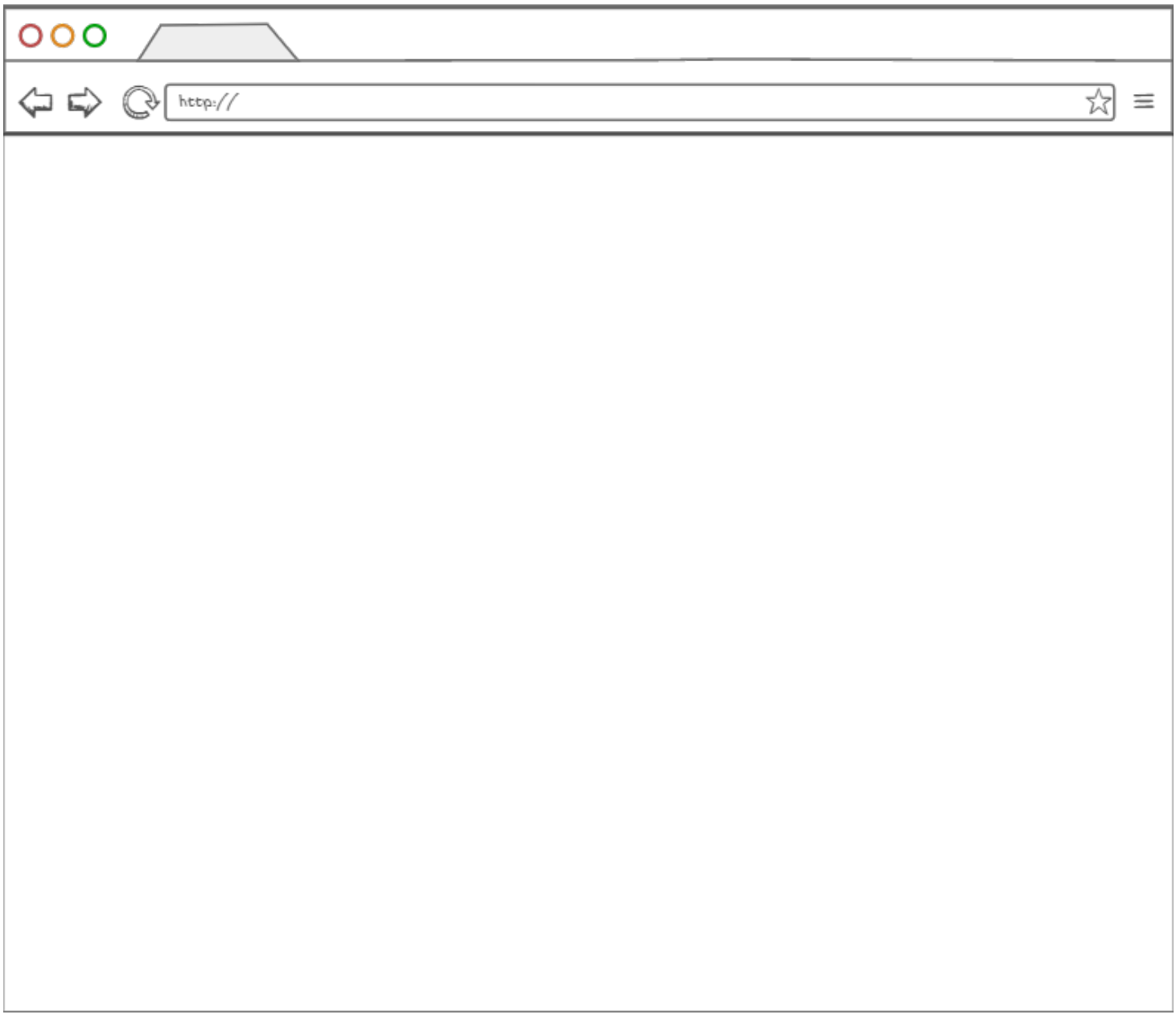
9

12

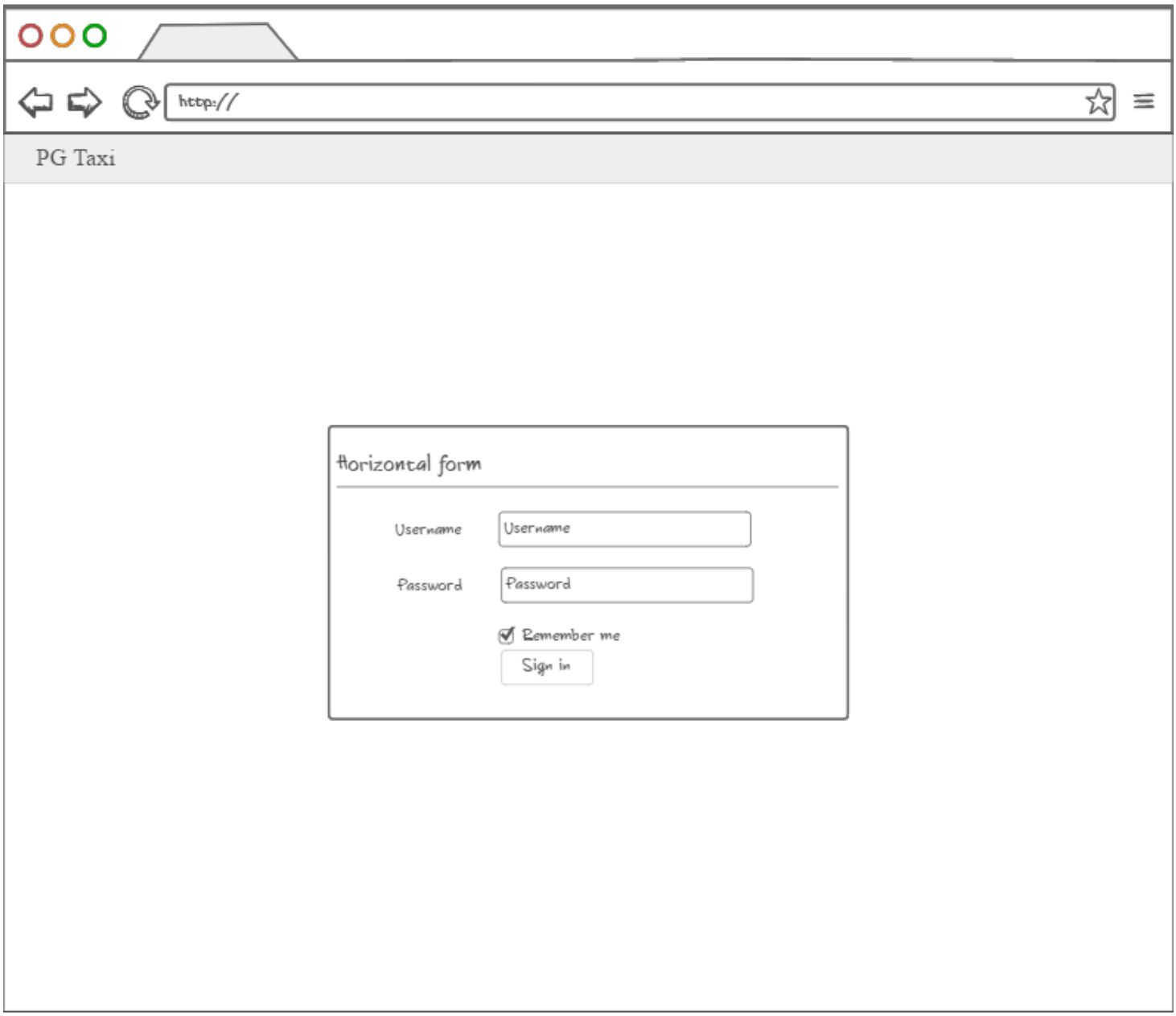
12 - Taxiruf

9

2



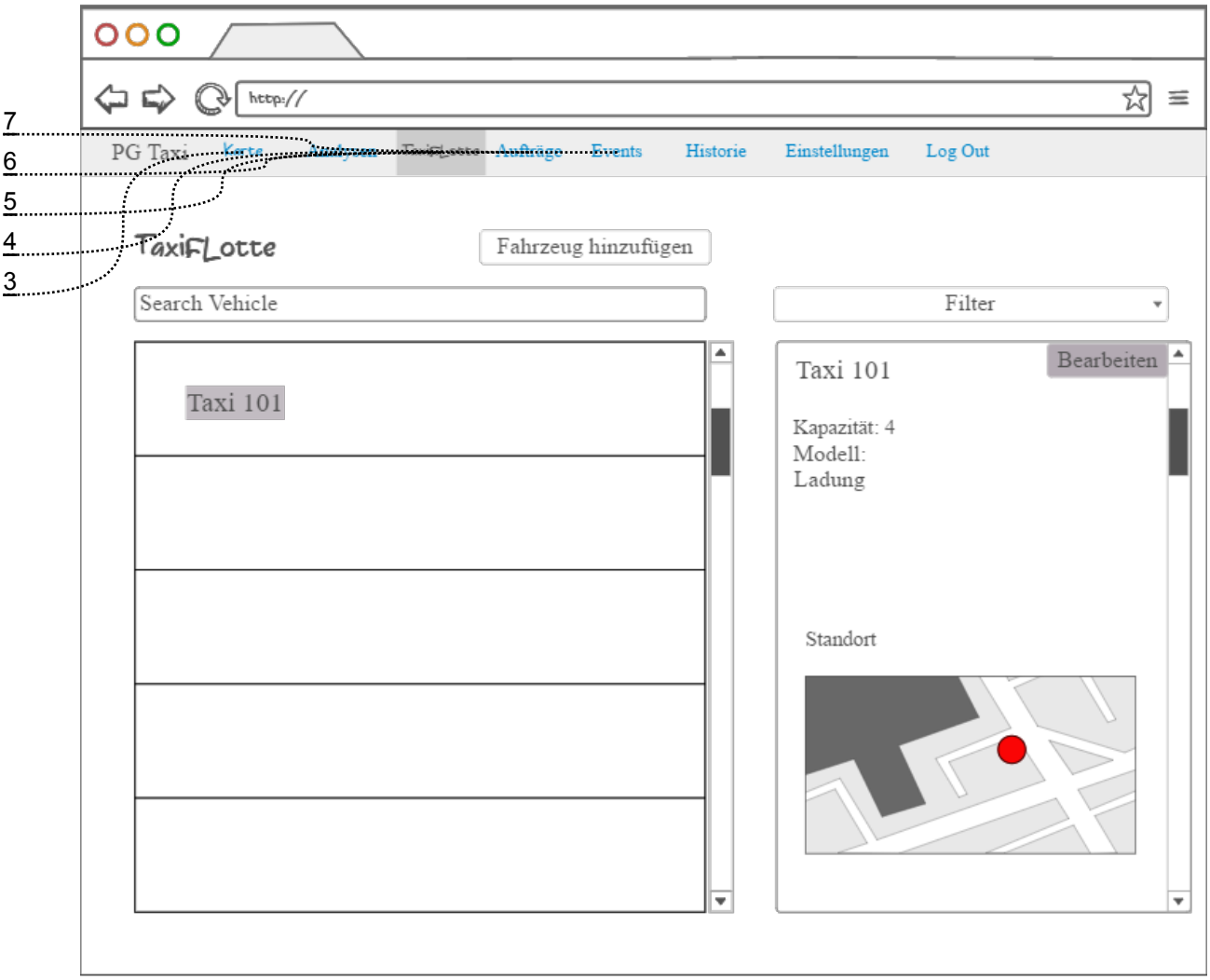
2 - Default group - Log In



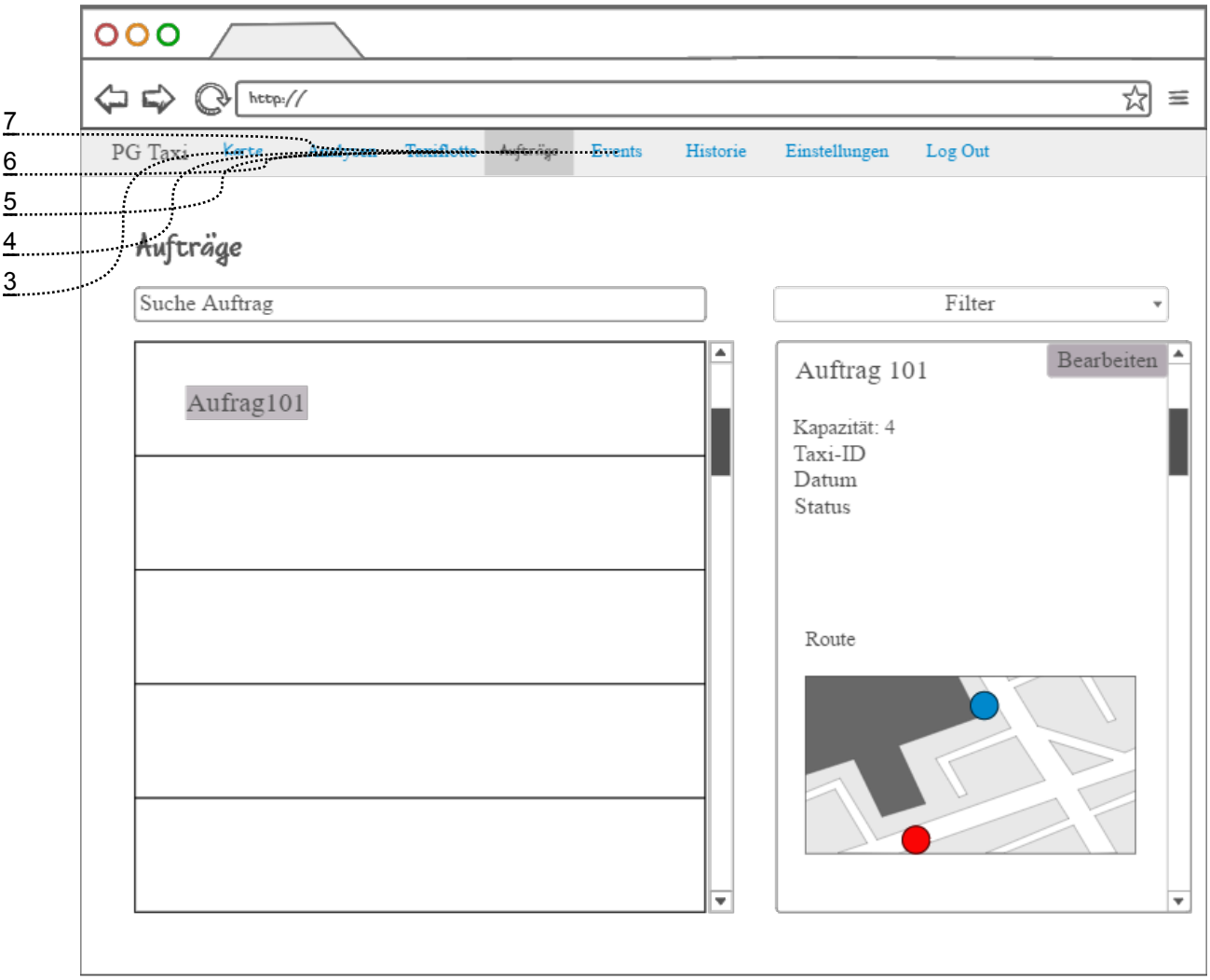
3 - Default group - Karte

4 - Default group - Analysen

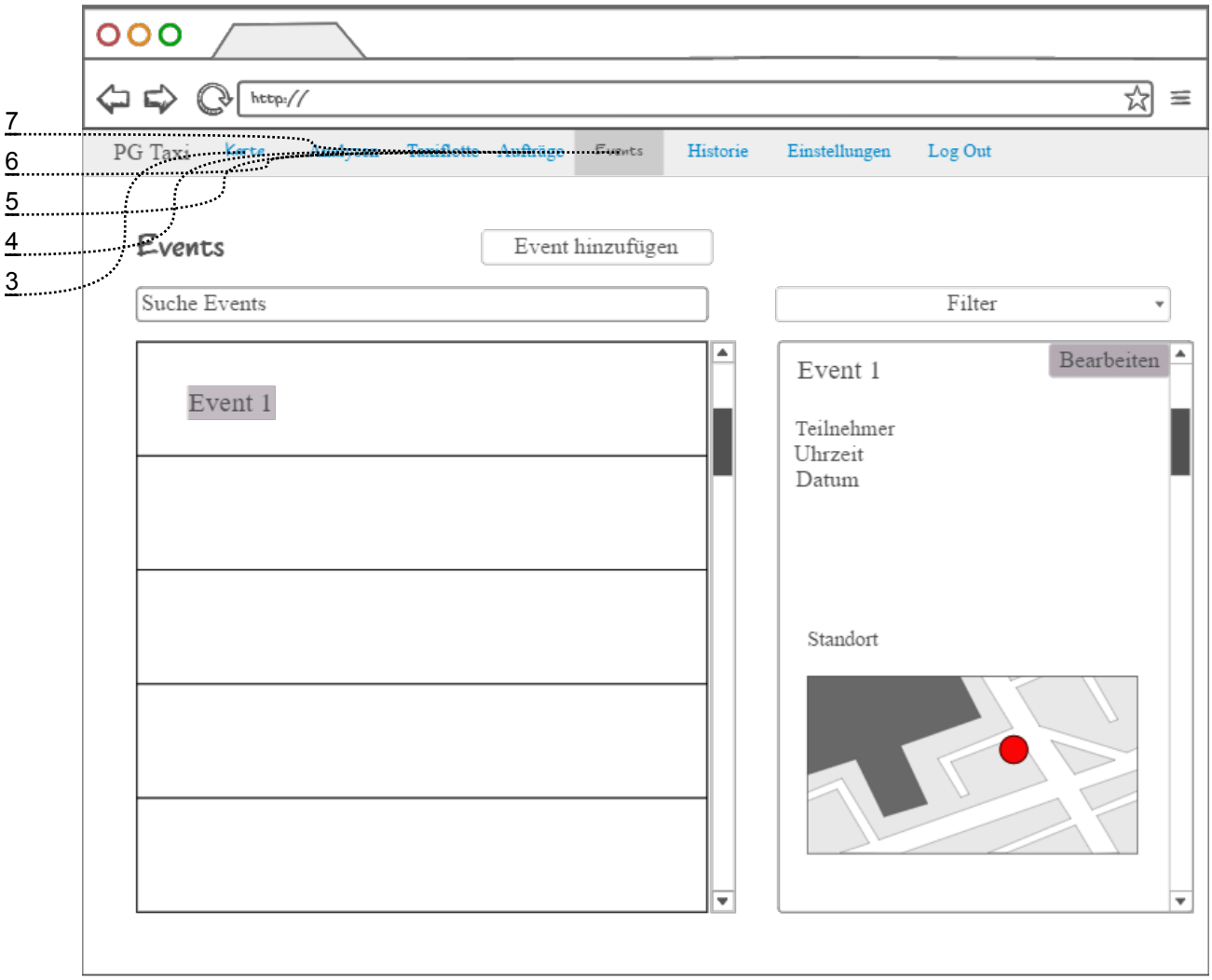
5 - Default group - Taxiflotte



6 - Default group - Aufträge



7 - Default group - Events



6.2 Virtuelle Maschine - Server

Um die Dienste auf der Serverseite bereitzustellen, wurde eine Virtuelle Maschine aus dem OFFIS für uns eingerichtet. Die virtuelle Maschine ist unter der IP 134.106.56.4 erreichbar. Man hat sich innerhalb der Projektgruppe für das Betriebssystem Windows Server 2012 in der 64-Bit Version R2[8] entschieden, aufgrund der breiten Kompatibilität für Software und der komfortablen Benutzeroberfläche. Die Virtuelle Maschine betreibt folgende Software:

- **Apache Webserver 2.4** ist ein quelloffener und der meistbenutzte Webserver im Internet weltweit.wurde zum Betrieb des webbasierten Dashboards installiert. Der Apache Webserver kann über den Server-Port 8080 erreicht werden

- **PHP7** steht für Hypertext Processor und ist eine Skriptsprache, die zur Erstellung von dynamischen Webanwendungen genutzt wird und für unser Dashboard, welches unter AngularJS entwickelt wurde, benötigt wird.

- **PHPMyAdmin** ist eine Open-Source-Webanwendung, welche eine grafische Benutzeroberfläche in einem Webbrowser zur Administration von MySQL-Datenbanken bereitstellt. Hierdurch wir eine webbasierte und komfortable Alternative zur MySQL Workbench geboten. Die Weboberfläche kann über die Adresse 134.106.56.4:8080/phpmyadmin erreicht werden.

- **MySQL Server** ist eines der weltweit verbreitetsten relationalen Datenbankverwaltungssysteme und ist eine Open-Source-Software. MySQL dient zu persistenten Datenspeicherung, auf die Dashboard und App angewiesen sind. item **MySQL Workbench** ist eine Software zur umfangreichen Datenbankverwaltung, die Modellierungswerkzeuge und Datenbankdesign integriert. Die Software kann unter GPL unter den gängigen Betriebssystemen Windows, macOS und Linux kostenfrei genutzt werden.

- Das komplette**Odysseus** Projekt wurde in Verbindung mit SVN auf dem Server hinterlegt. Der **Odysseus Server** beinhaltet das Basis-System. Die

[8]Microsoft Windows Server 2012 ist ein Betriebssystem der Windows-Serie und die Serverversion von Windows 8

Verarbeitungsmechanismen, Anfrageschnittstelle sowie die Benutzerverwaltung werden mit Odysseus Server mitgeliefert und sind die Grundlage für die Ausführung von Odysseus.

- **SUMO** in Verbindung mit **TraCI (Traffic Call Interface)** ist eine Open-Source Verkehrssimulation, die auf der virtuellen Maschine läuft und über die freigeschalteten Server-Ports 5000-5010 die Kommandos, die die TraCI-Schnittstelle überträgt, verarbeitet.

- **Python 2.7, Visual C++ Redistributable 2013 und Java RE 1.8** wurden mit den zugehörigen Laufzeitkomponenten installiert zur Ausführung von Java, C++ Oder Python Anwendungen

- **TortoiseSVN** wurde installiert, um unser Projekt auf dem Server zentral zu hinterlegen.

6.3 Odysseus

Odysseus stellt als Datenstrommanagementsystem eine zentrale Komponente des Systems dar. Zur Implementierung der benötigten Funktionen wurden von der Projektgruppe mehrere neue Operatoren erzeugt und verwendet. Diese werden im Folgenden aufgelistet und kurz beschrieben.

- Mit **CalcRoute** wird aus den Taxidaten und den Daten der eintreffenden Aufträge mit Hilfe von GraphHopper und JSprit eine optimale Route berechnet. Dabei wird zudem entschieden welches Taxi welchem Auftrag zugewiesen wird. Der entscheidende Faktor ist dabei die Entfernung vom Kunden zum nächstgelegenen Taxi. Der Output des Operators enthält die ursprünglichen Auftragsdaten, die berechnete Route, die ID des zugewiesenen Taxis sowie weitere Informationen wie die voraussichtliche Zeit bis zur Ankunft beim Kunden bzw. am Ziel sowie Fahrtkosten. Diese Daten werden zur Verkehrssimulation Sumo, zum Operator SendDataToApp sowie zum Operator SendDataToGoogleServer weitergeleitet.

- **TaxiHandler** dient dem Hinzufügen von Taxis zur Taxiflotte, dem Aktualisieren der Attribute von Taxis sowie dem Entfernen von entsprechend markierten Taxis.

- **TrafficHandler** verarbeitet Verkehrsevents, indem die erlaubte Geschwindigkeit auf den entsprechenden Edges im Graphen angepasst wird.

- **FacebookDataHandler** konvertiert die JSON Datei mit den Events aus Facebook und sendet sie für Odysseus leserlich zurück. Die JSON Datei stammt von einem externen Tool, das Facebook Events in einer bestimmten Region abrufen kann.

- **EventHandling** verarbeitet die vom FacebookDataHandler konvertierten Events. Dazu werden die Events auf die Anzahl der Zusagen analysiert und dann eine bestimmte Anzahl von Taxis zu diesem Event zugeordnet. Der Output besteht aus einem Befehl für Sumo, der die ausgewählten Taxis zu dem Event sendet.

- **RemoveEventSchedule** entfernt Event Schedules von Taxis, wenn entsprechende Events abgelaufen sind.

- **SendDataToApp** erhält den Output von CalcRoute und sendet diesen mit einer Verzögerung zur Android Applikation. Die Verzögerung ist notwendig, damit Firebase eine Nachricht an die Applikation senden und die Applikation daraufhin eine TCP-Connection öffnen kann, bevor der Output von CalcRoute an die Applikation gesendet wird.

- **SendDataToGoogleServer** wurde angelegt, um Firebase Cloud Messaging zu integrieren. Findet eine Aktualisierung der Route statt, sendet dieser Operator durch ein POST-Request eine JSON an den Google Server (https://fcm.googleapis.com/fcm/send). Der Google Server benachrichtigt daraufhin die App auf dem Smartphone des der Route zugehörigen Kunden das seine Route aktualisiert wurde.

- **DynamicLocationPlanning** empfängt die ankommenden Aufträge und ordnet sie den einzelnen Quarters zu. Er ist dazu da die dynamische Standortplanung der Taxis vorzunehmen indem er erhöhte Auftragsaufkommen in den Quartern von Oldenburg erkennt und ein Taxi zu deren Zentrum schickt.

- **AddChargingStation** liest Ladestationen aus der Datenbank und fügt sie in eine ArrayListe ein.

- **CheckTaxiRoute** verhindert, dass nicht mehr aktuelle Routen an Sumo gesendet und dabei aktuelle Routen überschrieben werden. Dieser Fall könnte eintreten, da Routen durch den Join mit einem Timer verzögert an Sumo gesendet werden. Ohne diese Verzögerung würden Taxis immer sofort starten, obwohl die Startzeit des Auftrags in der Zukunft liegt.

6.4 Routenberechnung und -optimierung

Zur intelligenten Planung von Taxi-Fahrten müssen Routen berechnet und optimiert werden. Die Routenberechnung soll mit der Routing-Bibliothek von *GraphHopper*[9] durchgeführt werden und die Optimierung der Routen durch *jsprit*[10].
Die Routing-API ist ein von der GraphHopper GmbH entwickeltes Werkzeug, um eine kürzeste Route zwischen zwei oder mehr Punkten zu berechnen. Die Routing-API ist Teil der Directions-Api von *GraphHopper* und kann als Webdienst eingeschränkt (bezüglich der Anzahl an Anfragen innerhalb eines Zeitfensters) kostenfrei genutzt werden[11].
Für die Projektgruppe muss das Dial-a-Ride-Problem[12] gelöst werden. Hierfür müssen zuerst die Kosten für alle Wege zwischen den Abhol- und Ablieferpunkten bekannt sein. Die Routing-Bibliothek, die auf eigenen Servern kostenlos genutzt werden kann, kann einen Teil dieses Problems bearbeiten. Hierfür muss zwischen je zwei Punkten und in beiden Richtungen die kürzeste Route berechnet werden. Dies ist für den Fall von nur in eine Richtung befahrbare Wege notwendig. Anschließend muss die optimale Reihenfolge für die Fahrt der Taxis zu allen Punkten berechnet werden. Im Gegensatz zur Routing-Bibliothek ist die Route-Optimization-Bibliothek – die dieses Problem lösen könnte – nicht kostenfrei verfügbar. Alle möglichen Kombination (Abfolge von besuchten Knoten) in der Routing-Bibliothek zu berechnen, ist aufgrund der vielen möglichen Reihenfolgen (n!) kein praktikabler Ansatz.
Dies lässt sich damit begründen, dass das Reihenfolge-Problem schnell wächst und schon für eine kleine Anzahl von Kunden nicht mehr exakt berechenbar ist. Zur Optimierung von Routen kann aber das frei verfügbare *jsprit* verwendet werden, das auch keine exakte Lösung für eine beliebig hohe Anzahl von Kundenanfragen bietet, aber mittels Meta-Heuristiken eine gute Lösung berechnen kann. Allerdings muss hierfür ein gerichteter Graph mit Kantengewichten (Kosten oder Fahrtzeit) aufgebaut werden, da *jsprit* standardmäßig nur mit euklidischen Distanzen zwischen den Punkten auf einem 2D-Gitter arbeitet. Aus diesem Grund soll eine eigene Komponente entwickelt werden, die diese Schritte ausführt und so die Lösung des

[9] `https://github.com/graphhopper/graphhopper`
[10] `https://github.com/graphhopper/jsprit`
[11] `https://graphhopper.com/enterprise`
[12] `http://www.itu.dk/people/pagh/CAOS/DARP.pdf`

DARPs in *jsprit* ermöglicht. Eine solche Lösung (Reihenfolge der Knotenbesuche) soll anschließend wieder in GraphHopper überführt werden, um dort die kürzeste Route zwischen diesen Punkten exakt auf Open-Street-Map-Daten zu berechnen und anschließend die Wegpunkte für die Verkehrssimulation auszugeben.
Die maximale Anzahl der in GraphHopper zu berechnenden Kosten für jede Kante entspricht der doppelten Anzahl an Kanten in einem vollständigen ungerichteten Graphen, da ein Weg von A nach B und B nach A zwischen je zwei Knoten berechnet werden muss. Für einen ungerichteten vollständigen Graphen gilt bezüglich der Kantenanzahl: $|K| = (n/2)(n-1)$, wobei n die Anzahl der Knoten ist[13]. Im Regelfall dürfte ein Weg in beide Richtungen befahrbar sein, aber andere Fälle (z. B. durch Einbahnstraßen) müssen auch betrachtet werden. Zur Vereinfachung wird in der Phase bis zum ersten Prototypen jedoch davon ausgegangen, dass eine Kante in beide Richtungen befahrbar ist. In Abbildung 14 ist ein gerichteter Graph zu erkennen, bei dem für alle Kanten gilt, dass die Strecke in beide Richtungen befahrbar ist.

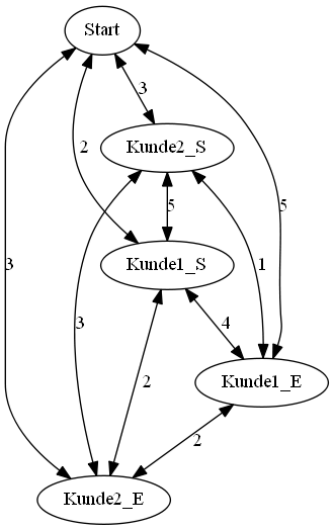


Abbildung 14: Vollständiger gerichteter Graph

[13]Vgl. Skript zur Veranstaltung: Algorithmische Graphentheorie, Elke Wilkeit.

6.4.1 Ausgabe der Route von GraphHopper

Die Ausgabe der Route bzw. Wegpunkte erfolgt wahlweise im JSON-Format (JavaScript Object Notation) oder GPX-Format (GPS Exchange). [14] Das GPX-Format dient der Speicherung von Geodaten (GPS-Daten) basierend auf dem XML-Standard. Das GPX-Format ist offen und lizenzfrei, kann also kostenfrei verwendet werden, um den Austausch von Geodaten zwischen verschiedenen Programmen zu ermöglichen. [15] Die Angabe des gewünschten Formats erfolgt über Parameter (type=*json* oder type=*gpx*).

Die JSON-Ausgabe enthält unter anderem Informationen zu möglichen Routen und Wegpunkten auf der Route. Ein Ausschnitt aus der Ausgabe im JSON-Format:

JSON Attribut	**Beschreibung**
paths[0].points	Koordination der Route
paths[0].points_encoded	*false*, enthält die geo json Informationen [lon,lat,elevation] der Route *true*, enthält die Geo-Koordination als String
paths[0].bbox	Begrenzung der Route [minLon, minLat, maxLon, maxLat]
paths[0].instructions	Textuelle Instruktionen über die Route Befindet sich in Entwicklung und kann fehlerhafte Informationen enthalten
paths[0].instructions[0].text	Anweisung über den nächsten Handlungsschritt, um der Route zu folgen.

[14] `https://graphhopper.com/api/1/docs/routing/output`
[15] `http://www.topografix.com/gpx.asp`

Neben der Ausgabe der Route im JSON-Format können Routen bzw. Wegpunkte auch im GPX-Format ausgeben werden, um den Datenaustausch mit anderen Programmen zu ermöglichen. Dabei wird das 1.1.-Schema benutzt.[16]

Das Problem der Zuordnung von GPX-Punkten auf digitale Karten ist auch bei GraphHopper vorhanden und kann nur über Umwege verbessert werden. Dazu kann das *Map Matching* angewendet werden, bei dem GPX-Punkte einer digitalen Karte zugeordnet werden.[17] Inzwischen existiert eine Open-Source-Software, die diese Aufgabe basierend auf GraphHopper übernimmt.[18]

Für die Zwecke der Projektgruppe wird zwischen zwei Arten von Routen mit je unterschiedlichen Empfängern unterschieden. Zum einen muss eine Route an die App geschickt werden, die von der Abholposition des Kunden bis zu dessen Ablieferposition verläuft und zum anderen muss die Route eines Taxis an die Verkehrssimulation übergeben werden, die von dessen Startposition zu allen zugewiesenen Punkten verläuft. Je nach interner Auftragszuweisung und Routenberechnung kann eine solche Route auch mehrere Zwischenpunkte enthalten, bei denen Kunden abgeholt bzw. abgeliefert werden. Übergeben werden Strings innerhalb einer ArrayList, die die Geokoordinaten der Route enthalten. Zwischenpunkte (Abhol- bzw. Ablieferpunkte) werden ebenfalls so übergeben und zum Output an die App bzw. Verkehrssimulation hinzugefügt.

6.4.2 Routing und Optimierung auf eigenem Server

Damit die Route und im späteren Verlauf auch die Optimierung auf dem eigenen Server durchgeführt werden kann, sind die entsprechenden Klassenbibliotheken (mit ihren Abhängigkeiten) in das eigene Projekt auf dem Server hinzugefügt worden. Dafür wurden die Klassenbibliotheken manuell in das *lib*-Verzeichnis des Projektes **.pgtaxi* geladen und im Projekt dem *Classpath* hinzugefügt. Anschließend sind zwei neue Klassen für den ersten Test beider Funktionen (Routenplanung und -optimierung) erstellt worden. Beide Funktionen konnten erfolgreich getestet werden und stehen somit auf dem Server der Projektgruppe zur Verfügung.

[16] `http://www.topografix.com/gpx/1/1/`

[17] `https://graphhopper.com/blog/2014/07/28/digitalizing-gpx-points-or-how-to-track-vehicles-with-graphhopper/`

[18] `https://github.com/karussell/map-matching-gf`

6.4.3 Darstellung der Karte

Zur Darstellung eine interaktiven Karte auf einem mobilen Endgerät (Android) und über das Dashboard kann *Leaflet*[19] verwendet werden. Leaflet ist eine führende open-source JavaScipt Bibliothek, mit der interaktive Karten erstellt werden können und zudem eine gut dukumentierte API[20] besitzt. Das Kartenmaterial ist in Leaflet nicht enhalten, es können aber unterschiedliche Anbieter (z. B. OpenStreetMap, Google Maps, Bing Maps) eingebunden werden. Dabei sind teilweise zusätzliche Plug-Ins notwendig sind und die entsprechenden Nutzungsbedingungen der Anbieter des Kartenmaterials müssen berücksichtigt werden. Am besten dokumentiert ist die Einbindung von OpenStreetMap-Daten. In der Projektgruppe wird bereits *OpenStreetMap* bei der Routenberechnung über GraphHopper verwendet und die Daten (Kartenmaterial von Oldenburg) liegen schon auf dem Server der Projektgruppe. Google hingegen erlaubt in ihren Nutzungsbedingungen nur die Einbindung des Kartenmaterials mittels der Google Maps API. Mit einem Plugin kann die Google Maps API zwar als zusätzliche Schicht in Leaflet hinzugefügt werden, allerdings führt dies zu Einschränkungen der Funktionalitäten von Leaflet.[21]

[19] `http://leafletjs.com/`

[20] `http://leafletjs.com/reference.html`

[21] `https://github.com/Leaflet/Leaflet/blob/master/FAQ.md`

Bei der Nutzung von Google Maps als Grundlage für das Kartenmaterial müssen zudem die Beschränkungen der benötigten APIs berücksichtigt werden. Die Google Maps Android API ist zwar unbeschränkt, allerdings hat die Places API for Android eine Standardobergrenze von 1.000 kostenfreien Aufrufen pro 24-Stunden-Zeitraum. Die JavaScript API hat eine Standardobergrenze von 25.000 kostenfreien Map-Aufrufen pro 24-Stunden-Zeitraum und die Google Maps Geocoding API eine Standardobergrenze von 2.500 kostenfreien Anfragen pro 24-Stunden-Zeitraum.[22] Aus diesen Gründen ist für die Darstellung der interaktiven Karte im Zusammenspiel mit Leaflet das Kartenmaterial von *OpenStreetMap* eher geeignet. Im Rahmen der Projektgruppe wurde sich – nach Absprache mit den Betreuern – dennoch dafür entschieden, Google Maps zur Darstellung der Karte auf mobilen Endgeräten (Android) zu verwenden. Die Obergrenzen werden wahrscheinlich im Kontext der Projektgruppe nicht erreicht und selbst für eine kommerziele Nutzung könnte entweder die Komponente zur Darstellung der interaktiven Karte auf mobilen Endgeräten gewechselt oder die Standardobergrenze bei Google kostenpflichtig erhöht werden.

Für die Darstellung von Routen auf OSM-Kartenmaterial kann das Plugin *Leaflfet Routing Machine*[23] verwendet werden, welches unter anderem die Routing Engine von GraphHopper für die Berechnungen von kürzesten Routen anhand von Wegpunkten benutzen kann.[24] Leaflet bietet zudem unter anderem die Möglichkeit, eigene Marker und verschiedene Layer zu erzeugen und anzeigen zu lassen, sowie GeoJSON-Objekte[25] zu verarbeiten. Hierfür werden diese GeoJSON-Objekte durch einen GeoJSON-Layer der Map hinzugefügt.

[22] `https://developers.google.com/maps/pricing-and-plans`
[23] `http://leafletjs.com/plugins.html/routing`
[24] `http://www.liedman.net/leaflet-routing-machine/`
[25] `http://geojson.org/`

6.4.4 Aufbau des Graphen in GraphHopper

Ein Straßennetz besteht aus Straßen und Kreuzungen/Abbiegungen, die durch einen Graphen (Kanten, Knoten und Verbindungen) als Datenstruktur in GraphHopper repräsentiert werden. Dabei stehen Kanten für Straßen und Knoten für Kreuzungen/Abbiegungen. Eine Kante verbindet zwei Knoten, sodass eine Straße bzw. ein Straßenabschnitt (bis zur nächsten Kreuzung/Abbiegung) jeweils einer solchen Verbindung entspricht. Außerdem können Eigenschaften, wie z. B. die erlaubte Höchstgeschwindigkeit und die Fahrrichtung, zugeordnet werden. Zum Aufbau des Graphen in GraphHopper wird eine neue Instanz der Klasse *GraphBuilder* erzeugt. Innerhalb dieser werden Knoten und Kanten erzeugt und diese anschließend verbunden. Den Kanten können dabei Eigenschaften ("Flags"), wie die Länge der Kante, die Bezeichnung (String), die Fahrtrichtung, sowie eine Geschwindigkeitsbegrenzung (in km/h) je Richtung, zugeordnet werden.[26] Ein einfaches Beispiel eines Graphen könnte wie in Abbildung 3 aussehen. Dabei kann eine Straße aus mehreren Abschnitten (z. B. 0-1 und 1-2) bestehen.

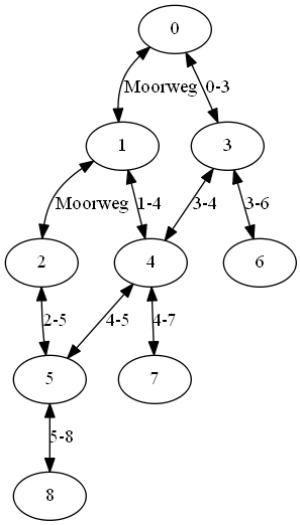


Abbildung 15: Straßennetz als Graph

[26] `https://github.com/graphhopper/graphhopper/blob/master/core/src/main/java/com/graphhopper/routing/util/FlagEncoder.java`

Beispiel: Erzeugen eines Graphen (verkürzte Darstellung)[27]:

```
1  public void newGraph() {
2  EncodingManager carManager = new EncodingManager("CAR");
3          Graph g = new GraphBuilder(carManager).create();
4          NodeAccess na = g.getNodeAccess();
5          na.setNode(0, 1.2, 1.0); na.setNode(1, 1.2, 1.1);
6          na.setNode(2, 1.2, 1.2);
7          ...
8          g.edge(0, 1, 10000, true).setName("Moorweg").
9          setFlags(flagsForSpeed(carManager, 10));
10         g.edge(1, 2, 21000, true).setName("Moorweg").
11         setFlags(flagsForSpeed(carManager, 30));
12         ...
13 }
```

Die Methode *setNode* zum Erstellen von Knotenpunkten erwartet als Übergabeparameter eine ID (int), Breitengrad (double), Längengrad (double) und optional die Erhebung (double) des Punktes.[28] Die Kennzeichnungen ("Flags") der Kanten erlauben Angaben (und entsprechende Abfragen) z. B. über die erlaubte Höchstgeschwindigkeit auf dieser Strecke und die Fahrrichtung.[29]. In dem verkürzten Beispiel hätte die Kante von "0"nach "1"den Namen *Moorweg* und für Autos eine erlaubte Höchstgeschwindigkeit von 10 km/h.

[27] http://androidadb.com/source/github/99/48/994848896/core/src/test/java/com/graphhopper/util/\InstructionListTest.java.html#259

[28] https://github.com/graphhopper/graphhopper/blob/master/core/src/main/java/com/graphhopper\/util/PointAccess.java

[29] https://github.com/graphhopper/graphhopper/blob/master/core/src/main/java/com/graphhopper/\routing/util/FlagEncoder.java

6.4.5 Aktualisieren des Graphen

Zur Berücksichtigung von neuen Verkehrsinformation aus der Verkehrssimulation, müssen Eigenschaften der Kanten des Graphen in *GraphHopper* aktualisiert werden. Dies könnte beispielsweise durch einen Verkehrsstau in der Simulation ausgelöst werden, der zu einer verringerten Geschwindigkeit auf der entsprechenden Kante in *GraphHopper* führt. Bevor die Eigenschaften einer Kante aktualisiert werden können, muss zuerst die Zuordnung zur gewünschten Kante erfolgen. Hierfür werden Längen- und Breitengrad der Straße benötigt, für die Werte (z. B. die Geschwindigkeit oder Fahrtrichtung) aktualisiert werden sollen. Eine entsprechende Methode in *GraphHopper* sucht nun im Graphen - der das Straßennetz von Oldenburg repräsentiert - die nächstliegende Kante, deren Eigenschaften anschließend wie gewünscht aktualisiert werden können.

6.4.6 Aufbau der Kostenmatrix

Standardmäßig nutzt *jSprit* zur Berechnung der Routenoptimierung euklidische Distanzen, die als Einträge in einer Adjazenzmatrix gespeichert sind. Für die Anforderungen in der Projektgruppe reicht dies allerdings nicht aus, da die Einträge die jeweilige Fahrtdauer zwischen zwei Knoten enthalten müssen, um die Routenoptimierung realitätsnah durchführen zu können. Damit die Zuordnung der Fahrzeuge zu den Aufträgen sowie die Reihenfolge der Wegpunkte (Abhol- bzw. Ablieferpunkte) optimiert werden können, müssen die von *GraphHopper* ermittelten Geschwindigkeiten auf den Straßen der Oldenburg-Karte zu einer Kostenmatrix in *jSprit* hinzugefügt werden. Punkte in der Kostenmatrix sind die Fahrzeuge und die Kundenaufträge, die sich in Abhol- und Ablieferpunkte aufteilen. Die Verbindungen zwischen diesen Punkten sind die Einträge in der Kostenmatrix. Eingetragen wird die von *GraphHopper* ermittelte durchschnittliche Fahrtdauer zwischen den jeweiligen Punkten.

Zum Befüllen der Kostenmatrix müssen alle möglichen Verbindungen zwischen den Akteuren (Fahrzeuge und Kunden) mit Kosten verbunden werden. Dabei ist es wichtig, dass es zwischen je zwei Akteuren immer mindestens zwei Kanten gibt. Hierdurch können die Kosten für den Hin- und Rückweg berücksichtigt werden. Die Kosten werden durch Anfragen über die *RouteEngine* von *GraphHopper* ermittelt

und anschließend von der Klasse *RoutingCosts* in eine Kostenmatrix übertragen. Hierfür müssen die Kosten einzeln zu dem *Builder* der Kostenmatrix hinzugefügt werden. Nachdem alle möglichen Verbindungen so mit Kosten belegt worden sind, kann die Matrix endgültig erstellt und von der Klasse *RouteOptimization* genutzt werden. Die Klasse *RouteOptimization* benötigt zur Ermittlung der Transportkosten Information (u. a. Standorte für Start und Ziel und die ID) über die Akteure. Diese Informationen werden jeweils durch den *ClientsManager* und den *VehicleFleetManager* bereitgestellt, welche über ein Interface für den Zugriff verfügen. In Abbildung16 sind die wesentlichen Klassen zur Befüllung der Kostenmatrix dargestellt. Die Klasse *UpdateGraph* stellt Methoden bereit, mit denen u. a. die Kosten auf den Kanten des Graphen, der die Straßenkarte von Oldenburg abbildet, durch die Verkehrssimulation aktualisiert werden können. Bei jeder neuen Kundenanfrage wird die Kostenmatrix neu aufgebaut, so dass die aktualisierten Informationen für nachfolgende Aufträge berücksichtigt werden. Dies kann zur Folge haben, dass Routen sich ändern, weil die Kosten auf einer vorher genutzten Route (z. B. durch eine langsamere Durchflussgeschwindigkeit) höher geworden sind, als die Kosten auf einer alternativen Route zwischen je zwei Punkten.

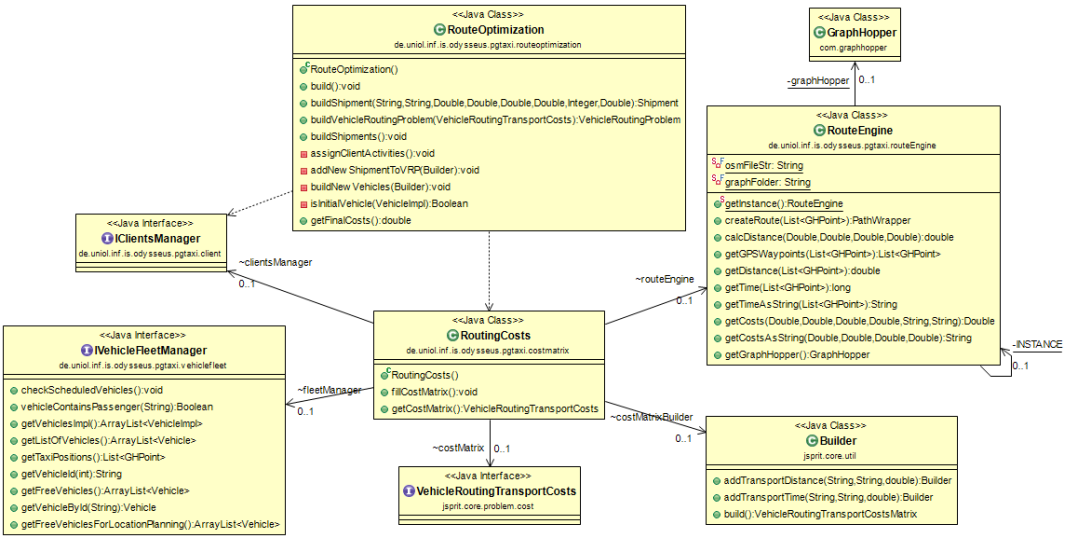


Abbildung 16: Ermittlung der Transportkosten

6.4.7 Dynamische Optimierung

Beim Dial-a-Ride-Problem (DARP) wird davon ausgegangen, dass die zu betrachtenden Aufträge vor der Optimierung bekannt sind und das Optimierungsproblem anhand dieser aufgebaut und gelöst wird. Für Fahrdienste ist aber besonders die dynamische Optimierung von Aufträgen und Routen interessant, die durch das Dynamic-Dial-a-Ride-Problem (DDARP) beschrieben werden kann. Bei dem Dynamic-Dial-a-Ride-Problem können Aufträge nicht nur statisch vorliegen, sondern dynamisch zu bereits bestehenden Lösungen hinzukommen. Dies kann dazu führen, dass sich bereits aus vorherigen Lösungen berechnete Routen dynamisch ändern. Zur Umsetzung der dynamischen Planung werden die zugewiesenen Aufträge (Taxis zu Kunden und Routen) aus vorherigen Lösungen entnommen und beim Aufbau eines neuen Optimierungsproblems als initiale Routen wieder hinzugefügt.[30]. Dies führt auch dazu, dass bereits einmal zugewiesene Aufträge in neuen Iterationen nicht wieder verworfen werden. Würden hingegen nur die aktualisierten Daten (z. B. aktualisierte Taxipositionen, neue Aufträge) für das neue Problem betrachtet, ohne die vorherige Lösung im neuen Optimierungsproblem zu berücksichtigen, könnte es passieren, dass vorher zugewiesene Aufträge in neuen Lösungen nicht mehr berücksichtigt werden. Beispielsweise könnte eine vorher zugewiesene Route in einer neuen Lösung verworfen werden, weil ein neuer Auftrag weniger Kosten verursacht und nicht beide Aufträge (gleichzeitig) bedient werden können. Dies soll in unserem Kontext nicht vorkommen, da bereits einmal zugestimmten Kundenaufträgen - mit entsprechender Rückmeldung beim Kunden - in zukünftigen Lösungen nicht mehr verworfen werden dürfen.

[30] `https://github.com/graphhopper/jsprit/blob/master/jsprit-core/src/main/java/com/graphhopper/jsprit/core/problem/VehicleRoutingProblem.java`

Beispiel zum Vergleich mit bzw. ohne initialer Lösung:

Annahmen:

- Taxi t1 wurde in vorheriger Lösung Kunde k1 zugeordnet. Anschließend kommt ein weiterer Auftrag dazu.
- Beide Aufträge können nicht zusammen (durch gemeinsame Fahrt, Ausnutzung der Zeitintervalle oder Zuordnung eines anderen Taxis) bedient werden.
- Der neue Auftrag ist günstiger (bzw. schneller), als der alte Auftrag.

(i) Neue Lösung ohne initialer Route aus vorheriger Lösung:

Taxi t1 wird Kunde k2 zugeordnet. Die Zuordnung aus der initialen Lösung geht verloren.

(ii) Neue Lösung mit initialer Route aus vorheriger Lösung:

Taxi t1 wird weiterhin Kunde k1 zugeordnet, weil die Zuweisung/Route aus der initialen Lösung bedient werden muss. Der Auftrag von Kunde k2 kann nicht bedient werden.

Beim Hinzufügen von initialen Routen aus der vorherigen Lösung werden diese Lösungen bei der Optimierung durch die Meta-Heuristik von JSprit nicht vernichtet [31], während bei neuen Aufträge weiterhin während der Optimierung verschiedene Lösungen erstellt und vernichtet werden, um eine optimale Lösung zu erhalten. Dass günstigere Aufträge nicht bedient werden, weil vorherige Aufträge diese blockieren ist im Kontext eines Fahrdienstes eine notwendige Voraussetzung, weil zugesagte Aufträge weiter bedient werden müssen und nicht durch neue Aufträge aus der Lösung rausfallen dürfen.

[31] `https://discuss.graphhopper.com/t/resume-previously-planned-solution/729/5`

6.4.8 Personenmanagement

Um für die Berechnung des Dial-A-Ride-Problems durch JSprit die Belgung der einzelnen Taxis zu ermitteln wurde das Personenmanagement eingeführt. Löst ein Auftrag im *CalcRoutePO* die Berechnung einer neuen Route für ein Taxi aus, wird mittels der Methode *routePointsOfVehicle()* des *ActivityManagers* die einzelnen Activities (Pickup und Delivery) der neuen Taxi-Route als *Stopover* im Taxi gespeichert. Ein *Stopover* ist ein Zwischenstopp auf der Route und beinhaltet neben der GPS-Position die Anzahl an Personen, um die sich die Belegung des Taxis ändert und eine Shipment-Id, um die Activity in JSprit zuordnen zu können. Wird nun die Route vom Taxi in SUMO abgefahren, prüft der *TaxiHandlerPO* ob der nächste *Stopover* erreicht wurde, indem er bei den als Tupel aus SUMO ankommenden Taxidaten die Entfernung zum *Stopover* berechnet. Dies geschieht durch die Funktion *updateNumberOfTaxiPassengers()* in der Klasse *UpdateVehicle*, welche die Luftlinie zwischen den beiden GPS-Punkten misst und wenn diese eine definierte Entfernung unterschreitet, die Anzahl der Passagiere im Taxi um den Betrag aus dem *Stopover* aktualisiert. Anschließend wird der *Stopover* für das Taxi gelöscht und die entsprechende Activity aus der Berechnung im JSprit.

6.5 Standortplanung

6.5.1 Standortplanung auf Basis von Veranstaltungen

Mit Hilfe der in Odysseus implementierten Funktionen des Event-Handlings ist es möglich Taxis zu Orten zu schicken, an denen Veranstaltungen stattfinden.
Veranstaltungen können dabei aus folgenden Quellen stammen.

- Veranstaltungsgenerator: Mit Hilfe des Veranstaltungsgenerator können Veranstaltungen erzeugt werden, die direkt vom Event-Handling in Odysseus verarbeitet werden können. Variabel ist dabei die zeitlich Frequenz in der Veranstaltungen erzeugt werden.

- Facebook-Veranstaltungen: Odysseus Veranstaltungen können mit Hilfe einer Konsolenanwendung aus Facebook abgefragt werden. Anders als die Veranstaltungen des Generators werden die Veranstaltungen aus Facebook nicht einzeln gesendet. Daher werden diese mit den Operationen „CONVERTER“, „PROJECT“und „TOTUPLE“auf das gleich Tupel-Format gebracht.

Eine genauere Beschreibung der Funktionsweise und Nutzung befindet sich in den Kapiteln 7 und 8.
Trifft ein Tupel aus den beschrieben Quellen ein wird es vom EventHandlingPO verarbeitet. Zunächst wird es in ein Objekt der implementierten Java-Klasse Event konvertiert. Anschließend wird überprüft, ob die Veranstaltungs-ID neu ist. Ist dies der Fall, wird sie einer Liste hinzugefügt. Wenn nicht, wird die Liste aktualisiert. Nun wird überprüft, welche (der nicht bereits angefahrenen) Veranstaltungen der Liste die größte Veranstaltung ist. Diese Veranstaltung wird nun angefahren. Die benötigte Route wird wie auch in CalcRoutePO berechnet und anschließend an SUMO gesendet. Taxis, die eine Veranstaltung anfahren, werden bis zum Ende der Veranstaltungen markiert. Ist eine Veranstaltungen beendet, entfernt der Operator RemoveEventScheduledPO diese Markierung und ein Taxi kann wieder für das Event-Handling verwendet werden.
Trifft ein Auftrag in Odysseus ein, hat dieser stets eine höhere Priorität als das Event-Handling.

6.5.2 Ladekonzept

Das Ladekonzept soll den Stromverbrauch der Fahrzeugflotte und die notwendigen Ladevorgänge in die Routenplanung und -optimierung einbeziehen. Der Stromverbrauch eines Fahrzeuges wird hierfür in der Verkehrssimulation simuliert. Hierfür berechnet Sumo den Stromverbrauch für ein Fahrzeug auf der gerade abgefahrenen Route. Das Lademanagement hingegen erfolgt in Odysseus. Hierfür wird der Stromverbrauch für alle aktuell eingeplanten Aufträge eines Fahrzeuges berechnet. Damit kann der Ladevorgang vorausschauend eingeplant werden, anstatt nur jeweils den aktuellen Ladestand zu berücksichtigen und dann bei einem akuten Bedarf eine Ladestation anzufahren. So soll verhindert werden, dass aktuell noch abgefahrene Kundenrouten für einen Ladevorgang unterbrochen werden müssten. Dies hätte auch negative Folgen für die Routenplanung und -optimierung, da diese den Ladevorgang nicht mit einbezogen hätte. Damit Ladevorgänge in der Routenplanung und -optimierung berücksichtigt werden können, bietet es sich an, diese als einen internen Auftrag zu behandeln. Dies bietet den Vorteil, dass die Routen zur und von der Ladestation, sowie die dortige Verweildauer für das Aufladen des Akkus, eng mit den Kundenaufträgen gekoppelt sind und in die bisherige Struktur mit eingebunden werden können. So können Ladevorgänge dynamisch mit eingeplant werden, ohne Aufträge bzw. Routen erst im Nachhinein auseinanderziehen zu müssen, um den Ladevorgang einzubeziehen. Vorbild zur Berechnung des Akkuverbrauchs eines Fahrzeuges soll der BMW i3 (Sommer 2016) sein. Der i3 ist ein kompaktes Stadtauto mit vier Sitzen und erfüllt somit die Anforderungen für die Fahrzeugflotte. Der i3 hat eine Reichweite von ca. 200 km im Stadtverkehr, eine Batterie mit 29 kWh und benötigt für einen Schnellladung ca. 39 Minuten. Soll ein Fahrzeug eine Ladestation anfahren, weil ein vordefinierter Grenzwert für den Akkuladestand überschritten worden ist und ein neuer Ladeauftrag eingeplant wurde, wird die nächste Ladestation vom Standort des Fahrzeuges ermittelt. Die Ladestationen stehen in Odysseus und der Datenbank zur Verfügung. Zur Berechnung der Distanz wird die vorhandene *RouteEngine* benutzt, welche die Route vom aktuellen Fahrzeug-Standort zur jeweils nächsten Ladestation anhand der Route auf der Oldenburg-Karte von *OpenStreetMap* berechnet. Die Ladestation, die am nächsten am Fahrzeugstandort liegt, wird anschließend als Zielpunkt des neuen internen Lade-Auftrages gespeichert. Anders als in den Kundenaufträgen

wird dem internen Lade-Auftrag zudem eine Wartezeit am Zielort mitgegeben, die sich an dem Modell zur Schnell-Ladung des BMW i3s orientiert. Dies hat zur Folge, das nachfolgende Aufträge frühestens dann zeitlich eingeplant werden, wenn der Schnelllade-Vorgang abgeschlossen ist. Damit ist gegeben, dass der Ladevorgang für ein Fahrzeug komplett in der Auftragsplanung mit eingeplant wird.

Modell zur Berechnung des Akkuverbrauchs Vorbild zur Berechnung des Akkuverbrauchs eines Fahrzeuges soll der BMW i3 (Sommer 2016) sein. Der i3 ist ein kompaktes Stadtauto mit vier Sitzen und erfüllt somit die Anforderungen für die Fahrzeugflotte zum Transport von Passagieren im Stadtverkehr.

Technische Daten:[32]

- Viersitzer
- Batterie: 29 kWh
- Reichweite: ca. 200 km
- Ladedauer: ca. 39 Min. (80

In Abbildung 17 sind die wesentlichen Klassen für die Erstellung von Ladeaufträgen dargestellt. Der *TaxiHandler-Operator* bekommt regelmäßig von der Verkehrssimulation aktuelle Informationen (z. B. den Akkuladestand eines Fahrzeuges) über die Fahrzeugflotte gesendet und aktualisiert anschließend den *VehicleFleetManager*, der zur Verwaltung der Fahrzeugflotte in Odysseus dient. Im *TaxiHandler-Operator* ist ein Grenzwert für die Akkuladung definiert, die beim Eintreffen eines neuen Tupels überprüft wird. Hierfür wird über die Klasse*EstimatedBatteryDrain* der *virtuelle Akkustand* für jedes Fahrzeug ausgelesen, der den vorausberechneten Akkustand nach der Bedienung aller eingeplanten Aufträge enthält. Es werden also auch für die Zukunft eingeplante Aufträge bei der Berechnung des Akkustandes berücksichtigt, um den Ladeauftrag an geeigneter Stelle einplanen zu können. Ist der Grenzwert unterschritten, wird für das entsprechende Fahrzeug über die Klasse *ChargeVehicle* ein neuer interner Auftrag erzeugt und in *jSprit* eingeplant. Über

[32] `http://www.bmw.com/com/de/newvehicles/i/i3/2016/showroom/range_charging.html`

die Klasse *ChargingStation* wird die anzufahrende Ladestation für den Ladeauftrag festgelegt, in dem die nächste Ladestation - berechnet vom Ziel des vorherigen Auftrags aus - gesucht wird.

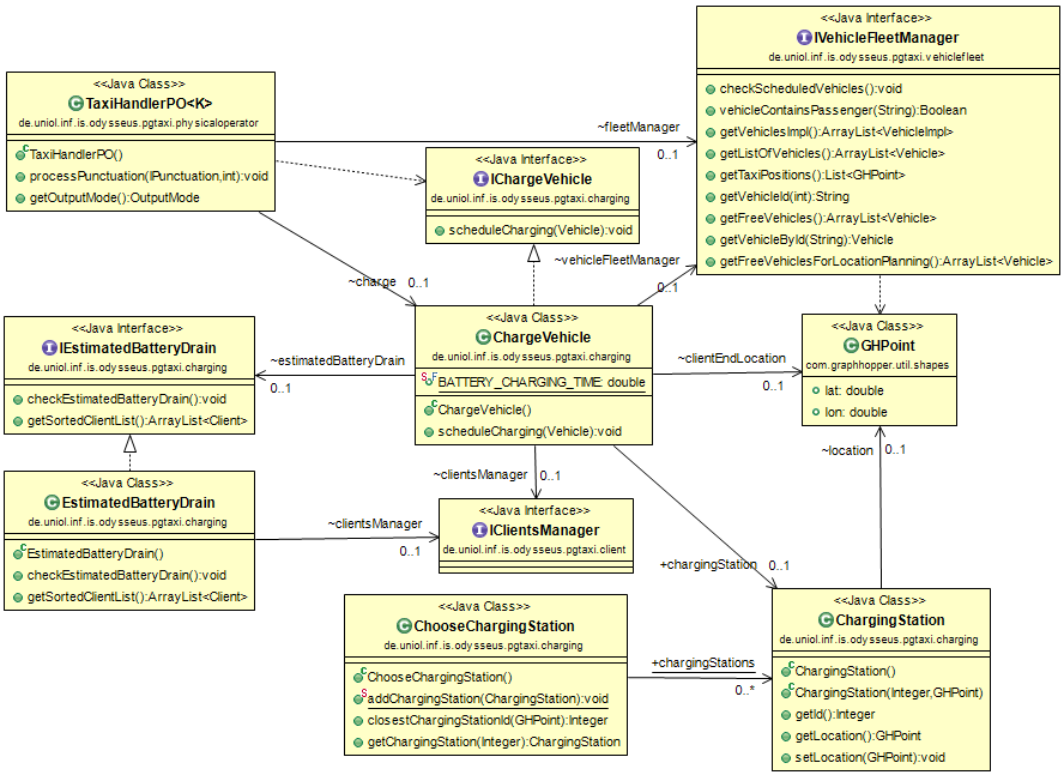


Abbildung 17: Erstellung von Ladeaufträgen

6.5.3 Reaktive Standortplanung

Bei der reaktiven Standortplanung sollen erhöhte Auftragsaufkommen in bestimmten Bereichen in Oldenburg zeitnah erkannt werden. Als Reaktion darauf soll ein Taxi in diesem Bereich positioniert werden, um neu einkommende Aufträge schneller bearbeiten zu können. Odysseus verarbeitet alle einkommenden Aufträge in einem bestimmten Zeitfenster, um so ermitteln zu können wo aktuell Events stattfinden bzw. Menschenmassen sind. Zu diesem Zweck wird Oldenburg in einer Raster von gleich großen Bereichen, den sogenannten *Quarters*, unterteilt (siehe Abbildung 18). Wird in einem dieser Bereich der Grenzwert an Aufträgen in einer definierten Zeitspanne überschritten, so wird erkennt Odysseus dies und schickt

ein Taxi in den Bereich.

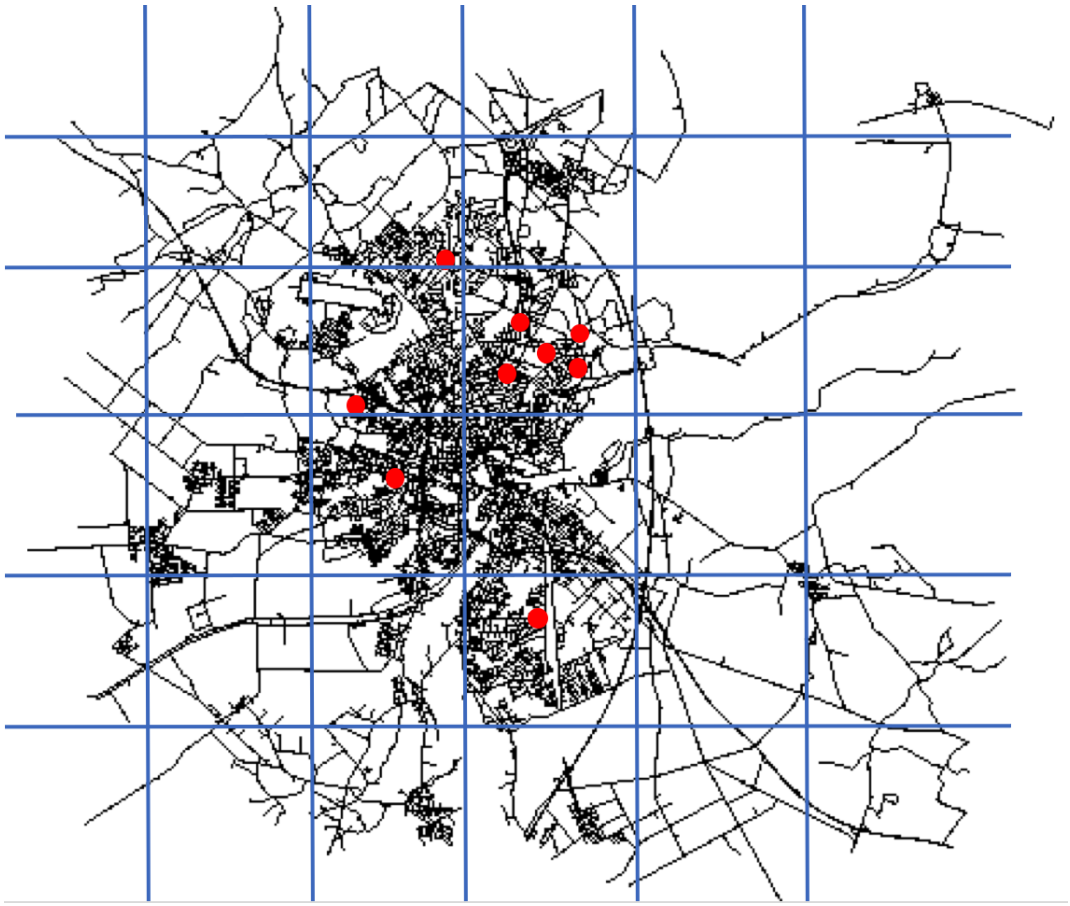


Abbildung 18: Skizzierte Darstellung des Rasters zur reaktiven Standortplanung über Oldenburg

Die bereits beschriebenen *Quarter* werden durch einen sogenannten *QuarterManager* verwaltet. Dieser hält 15 *Quarter*, die den Großteil von Oldenburg, inklusive der wichtigsten Stadtbereiche abdecken. Diese *Quarter* sind aktuell alle gleich groß. Die linke obere Ecke des Rasters liegt bei den Koordinaten *(53.18,8.15)*, die untere rechte Ecke bei den Koordinaten *(52.91050999999998,8.257796000000003)*. Das Raster besteht insgesamt aus drei Spalten mit jeweils fünf *Quarters*. Die Struktur des Rasters über Oldenburg lässt sich der Abbildung 19 entnehmen. Die mit Buchstaben versehenen Marker in der Karte stehen dabei jeweils für die Eckpunkte der einzelnen *Quarters*.

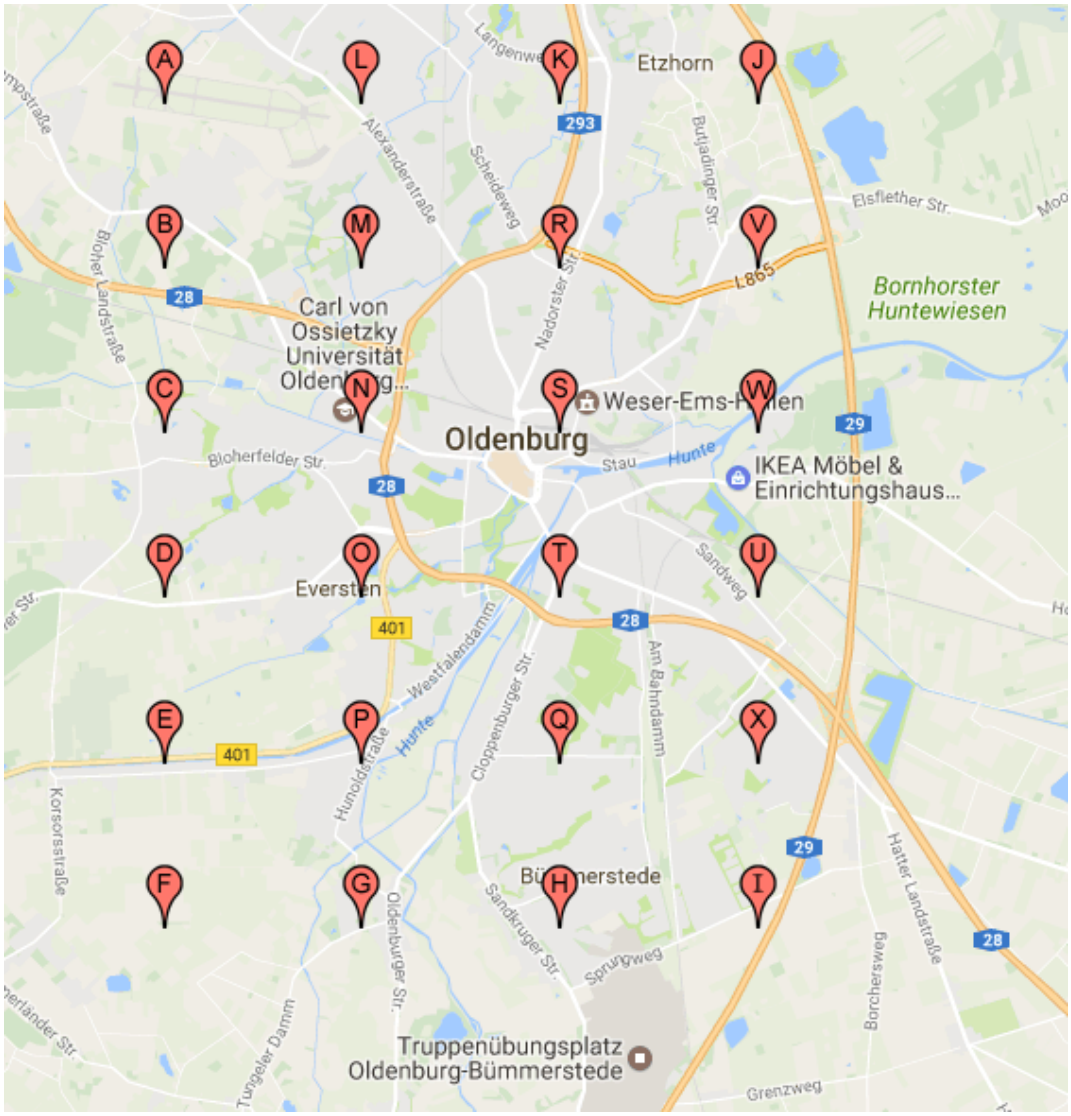


Abbildung 19: Darstellung der tatsächlichen Quarter zur reaktiven Standortplanung über Oldenburg

Als Zeitfenster wird eine Dauer von 15 Minuten benutzt und als Grenzwert zum Auslösen der Reaktion eine Anzahl von fünf Aufträgen in dieser Zeitspanne. Um das Zeitfenster in Odysseus darzustellen wird der *TIMEWINDOW*-Operator benutzt. Dieser sieht in unserem Fall wie folgt aus:

```
1  output = TIMEWINDOW({
```

```
2                 size = [15, 'MINUTES'],
3                 advance = 1,
4               }, app:orders)
```

Die Berechnung der einzelnen Grenzwerte der Bereiche werden mithilfe einer Aggregation berechnet sobald ein neuer Auftrag in Odysseus ankommt. Der Grenzwert wird zu Beginn mit der Anzahl 5 festgelegt, kann allerdings im Nachhinein angepasst werden. Bei jedem eintreffenden Auftrag wird ein Zähler innerhalb des jeweiligen Quarters inkrementiert. Sollte dieser Zähler den Grenzwert überschreiten, wird automatisch ein freies Taxi zum Mittelpunkt des *Quarters* gesendet.
Das aktuelle Vorgehen könnte in einem weiteren Schritt noch verfeinert werden, indem nicht nur äquivalente *Quarters* benutzt werden sondern einzelne *Quarter* näher spezifiziert werden. So könnte man überlegen *Quarter* mit generell hohem Auftragsaufkommen weitere aufzuteilen und in kleinere Einheiten zu bringen. Ein Beispiel für eine sinnvolle Verfeinerung der Aufteilung ist die Innenstadt, die aktuell zum großen Teil in ein *Quarter* fällt. Weiterhin ist zu überlegen statt den aktuellen statischen Grenzwerten zur Aktivierung der Quarter, dynamische Grenzwerte anhand des aktuellen Auftragsaufkommens im Gesamtsystem für die *Quarter* zu erzeugen.

6.6 App

Die App ist dazu da, es dem Nutzer zu ermöglichen einen Auftrag anzulegen und somit ein Taxi zu bestellen. Der Auftrag wiederum löst die Routenberechnungen mit Hilfe von *Odysseus*. Die App ist auf das Betriebssystem Android ausgelegt. Die Entscheidung für die Entwicklung einer App auf Basis von Android wurde getroffen, da man zur Androidentwicklung jeden beliebigen Computer bzw. alle üblichen Computerbetriebssysteme verwenden kann. Zusätzlich hatten mehrere Gruppenmitglieder bereits Vorkenntnisse in der Entwicklung mit Android. Weiterhin stellte uns das Offis mobile Endgeräte zur Verfügung, auf denen wir unsere App bauen und testen konnten.

6.6.1 Nutzermanagement

Die App enthält ein Nutzermanagement und um die Funktionen der App nutzen zu können, ist zunächst eine Registrierung des Nutzers notwendig. Nach Angabe von Name, Vorname, E-Mail-Adresse, Telefonnummer, Passwort und Zahlungsinformationen kann er sich nun mit seiner E-Mail-Adresse und seinem Passwort einloggen. Seine Nutzerdaten kann der Nutzer im Profil in der App bearbeiten. Hier stehen ihm mit dem Ausloggen oder dem Löschen seines Kontos weitere Funktionen zur Verfügung. Über den Button „Passwort ändern“ im Nutzerprofil gelangt der Nutzer auf eine Activity, auf der der Nutzer sein Passwort ändern kann. Eine Funktion zur Erneuerung des Passworts – falls der Nutzer sein Passwort vergessen hat und sich anmelden möchte – steht dem Nutzer im Login-Screen zur Verfügung. Eine weitere nutzerspezifische Funktion der App ist das Speichern von Favoriten. Als Favoriten können sowohl Startpunkte bzw Abholorte als auch Zielorte gespeichert werden. Diese kann der Nutzer anschließend bei der Erstellung eines Auftrags nutzen.

6.6.2 Erstellung eines Auftrages

Ist der Nutzer registriert und angemeldet kann er einen Auftrag erstellen. Dieser Vorgang besteht aus fünf Schritte. Diese werden jeweils durch eine Aktivität in Android repräsentiert. Der Vorgang läuft wie folgt ab:

- Auswahl des Abholortes (StartingPointActivity.java): Ist der Nutzer in der

Aktivität zur Auswahl des Abholortes, wird ihm eine Karte eingeblendet, auf der ein Marker seinen aktuellen Standort markiert. Möchte der Nutzer nicht an seinem aktuellen Standort abgeholt werden, so kann er diesen Marker zur Auswahl eines anderen Ortes vorschieben. Hat der Nutzer dies getan, wird ihm die Adresse des gewählten Ortes in einem unten eingeblendeten Textfeld angezeigt. Zusätzlich zum Setzen des Markers kann der Nutzer die gewünschte Adresse in ein Textfeld eingeben. Während der Suche werden ihm Suchvorschläge eingeblendet, die er durch anklicken auswählen kann. Hat er dies getan, wird wiederum der Marker an die entsprechende Position der eingegebenen Adresse gesetzt und das Textfeld mit der entsprechende Adresse gefüllt. Möchte der Nutzer den Ort auf seinen aktuellen Ort zurücksetzen, bietet die Karte über ein Symbol rechts oben die Möglichkeit, dies zu tun. Möchte der Nutzer einen Startort als Favorit speichern, kann er dies über einen neben dem Textfeld liegenden Button tun. Des Weiteren hat er die Möglichkeit, über die oben angezeigten Buttons seine Favoriten auszuwählen oder sein Profil zu betrachten. Hat der Nutzer den Abholort ausgewählt, gelingt er durch Klicken auf den Button „Weiter„ in die nächste Aktivität zur Auswahl des Zielortes.

- Auswahl des Zielortes (DestinationActivity.java): Die Auswahl des Zielortes verläuft analog. Durch Klicken auf den „Weiter“-Button gelangt der Nutzer in zusätzliche Einstellungen.

- Weitere Angaben (AdditionalInformationAcitivity.java): Hier hat der Nutzer die Möglichkeit, ein Zeitfenster anzugeben in dem er abgeholt werden möchte bzw. ankommen möchte. Er gibt das Datum der frühestmöglichen Abholung sowie der spätestens möglichen Ankunft an. Der Nutzer hat jedoch die Möglichkeit entweder nur ein Datum für die Abholung oder nur ein Datum für die Ankunftszeit einzutragen. Trägt der Nutzer lediglich ein Ankunftsdatum ein, wird als Abholdatum das aktuelle Datum und die aktuelle Zeit gesetzt. Zusätzlich kann er wählen, mit welchem Fahrzeugtyp er fahren möchte, ob er alleine fahren möchte, ob er einen Rollstuhl- und/oder Fahrradslot benötigt und ob er zusätzliche Gepäckstücke bei sich trägt. Wiederum über den „Weiter“-Button gelangt er in den nächsten Schritt bzw. zur nächsten Aktivität. Sobald auf den Button geklickt hat, öffnet die App eine

TCP-Verbindung zu Odysseus und sendet den Kundeauftrag.

- Warten auf die berechnete Route (WaitingActivity): Auf dieser Activity wartet der Kunde, während sein Auftrag berechnet und eine Route erzeugt wird. Ist die Route erzeugt, wird per Firebase Cloud Messaging eine Benachrichtigung and die jeweilige App des Kunden gesendet. Empfängt die App die Benachrichtigung von Firebase, wird eine TCP-Verbindung zu Odysseus aufgebaut und die Route empfangen. Ist die Route empfangen, wechselt der Nutzer automatisch auf die nächste Activity.

- Vorhersage (PredictionActivity.java): Auf dieser Acitivity wird dem Nutzer angezeigt welche Route sein Taxi nehmen wird. Außerdem werden ihm weitere Informationen angezeigt, wie beispielsweise Zwischenziele, Kosten, Distanz usw. Weiterhin wird dem Nutzer durch einen Taxi-Marker dauerhaft die aktuelle Position seines Taxis angezeigt.

6.6.3 Verwendete Technologien und Frameworks

- GoogleMaps: Alle Karten der App sind GoogleMaps-Karten und verwenden dementsprechend auch dessen Funktionen und API. Das Setzen der Marker verläuft über die GooglePlaces API. Das Umwandeln der durch den Marker erzeugten GPS-Daten in Adressen nutzt ebenso wie das Umwandeln von Adressen in GPS-Daten für das Setzen der Marker die GoogleMaps Geocoding API.

- MySQL: Alle Informationen, die der Nutzer bei der Registrierung bzw. bei der Änderung der Profildaten angibt, werden in einer MySQL-Datenbank gespeichert. Daten, die an die Datenbank gesendet werden sollen, werden über eine Rest-Schnittstelle gesendet. Die Restschnittstelle wurde in PHP implementiert. Um die Restschnittstelle anzusprechen wird das Framework OkHttp verwendet. Um ein Objekt einer Klasse – in diesem Fall Benutzer – zu senden wird das Objekt mit Hilfe von *gson* in ein JSON-Format convertiert.

- OkHttp: Bei OkHttp handelt es sich um Framework mit dem verschiedene HTTP-Request erstellt und ausgeführt werden können. Im Rahmen unseres

Projektes wird OkHttp benutzt um auf die REST-Schnittstelle zuzugreifen und alle für das Nutzermanagement notwendigen Funktionen auszuführen.

- Gson: Gson ist eine Libary von Google, die es ermöglicht Java-Objekte in das JSON-Format zu convertieren und aus dem JSON-Format Java-Objekte zu erzeugen.
- Butterknife: Um UI-Objekt in XML-Dateien mit Java-Objekt zu verknüpfen, werden die UI-Objekte mit Hilfe von Butterknife injectet.
- JUnit: Tests für wesentliche Funktionen der App wurden mit Hilfe vo JUnit getestet.
- Firebase Cloud Messaging: Wird benutzt um die App des Kunden zu benachrichtigen sobald seine Route berechnet ist bzw. eine Routenaktualiseriung vorliegt.
- AngularJS: AngularJS ist ein clientseitiges Java-Webframework zur Erstellung von Single-Page-Webanwendungen.
- Spring MVC: Spring MVC ist ein Framework zur Erstellung von Webanwendungen. Es ist eine Erweiterung des Spring Frameworks, das als quelloffenes Framework und mit einem breiten Spektrum an Funktionalitäten die Entwicklung mit Java vereinfachen soll.
- Leaflet: Leaflet ist eine JavaScript-Bibliothek, mit der Web Map Tile Services zusammen mit eigenen Geodaten auf einer Webseite präsentiert werden können.

6.6.4 Nutzung von Odysseus

Nachdem ein Taxi vom Nutzer gerufen worden ist, wird dieser mittels TCP-Verbindung im CSV-Format an Odysseus gesendet. Das Empfangen eines Auftrag löst dann die entsprechende Methode im angelegten Operator in Odysseus aus. Hier werden die notwendigen Berechnungen zur Erstellung der Route durchgeführt. Sind diese abgeschlossen wird die Route im CSV-Format wiederum per TCP-Verbindung an die App gesendet. Zusätzlich zur Route sind Informationen über Ankunftszeitpunkt, Kosten und Distanz zum Zielort enthalten. Diese werden

durch die entsprechende Textfelder in der App angezeigt. Die empfangene Route wird durch die Karte visualisiert, in der zwei Marker Start und Ende der Route markieren.
Hat der Nutzer die Option Auftrag stornieren gewählt, wird wie oben geschrieben ein Auftrag gesendet. Dieser enthält die gleiche ID und wird zusätzlich mit einem Status versehen, der der Routenplanung signalisiert, dass der Auftrag storniert wurde.

6.6.5 Umsetzung der App

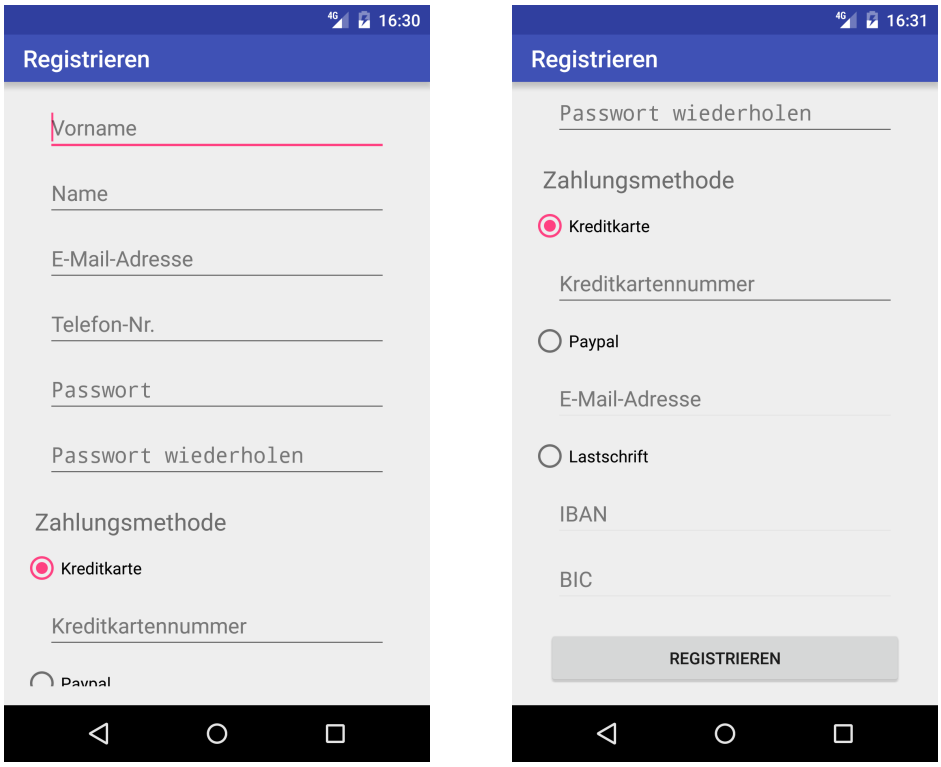
Die App wurde durch insgesamt zehn Android-Activities umgesetzt, die dazu genutzt werden, dem Kunde den oben beschriebenen Ablauf zum Bestellen eines Taxis zu ermöglichen. Folgende Acitivities wurden von der Projektgruppe erstellt:

- AdditionalInformationActivity
- DestinationActivity
- ForgotPasswortActivity
- LoginActivity
- MainActivity
- PredictionActivity
- ProfileActivity
- RegisterActivity
- StartingPointActivity
- WaitingActivity

Im Folgenden werden die Activities in logischer Reihenfolge anhand von Screenshots vorgestellt und ihre Funktionsweise näher erläutert.

Abbildung 20: MainActivity

PredictionActivity Diese Acitvity sieht der Nutzer sobald er die App öffnet. Sie bietet dem Nutzer keinerlei Interaktionsmöglichkeiten. Allerdings wird durch diese Activity im Hintergrund geprüft, ob die *Google Play Services* auf dem Gerät bereits installiert sind. Sollte dies nicht der Fall sein, wird der Nutzer darauf hingewiesen und kann sie direkt installieren. Weiterhin wird im Hintergrund geprüft, ob bereits ein Nutzer eingeloggt ist, sich also Daten in den SharedPreferences befinden. Die MainActivity sieht wie in Abbildung 20 dargestellt aus.



(a) Activity Register oben (b) Activity Register unten

Abbildung 21: Activity Register

RegisterActivity Über die RegisterActivity kann sich der Nutzer für den angebotenen Service registrieren. Er muss dazu sein persönlichen Daten, wie Name, E-Mail-Adresse und Passwort angeben. Ist die Registrierung erfolgreich, werden die Daten in unsere MySQL-Datenbank geschrieben und der Kunde ist in der Lage sich anzumelden. Die RegisterActivity sieht wie in der Abbildung 21 dargestellt aus.

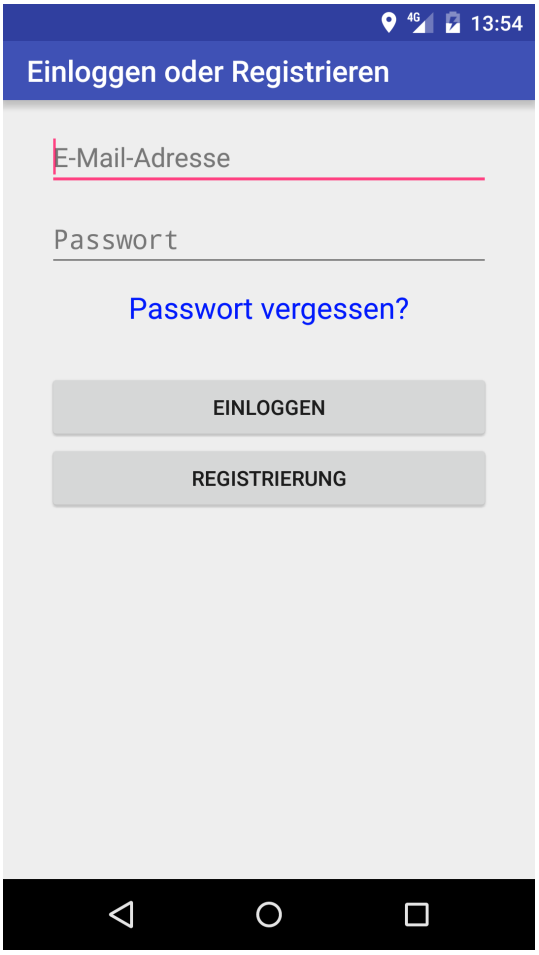
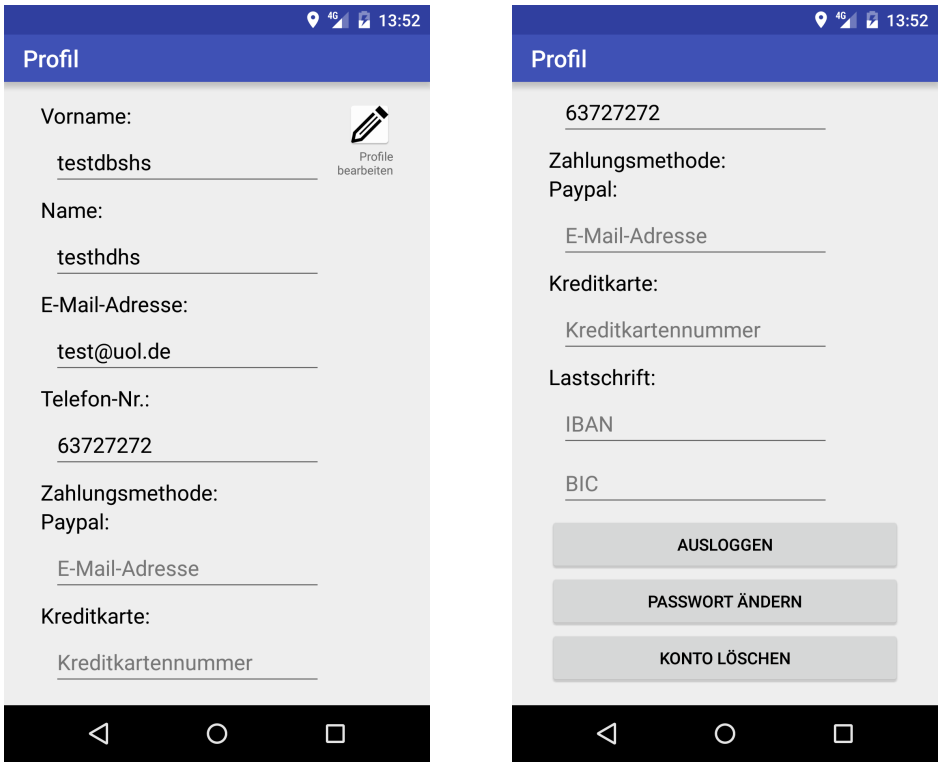


Abbildung 22: LoginActivity

LoginActivity Über diese Activity kann sich der Kunde in der App durch seine persönlichen Daten anmelden. Voraussetzung dafür ist, dass er bereits ein Konto besitzt. Ist dies nicht der Fall, kann er weiter zur RegisterActivity gelangen, um sich zu registrieren. Diese Activity wird nur angezeigt falls noch keine Nutzerdaten in den SharedPreferences der App gespeichert sind. Sie sieht wie in Abbildung 22 dargestellt aus.

Abbildung 23: ChangePasswortActivity

ChangePasswortActivity Diese Activity gibt dem Kunde die Möglichkeit, sein Passwort zu ändern. Dazu muss er zunächst sein aktuelles Passwort korrekt eingeben und anschließen zweimal sein gewünschtes neues Passwort bestätigen. Die Activity sieht wie in Abbildung 23 dargestellt aus.



(a) Profile Activity oben

(b) Profile Activity unten

Abbildung 24: Profile Activity

ProfileActivity Auf der ProfileActivity kann der Nutzer alle seine persönlichen Daten einsehen, verwalten und ändern. Dazu kann er den Button in Form eines Stifts oben rechts verwenden. Außerdem bietet das Profil die Möglichkeit der Weiterleitung zur ChangePasswortActivity. Darüber hinaus kann der Nutzer über das Profil sein Benutzerkonto löschen oder sich von der App ausloggen. Die RegisterActivity sieht wie in der Abbildung 24 dargestellt aus.

Abbildung 25: StartingPointActivity

StartingPointActivity Über diese Activity wählt der Kunde den Abholort seiner Taxifahrt an. Dafür hat der Nutzer mehrere Möglichkeiten. Er kann entweder bereist als Favoriten gespeicherte Adressen benutzen, den Marker auf der Karte auf dem Bildschirm an den gewünschten Ort ziehen oder die Suchfunktion benutzen. Standardmäßig steht der Marker auf der Karte und somit die ausgewählte Adresse auf dem aktuellen Standort des Kunden. Sollte er den aktuellen Standort anpeilen wollen, so ist dies über den Button oben rechts in der Karte möglich. Die Activity sieht wie in Abbildung 25 abgebildet aus.

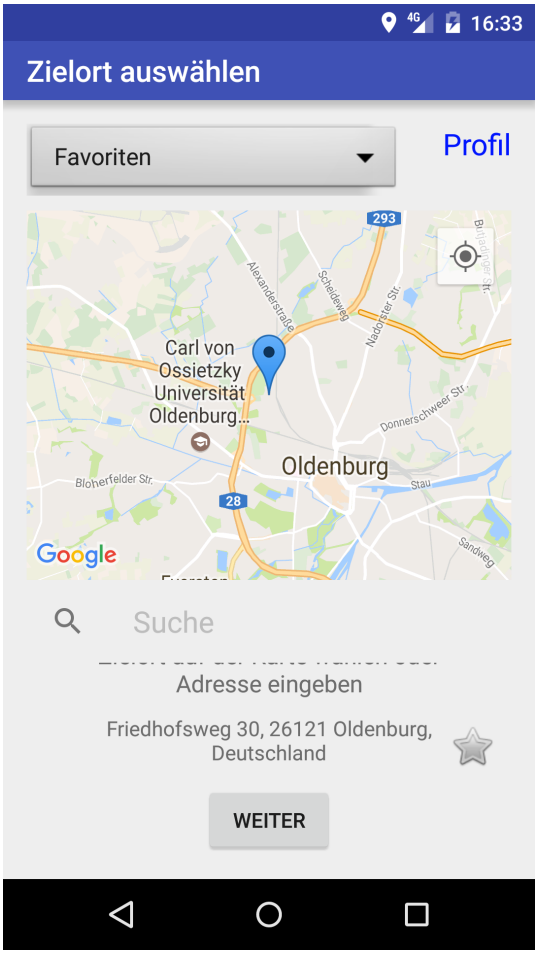


Abbildung 26: DestinationActivity

DestinationActivity Über diese Activity wählt der Kunde den Zielort seiner Taxifahrt. Dafür hat der Nutzer die Möglichkeiten eine bereist als Favoriten gespeicherte Adressen zu benutzen, den Marker auf der Karte auf dem Bildschirm an den gewünschten Ort zu ziehen oder die Suchfunktion zu benutzen. Die Activity sieht wie in Abbildung 26 abgebildet aus.

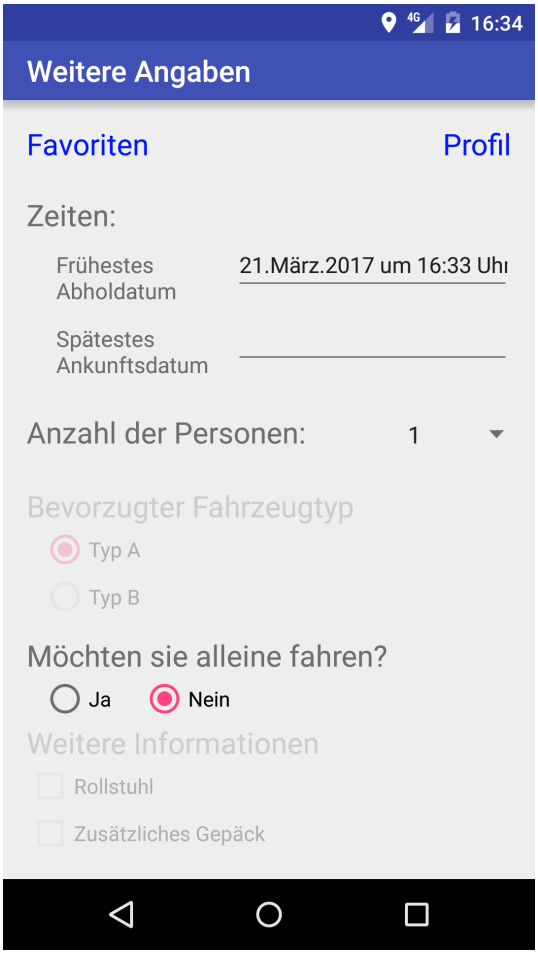


Abbildung 27: AdditionalInformationActivity

AdditionalInformationActivity Diese Activity dient dem Kunden dazu, weitere Informationen zu seiner Fahrt einstellen zu können. Sie erscheint nachdem er bereits einen Abhol- und Zielort festgelegt hat. Auf der AdditionalInformationActivity kann er nun durch Datepicker Start- und Ankunftszeitpunkte einstellen. Weiteren Informationen sind u. a. Anzahl der Personen oder ob er alleine fahren will. Die Activity sieht wie in Abbildung 27 dargestellt aus.

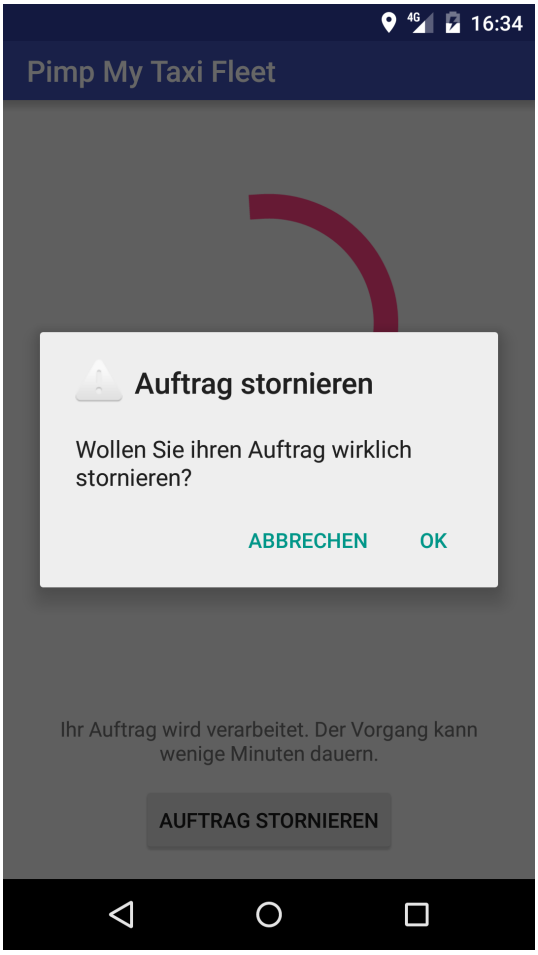


Abbildung 28: WaitingActivity

WaitingActivity Während die Route in Odysseus berechnet und der Auftrag verarbeitet und zugewiesen wird, wird dem Kunde in der App ein Wartebildschirm angezeigt (siehe Abbildung 28). Währenddessen hat der Nutzer die Möglichkeit jederzeit seinen Auftrag zu stornieren. Diese Activity wird dem Nutzer solange angezeigt bis Odysseus der App per *Firebase Cloud Message* rückmeldet, dass die Route berechnet ist.

6.7 Persistentes Datenmanagement

Neben der Vielzahl an im Gesamtsystem auftretenden Datenströmen, die mit Hilfe von Odysseus verarbeitet werden, treten ebenso Daten auf, die für eine erneute Verwendung persistiert werde müssen. Im Folgenden wird zunächst auf die einzelnen abstrakten Daten, ihre Speicherung in einer Datenbank und anschließend auf die Bereitstellung der Daten für die App eingegangen.

6.7.1 MySQL Datenbank

Für die Verwaltung der Bestandsdaten wird eine MySQL Datenbank eingesetzt. MySQL ist ein weit verbreitetes Open-Source Datenbankverwaltungssystem, zur Verarbeitung von relationalen Daten. Es kann im Rahmen des Projektes kostenfrei unter der General Public License verwendet werden und wird von den eingesetzten Programmiersprachen und Betriebssystemen unterstützt. Zur Verwaltung der Datenbank werden die ebenso kostenfreien Softwarelösungen MySQLWorkbench und PHPMyAdmin verwendet. Das Datenbankschema wurde aus den zu speichernden Objekten der Systemkomponenten abgeleitet und verallgemeinert. Es ergeben sich Tabellen zu folgenden Objektklassen, die in der Abbildung 29 detailliert spezifiziert sind:

- Nutzer
- Ladesäulen
- Beförderungsaufträge
- Taxis

Alle Klassen und Attribute sind eindeutige Worte oder im Kontext gängige Abkürzungen in englischer Sprache. App, Dashboard und Odysseus sind sowohl Datenquellen als auch Datensenken. Neue Benutzer werden in der Regel über die App erstellt und in die Datenbank geschrieben. Bei jeden erneuten Anmelden, wird ein Zugriff auf die App und die eigenen Benutzerdaten erst gewährt, wenn die Richtigkeit der Anmeldedaten geprüft wurde. Näheres wird im folgenden Kapitel erläutert. Odysseus liest bei Systemstart alle Ladesäulen und Taxis aus der Datenbank aus,

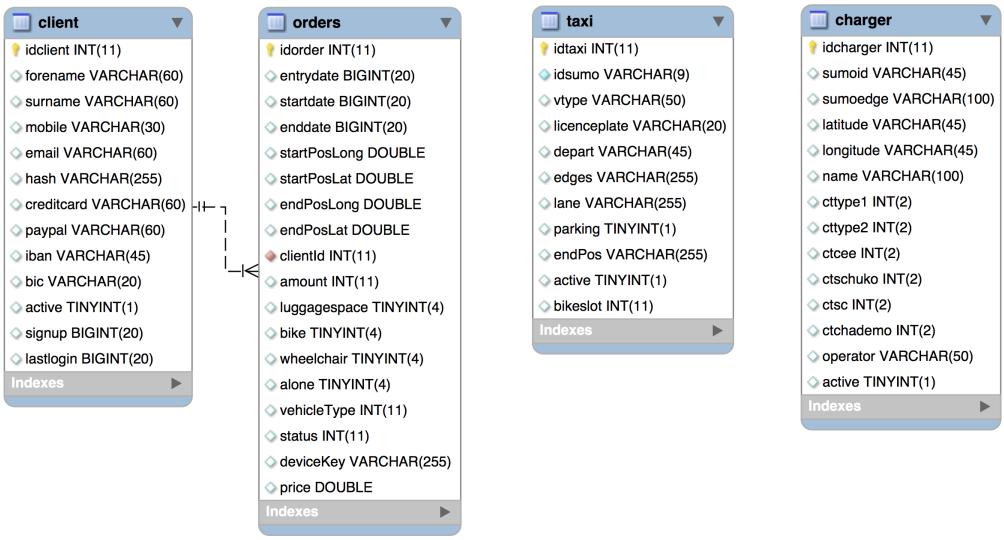


Abbildung 29: Datenbankschema

um diese dem internen Datenpool und der Verkehrssimulation SUMO bereitzustellen. Ebenso werden neue Taxis und von der App erhaltene Aufträge über Odysseus sowohl dem internen Datenpool bereitgestellt, als auch zur längerfristigen Speicherung in die Datenbank geschrieben. Ein ähnliches Vorgehen wäre zukünftig auch für Ladestationen aus externer Quelle denkbar. Gespeicherte Aufträge aus der Datenbank können über das Dashboard einzeln nachvollzogen oder als Gesamtmenge auf einer Heatmap dargestellt werden. Zukünftig geben die Koordinaten dieser Datenpunkte das Potential, Heuristiken aus vergangenen Aufträgen zu generieren und auf zukünftige Auftragsvorkommen zu schließen.

6.7.2 RESTful App Interface

Eine RESTful Schnittstelle wird zur serverseitigen Realisierung des Nutzermanagements eingesetzt. *Representational State Transfer* ist grundsätzlich ein Programmierparadigma das sich dem Problem der Maschinen-zu-Maschinen-Kommunikation annimmt. Im vorliegenden Fall entkoppelt es die direkte Datenkommunikation zwischen App und Datenbank und repräsentiert die Inhalte der Datenbank als Ressourcen über eine Webadresse nach dem standard Internetprotokoll

HTTP. Diese Architekturentscheidung ermöglicht, gegenüber dem direkten Zugriff der App auf die Datenbank, die Kommunikation zu standardisieren. So erfordern Änderungen in Datenstruktur oder Logik im Backend keine Anpassungen und Updates der App. Implementiert ist die API in der serverseitigen Skriptsprache PHP, die sich durch breite Datenbankunterstützung, Internetprotokolleinbindung und zahlreiche Funktionsbibliotheken auszeichnet. Die in der Datenbank liegenden Inhalte werden von der API geladen und über spezifische Pfade als JSON Objekte für die App bereitgestellt. Die App kann diese Daten nach einer erfolgreicher Verifizierung mittels HTTP Request über die folgenden HTTP-Methoden auslesen oder manipulieren.

- GET - Bestehenden Eintrag auslesen
- POST - Neuen Eintrag hinzufügen
- PUT - Bestehenden Eintrag verändern
- DELETE - Bestehenden Eintrag löschen/deaktivieren

Die Verifizierung der HTTP Requests wird über die Variable *HTTP_API_KEY* im HTTP Header vorgenommen. Stimmt die von der App mitgelieferte Schlüssel Variable nicht mit der Variable in der Konfigurations Datei überein, wird der Zugriff verweigert und eine Fehlermeldung zurückgegeben. Alle verwendeten Fehlercodes wurden aus den HTTP Status Codes 400 - 501 übernommen. Der Basispfad für Anfragen kann beliebig gewählt werden und ist in der App zu hinterlegen. In diesem Fall wurde die Adresse *http://134.106.56.4:23080/app2db/api.php* verwendet. Dieser kann je nach Methode um verschiedene Pfade wie */client/byLogin* erweitert werden. Bei auslesenden Abfragen mit der HTTP GET Methode, werden die für die Abfrage benötigten Daten in der URL mitgegeben, sodass sich eine URL wie *http://host/app2db/api.php/client/byLogin?email=mail@uol.de&password=secret* ergibt. Manipulierende Anfragen müssen hingegen mitgelieferte Daten im Post Body als JSON Objekt übergeben.
Da aus Sicherheitsgründen Passwörter nicht im Klartext in der Datenbank abzulegen sind, werden diese lediglich als Prüfsummen in der Datenbank hinterlegt. Beim Anlegen eines neuen Nutzers wird somit aus dem gegebenen Passwort ein Hash ermittelt und der Zeitpunkt der Registrierung erfasst. Beim Anmelden wird

mit den selben Parametern ein erneuter Hash gebildet und mit dem bestehenden Verglichen. Eine erfolgreiche Anmeldung wird mit Zeitstempel in der Datenbank vermerkt.

6.8 Verkehrssimulation

Als Verkehrssimulation wurde mit Hilfe einer Evaluation SUMO ausgewählt. Die Abkürzung SUMO steht für „Simulation of Urban MObility“ und ist ein kostenloses, quelloffenes Open Source Projekt, welches hauptsächlich vom Deutschen Zentrum für Luft- und Raumfahrt entwickelt wird. Es ist in C++ implementiert und verwendet ein mikroskopisches Fahrzeugfolgemodell, welches mit Hilfe von Sensormessdaten eines Autobahnabschnittes kalibriert wurde.

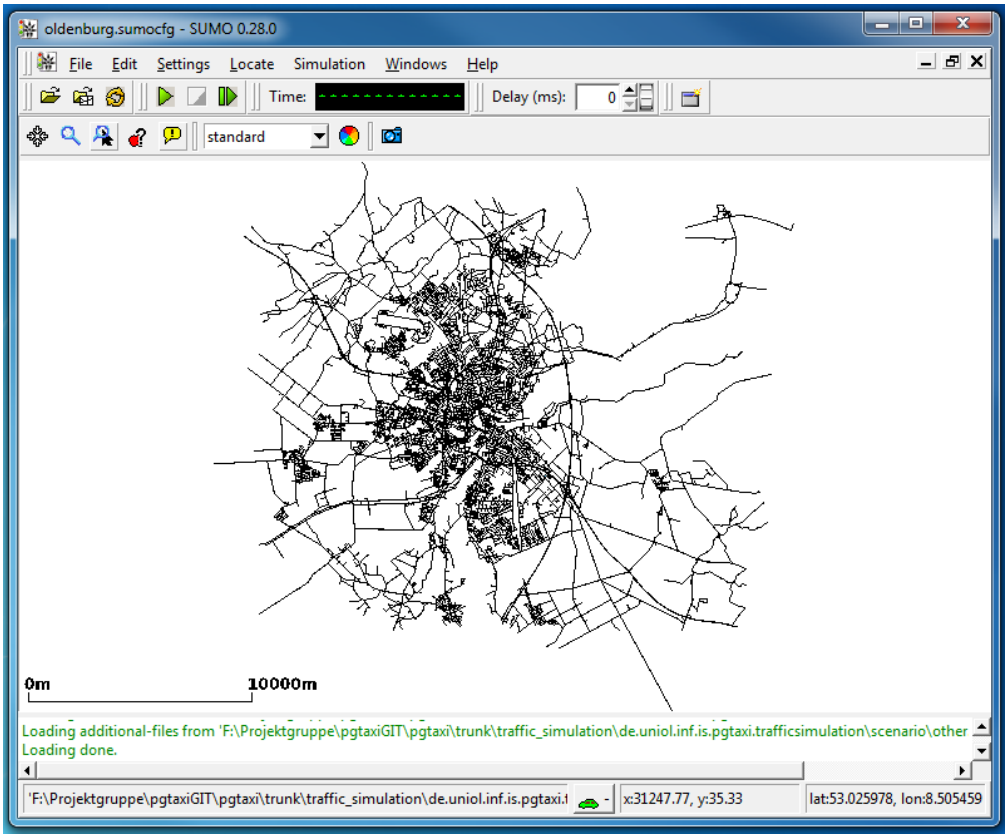


Abbildung 30: SUMO mit grafischer Oberfläche

Die Fahrzeuge werden raumkontinuierlich sowie zeitdiskret mit einem Zeitschritt von einer Sekunde simuliert. In der Verkehrssimulation werden aus soziodemo-

graphischen, statistischen Daten der simulierten Welt Tagespläne ermittelt, die dann mit Hilfe von simulierten Agenten ausgeführt werden. In SUMO besteht die Möglichkeit multimodalen Verkehr zu modellieren, was bedeutet, dass verschiedene Verkehrsteilnehmer und Fahrzeugtypen simuliert werden können. Allerdings sind andere Verkehrsteilnehmertypen als Autos nur durch Schätzung der Reisedauer, also ohne die Simulation des zurückgelegten Weges implementiert. Somit kann eine Interaktion zwischen den verschiedenen Verkehrsteilnehmertypen nicht berücksichtigt werden. SUMO wird seit 2001 entwickelt und erweitert. So stellt es Funktionen zur Generierung von Graphen aus Kartenmaterial im Shapefile- und OpenStreetMap-Format (OSM) zur Verfügung. Um eine Kommunikation mit SUMO aufnehmen zu können, kann das in Python geschriebene TraCI verwendet werden. Dabei werden verschiedene Funktionen, wie zum Beispiel die Abfrage einer Position eines Fahrzeuges, zur Verfügung gestellt. Um diese Funktionen nach den eigenen Anforderungen nutzen zu können, muss zunächst ein eigenes Python-Skript geschrieben werden, welches alle gewünschten Funktionen der TraCI Bibliothek implementiert. Die wichtigsten Funktionen sind dabei:

- initLocal: Baut einen lokalen TCP/IP Socket auf, um mit TraCI kommunizieren zu können.
- doTimestep: Führt einen Zeitschritt innerhalb der Simulation durch.
- getGPSPosition: Gibt die GPS-Position eines Fahrzeuges zurück.
- addVehicle: Fügt der Simulation ein Fahrzeug hinzu.
- stop: Fügt für ein bestimmtes Fahrzeug an einer bestimmten Position einen Stop hinzu.
- resume: Lässt ein Fahrzeug weiterfahren.
- weitere Hilfsfunktionen.

Da kein Projektgruppenmitglied Vorkenntnisse in Python hat, wurde deshalb als Basis ein Java Projekt erstellt. Außerdem wurde mit Hilfe von Jython eine Verbindung zwischen dem Java Projekt und den Python-Skripten hergestellt. Dafür wurde zum einen die Klasse PyManager.java geschaffen, welche Aufgaben wie die

Konvertierung und Ausführung von Python-Funktionen übernimmt und somit spezielle Funktionen, wie die Abfrage einer Position, erfüllt. Eine weitere Jython Klasse ist PyRunner.java, welche die Kommunikation zu einem weiteren Python-Skript übernimmt, das zum Starten von SUMO verwendet wird.

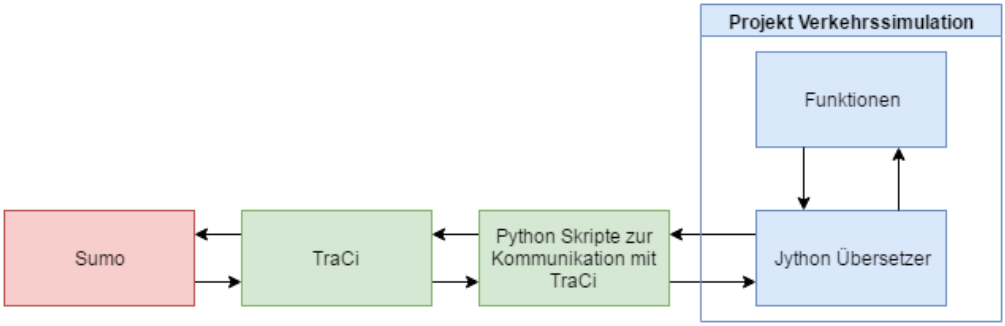


Abbildung 31: Architektur vom Verkehrssimulationsprojekt

Das Architektur-Schaubild zeigt den typischen Weg zum Abfragen von Werten wie die Fahrzeugposition. Beim Start einer Abfrage wird der Befehl immer von Jython übersetzt, welcher dann in den geschriebenen Python-Skripten an unsere Bedürfnisse angepasst wird. Das Python-Skript ruft dann eine TraCI Funktion auf. TraCI holt sich dann den Abfragewert von SUMO und sendet es zurück an das Python-Skript, welches zurück an Jython sendet, wo es zurück in das Java Format übersetzt wird.

6.8.1 Routenerzeugung in SUMO

Um die von Odysseus gesendete Route in SUMO zu empfangen, wurde ein Socket textitRouteDataReceiver erstellt, welches aus der empfangenen Route im CSV-Format die Taxi-ID, die GPS-Koordinaten der Route und der Zwischenstopps ausliest. Es wird anhand der Taxi-ID geprüft, ob bereits ein *RouteCreator* für das Taxi in der Hashmap *routeHandler* für das Taxi existiert. Der *RouteCreator* ist ein Thread, welcher für die Erzeugung der Route des Taxis in SUMO, sowie für das Starten der Taxis und Setzen der Zwischenstopps zuständig ist. In dem Fall, dass bereits ein *RouteCreator* vorhanden ist, wird im *RouteCreator* die Variable *terminated* auf true gesetzt und der *RouteCreator* beendet die Erzeugung der alten Route des Taxis. Anschließend wird für das Taxi ein neuer *RouteCreator* mit der neuen Route erzeugt, welcher dann im *RouteHandler* gespeichert wird. Um eine

Route zu erzeugen, greift der *RouteCreator* über den *PyManager* auf die Python-Skripte für SUMO zu. SUMO benötigt für die Route eines Taxis eine vollständige Reihenfolge an Straßenabschnitten, welche über eine interne ID identifiziert werden. Der *RouteCreator* konvertiert zunächst die GPS-Koordinaten der Route und der Zwischenstopps in die ID der Straßenabschnitte. Die erstellte Route aus den GPS-Koordinaten besitzt jedoch einige Fehler, welche korrigiert werden müssen. Die Route ist häufig nicht vollständig, da nicht unbedingt auf jedem Straßenabschnitt der Route ein GPS-Punkt liegt. Weiter kann der GPS-Punkt sich auf der falschen Fahrbahnseite des Straßenabschnitts befinden oder komplett auf einem falschen Straßenabschnitt, da z.B. die Straße von einer Autobahn gekreuzt wird oder zwei Straßenabschnitte zu eng nebeneinander liegen. Die Korrektur findet in der Funktion *sumoEdgeCorrection()* statt, hier werden die Fehler bestmöglich korrigiert, indem z.B. die sich verändernde Position des Taxis auf dem Straßenabschnitt berücksichtigt wird oder bei fehlenden Straßenabschnitten ein Graph mit dem Straßennetz zwischen den beiden Punkten aufgespannt wird. Jedoch lassen sich nicht alle Fehler korrigieren, so dass nochmals die SUMO-Funktion *reroute* eingesetzt wird, die eine Route zwischen zwei nicht zusammenhängenden Straßenabschnitten erzeugt. Diese Funktion wird jedoch erst nach unserer eigenen Korrektur eingesetzt, da in diesem Fall immer nur zwischen zwei nicht verbundenen Straßenabschnitten eine Route erzeugt wird, welches ohne vorheriger Korrektur häufig zu weiteren unnötigen Schleifen in der Route führt. Zum Schluss werden noch die Zwischenstopps auf der Route mit einer zufällig erzeugten Wartezeit in einem vorher definierten Bereich gesetzt und die Taxis gestartet.

6.8.2 Erweiterungen durch zusätzliche Simulationen

Im Java Projekt namens SumoTraCi wurden im Laufe des Projektes weitere Simulationen und Tools eingefügt. Dazu zählen:

- Eventgenerator: Erzeugt zufällig Events und sendet diese an Odysseus.
- Auftragsgenerator: Erzeugt zufällig Aufträge und sendet diese an Odysseus.
- Generator Straßensperrungen: Erzeugt zufällige Straßensperrungen und sendet diese an Odysseus.
- Verkehrsgenerator: Erzeugt zufällig Verkehr, der an Sumo gesendet wird.

- weitere Hilfsfunktionen.

6.9 Dashboard

Der folgende Abschnitt zeigt die technischen Architektur des entwickelten Dashboards. Hierbei wird auf die einzelnen Komponenten im Detail eingegangen und beschrieben, wie diese miteinander in Beziehung stehen bzw. miteinander kommunizieren. Folgende Abbildung stellt alle Komponenten und Beziehungen visuell dar.

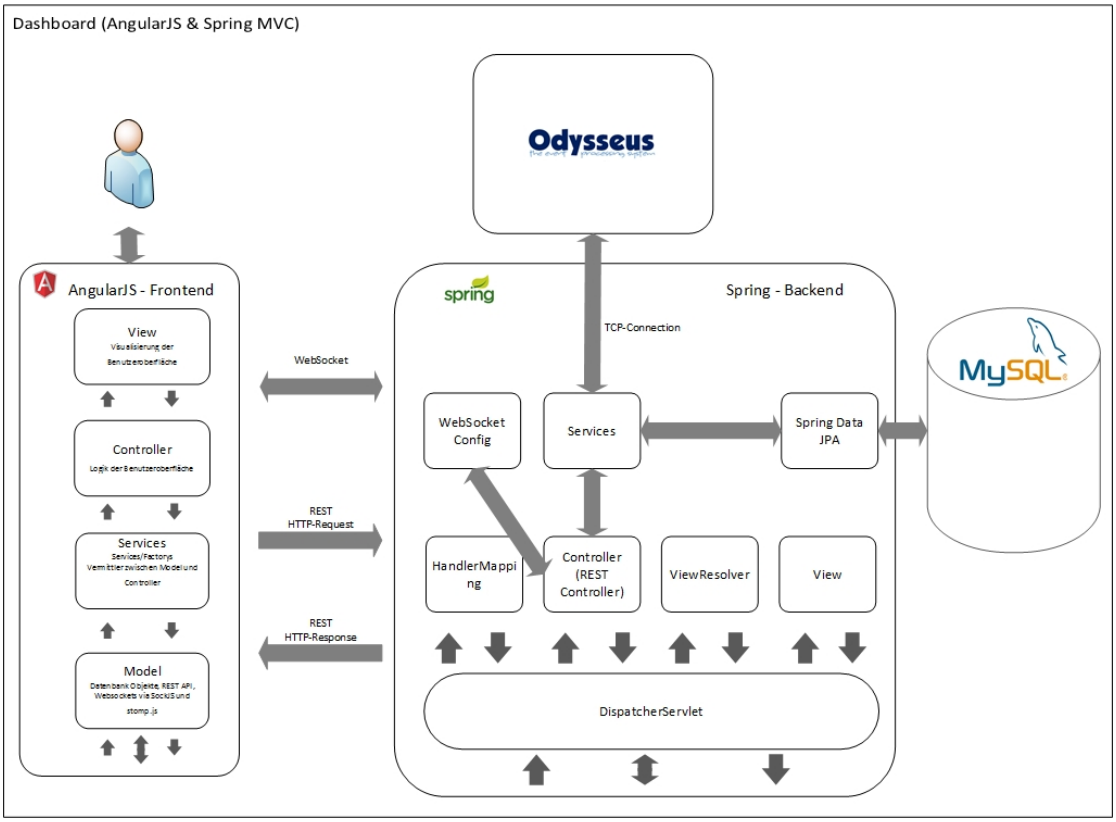


Abbildung 32: Dashboard Architektur

Das Dashboard wurde mithilfe eines Spring MVC-Projekts erstellt. Mit Spring MVC in Verbindung mit JavaScript können dynamische und REST-konforme Webapplikationen, die auf Java basieren, umgesetzt werden. Das Dashboard wurde innerhalb der Spring-Version 4 entwickelt. Wir haben uns für das Spring-Framework entschieden, da das Framework in letzten Jahren immer bedeutsamer geworden ist

und zu den meistgenutzten Frameworks für die Anwendungsentwicklung auf Java-Basis zählt. Daraus folgt eine große hilfsbereite Community sowie große Mengen an Dokumentationen zu vielen verschiedenen Anwendungsfällen bzw. Entwicklungsstrategien.[33] Das Spring-Framework bietet eine umfangreiche Sammlung von Technologien an, die zur Vereinfachung der Entwicklung führen, da es den Entwicklern ermöglicht sich auf wesentliche Business-Logik zu konzentrieren. Die notwendige Software-Infrastruktur wird im Hintergrund bereitgestellt. Hierdurch entstehen Vorteile in den Bereichen der Modularität, Produktivität, Portabilität und Testbarkeit.[34] Das hier verwendete Spring MVC-Projekt bietet eine übersichtliche Model-View-Controller Struktur und greift auf etablierte Technologien, wie die JPA (Java Persistence API) und JSP (Java Server Pages), zurück. Mit Hilfe der vorgegebenen Struktur können die View-Komponenten von Model-Komponenten abgekapselt werden und nur über die Controller-Komponenten kommunizieren. Das heißt, dass die Benutzeroberfläche keine Anwendungslogik enthält und Daten z. B. aus Odysseus oder Datenbanken über die Controller erhält. Das Dashboard wurde unter Berücksichtigung der Dependency Injection entwickelt, um eine lose Kopplung der Klassen zu garantieren. Die Konfiguration der Dependency Injection erfolgten in diesem Fall direkt mit Hilfe von Feld-Annotationen. Das Spring-Framework führt zur Implementierung das zentrale DispatcherServlet ein, welches in der Serverkonfiguration eingerichtet wurde. Die Serverkonfiguration ist in unserem Fall java-basiert und wurde in der WebConfig.java hinterlegt. Das Servlet verarbeitet alle Requests, die unter der konfigurierten URL(in unserem Fall /") eintreffen. Die URLs werden im Handler-Mapping definiert. Des Weiteren muss das Package, in dem die Controller hinterlegt sind, in der Serverkonfiguration angegeben werden. Das Dashboard besitzt zum einen ein Package mit den REST-Controllern und ein Package, das die Controller enthält, die für die Spring-Websockets dienen. Die Controller Klassen wurden mit der Annotation "@Controller"bzw. "@RestController"gekennzeichnet, damit die Applikation die Klassen als Controller erkennt. Zur Bearbeitung der Anfragen gehört es den Output für den User zu rendern. Hierzu bestimmt der ViewResolver, der über das

[33] http://www.heise.de/developer/artikel/Mit-Java-ins-Web-Eine-Spring-MVC-Anwendung-im-Detail-Teil-1-1623020.html

[34] http://uws-software-service.com/de/leistungen/java-kompetenz/softwareentwicklung-mit-dem-spring-framework.html

Dispatcher-Servlet von einem Controller beauftragt wurde, einen logischen View-Namen in ein View-Objekt aufzulösen. Das View-Objekt delegiert die Darstellung an ein Template im Kontext der Webanwendung, in diesem Fall die jeweilige JSP. Die View wird am Ende innerhalb des Frontends dargestellt.[35]
Das Dashboard sendet/empfängt die Daten via REST-basierte HTTP-Requests bzw. durch Spring Websockets. Hierzu werden die Controller über Spring-Services mit Daten versorgt, die eine Schnittstelle zu einer MySQL-Datenbank und Odysseus besitzen. Die Verbindung zur MySQL-Datenbank wird durch eine JDBC-Verbindung hergestellt, die in der Klasse AppConfig.java innerhalb der Spring-Konfiguration festgelegt wurde. Die Daten aus der MYSQL-Datenbank können dann mit Hilfe der Spring-Data-JPA-Bibliotheken bearbeitet und abgerufen werden. Die Verbindung zu Odysseus wurde über TCP-Sockets umgesetzt, wobei der Spring Applikation auf definierte Ports horcht und Daten von Odysseus empfängt bzw. über die TCP-Verbindungen Daten an Odysseus sendet.
Wir haben uns dazu entschieden die Benutzeroberfläche oder Views nicht nur durch Java Server Pages umzusetzen, sondern grundsätzlich mit Hilfe von AngularJS. AngularJS ist ein clientseitiges JavaScript-Framework. Die Vorteile liegen vor allem in der Reduktion des Codes. AngularJS nutzt ebenfalls die Dependency Injection und sorgt somit wiederum für eine gute Testbarkeit und einfache Refaktorierung des Quellcodes. Durch AngularJS-Direktiven kann das HTML-Vokabular erweitert werden. Hierdurch steigt zum einen die Lesbarkeit, durch die semantische Bezeichnung der HTML-Tags des Quellcodes. Mit den zusätzlichen Direktiven können neue Funktionen eingebunden werden, die den Code ebenfalls reduzieren. Das AngularJS-Frontend des Dashboards wurde in den Modulen „myApp.controllers“, „myApp.services“ und dem Root-Modul „myApp“ strukturiert. Die Konfiguration der Module sowie die Abhängigkeiten zu weiteren JavaScript-Bibliotheken sind in der app.js festgelegt. Mit der zentralen index.jsp wird das Frontend initiiert. Dort werden alle nötigen JavaScript-Bibliotheken sowie die app.js, controllers.js und services.js inkludiert. In der controllers.js werden alle Controller festlegt, die die Daten aus verschiedenen Services mit den Templates verbinden. Die Services werden in der services.js festgelegt. Dort werden die Daten über SockJS in Verbindung mit Stomp.js und REST-basierter http-Requests vom Backend empfangen bzw. an das Backend gesendet.

[35]Vgl. S.184-185 Spring im Einsatz(2012), Craig Walls

6.10 Datenströme

Zum Anlegen eines Auftrags, der Berechnung einer Route und Zuweisen eines Taxis fließen Datenströme zwischen den unterschiedlichen Komponenten des Systems. In Abbildung 33 werden diese Datenströme dargestellt. Die Verkehrssimulation SUMO sendet Taxidaten zum TaxiHandler-Operator sowie Verkehrsdaten zum TrafficHandler-Operator. Der TaxiHandler legt neue Taxis im FleetManager an, entfernt entsprechend markierte Taxis oder aktualisiert die Daten bereits im FleetManager existierender Taxis. Ist der Batteriestand eines Taxis unter einem bestimmten Schwellenwert, sendet der TaxiHandler zudem einen Ladeauftrag nach SUMO. Mit Hilfe von Auftragsdaten aus einem Generator oder aus der App sowie den Taxidaten aus dem FleetManager und den Verkehrsdaten aus dem TrafficHandler werden im CalcRoute-Operator Aufträge angelegt, Routen berechnet und den Taxis zugewiesen. Die Route und weitere Informationen wie Fahrtzeit und -dauer werden an SUMO, die App und das Dashboard gesendet.

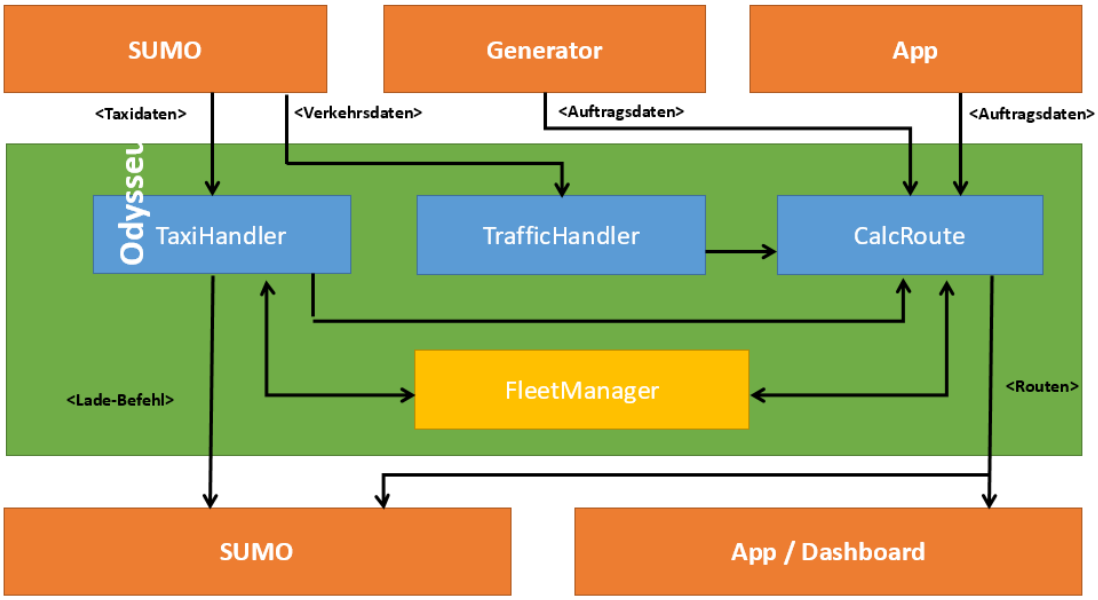


Abbildung 33: Datenströme zwischen den Systemkomponenten

7 Installation

Das Installationshandbuch soll dabei helfen, die entwickelten Softwarekomponenten sowohl einzeln aufzusetzen, als auch ihr Zusammenspiel erfolgreich einzurichten. Hierzu wird in den einzelnen Kapiteln jeweils erläutert, welche vorhandene Software benötigt wird und wie ein Importieren und Konfigurieren der entwickelten Projekte vorzunehmen ist.
Zunächst wird im Kapitel 7.1 erläuert wie Odysseus installiert werden muss. Danach folgt im Kapitel 7.2 die Installation unserer Verkehrssimulation SUMO in der bereits installierten Eclipse Umgebung. Es folgt im Kapitel 7.3 die Einrichtung der Datenbank und der Rest Schnittstelle zur App. Im Kapitel 7.4 wird aufgeführt, wie das Dashboard aufgesetzt und konfiguriert werden muss. Anschließend wird im Kapitel 7.5 erläutert, wie die Android-App zu installieren ist. Im Kapitel 7.6 wird gezeigt, wie die Facebook Schnittstelle zum Empfangen der Veranstaltungen von Facebook installiert und eingerichtet werden muss. Abschließend wird im Kapitel 7.7 erkläre, in welcher Reihenfolge die einzelnen Komponenten des Gesamtsystems gestartet werden müssen, um das Projekt im vollen Umfang zum Laufen zu bringen.

7.1 Odysseus

Zur Verwendung von Odysseus wird folgende Software benötigt:

- Java 8
- Eclipse for RCP and RAP Developers
- GIT Client

Im ersten Schritt muss der Odysseus Source-Code heruntergeladen werden. Dieser Source-Code befindet sich in dem öffentlichen GIT Repository unter der Adresse *https://git.offis.uni-oldenburg.de/scm/ody/odysseus.git*. In Eclipse sollte nun der Ordner, in dem der Odysseus Source-Code gespeichert wurde, als Workspace ausgewählt werden. Anschließend ist ein Import der Java-Projekte *pgtaxi*, *pgtaxi.feature*, *pgtaxi-spring-mvc* sowie *SumoTraCi* aus dem GIT-Repository der Projektgruppe notwendig. Diese Projekte lassen sich unter der Adresse *git.offis.uni-*

oldenburg.de/scm/odr/pg-taxi.git finden. Wegen Abhängigkeiten zu externen Features muss vor der erstmaligen Verwendung von Odysseus in dem Projekt *de.uniol.inf.is.odysseus.com* die Datei *target-platform.target* ausgewählt und auf „Set as Target Platform“ geklickt werden. Im Projekt *challengeServer* befindet sich die Datei „Odysseus Grand Challenge.product“, mit der Odysseus gestartet werden kann. Zuvor muss allerdings unter der Registerkarte *Contents* durch Klicken des Buttons *Add* das Feature des Projekts *de.uniol.inf.is.odysseus.pgtaxi.feature* hinzugefügt werden, um die von der Projektgruppe erstellten Operatoren nutzen zu können. Unter der Registerkarte *Overview* kann man nun Odysseus durch ein Klicken auf „Launch an Eclipse application“ starten. Im Ordner *Scripts* ist das MasterScript zu finden, das durch *Run Script* ausgeführt werden kann.

7.2 Verkehrssimulation

Zur Nutzung der Verkehrssimulation ist folgende Software notwendig:

- Eclipse Mars oder höher
- Windows 7 oder höher
- Java 8 oder höher

SUMO muss nicht zusätzlich installiert werden, da eine ausführbare Version in das Projekt eingebunden wurde. Diese trägt die Versionsnummer 0.28.0.
Zur Nutzung muss der Ordner *traffic_simulation* in Eclipse importiert werden.

- Klicken Sie auf File -> Import.
- Anschließend unter General auf Existing Projects into Workspace.
- Wählen Sie den genannten Ordner.
- Wählen Sie das Projekt SumoTraCi und klicken Sie auf Finish.

Durch Ausführen des Projekts kann nun eine Instanz von Sumo gestartet werden. Um mit dem GUI von SUMO zu arbeiten, ist in der Klasse *StaticParams* im Package *de.uniol.inf.is.pgtaxi.config* die Variable *gui* auf *true* zu setzen. Nach dem Aufbau der Simulation muss auf *Play* geklickt werden. Dadurch werden

die Generatoren und Verbindungen zu Odysseus sowie die Verkehrssimulation selbst gestartet. Dies setzt voraus, dass Odysseus bereits läuft und das Masterscript ausgeführt wird. Alternativ kann Sumo auch ohne GUI ausgeführt werden. Hierzu ist die zuvor genannte Variable auf *false* zu setzen und *SumoTraCi/de.uniol.inf.is.pgtaxi/src/traci/TraciRun.java* in Eclipse als Java Application auszuführen.

7.3 Datenbank und REST API

Für das Datenmangement persistenter Daten wird eine MySQL Datenbank eingesetzt. Der Zugriff der App auf diese findet über eine Rest Schnittstelle statt. Bei der Installation empfiehlt sich die Verwendung von XAMPP als Apache-Distribution, die bereits alle wesentlichen Komponenten bereits enthält, ausschließlich für Entwicklungsumgebungen. Für einen echten Betrieb der Software sollten hingegen Server und Pakete einzeln installiert und konfiguriert werden. Die Installation wird ausschließlich für die Entwicklungsumgebung beschrieben. Folgendes sollte somit vorliegen:

- XAMPP (Distribution für Entwicklungsumgebung) [36]
- Apache HTTP Server (Einzelinstallation) [37]
- PHP (Einzelinstallation) [38]
- MySQL Server (Einzelinstallation) [39]
- MySQL Workbench (hilfreich) [40]
- app2db (REST API) [41]
- pimp.sql (Datenbank) [42]

[36] unter https://www.apachefriends.org/de/index.html
[37] unter https://httpd.apache.org/download.cgi
[38] unter http://php.net/downloads.php
[39] unter https://www.mysql.de/downloads/
[40] unter https://www.mysql.de/products/workbench/
[41] Ordner im GIT unter pgtaxi/trunk/Web/htdocs/app2db
[42] Datei im GIT unter pgtaxi/trunk/Datenbank/pimp.sql

Nach Installation von Webserver und Datenbank können diese über das XAMPP Panel gestartet und konfiguriert werden. Mit Hilfe der MySQL Workbench kann auf die Datenbank zugegriffen werden. Nutzername und Passwort für den Datenbankzugriff befinden sich in der MySQL Server Konfiguration und sollten angepasst werden. Darüber hinaus sollten für den Zugriff von Odysseus und der REST API eigene Benutzer mit beschränkten Rechten angelegt werden. Über den Menüpunkt *Server/Data Import* kann die pimp.sql Datei ausgewählt werden, um das benötigte Datenbankschema inclusive Testdaten zu laden. Der Ordner *app2db* ist im Ordner des Webservers abzulegen und die enthaltene *config.ini* Datei ist an die Konfiguration der Datenbank anzupassen. Anschließend müssen *API Key* und *Serverpfad* entsprechend der gewählten Konfiguration in der App angepasst werden (siehe Kapitel 7.5). Ebenso muss die Konfiguration der Datenbank im *Odysseus Master-Script* und dem *Dashboard* wie im Kapitel 7.4 beschrieben, angepasst werden.

7.4 Dashboard

Bei der Installation des Dashboards ist zwischen einer Installation für Test- und Entwicklungszwecken und einer Installation für einen Live-Betrieb zu unterscheiden. Im Folgenden soll schrittweise erläutert werden, wie die eingesetzten Softwarekomponenten für die Entwicklung des Dashboards eingerichtet wurden. Stellen Sie für die weiteren Schritte sicher, dass Ihnen folgendes vorliegt:

- Eclipse Java EE IDE for Web Developers [43]
- Apache Tomcat Server 8.5 [44]
- springmvcdashboard [45]

Der komprimierte Apache Tomcat Server kann entpackt an einem beliebigen Ort abgelegt werden. Notieren Sie den gewählten Pfad und achten Sie darauf, dass der Pfadname dieses Ordners nicht zu lang wird. Im gestarteten Eclipse muss das Projekt importiert werden (Importieren/General/Existing Projects into Workspace). Geben Sie hierzu den Ihnen bekannten Ordnerpfad von springmvcdashboard an

[43] unter https://eclipse.org/downloads/eclipse-packages/ verwendete Version: 4.6.1
[44] unter http://tomcat.apache.org verwendete Version: 8.5.6
[45] Ordner im GIT unter pgtaxi/trunk/springmvcdashboard

oder importieren Sie das Projekt direkt aus Git. Um den Tomcat Server in Eclipse einzurichten, kann dieser unter dem Reiter *Servers*, der neben der Console zu finden ist, hinzugefügt werden. Wählen Sie die heruntergeladene Serverversion aus der Liste aus und geben Sie den Pfad an, in dem der Apache Tomcat Server zuvor abgelegt wurde. Fügen Sie das Projekt *pgtaxi-spring-mvc* im nächsten Setup-Fenster von Available zu Configured hinzu oder verwenden Sie bei einem bereits installierten Server die Option *Add and Remove*. Wird der Server nun (bspw. über den grünen Startknopf) in Eclipse gestartet, beinhaltet dieser bereits das Projekt und stellt die Webseite in diesem Fall unter *http://localhost:8080/pgtaxi-spring-mvc/#/* bereit. Der Server Port 8080 kann bei bei Bedarf im Project Explorer unter *servers/#servername-config/server.xml* angepasst werden. Hierzu muss die folgende Zeile angepasst werden:

```
1 <Connector connectionTimeout="20000" port="8080"
2   protocol="HTTP/1.1" redirectPort="8442"/>
```

Um erfolgreich eine Verbindung zu Datenbank aufbauen zu können, müssen die Zugangsdaten und Einstellungen zur Datenbank nach dessen installation angepasst werden. Hierzu kann an zentraler Stelle Änderungen an der Datei *config.properties* unter *pgtaxi/trunk/springmvcdashboard/src/main/resources/* vorgenommen werden. Ebenso kann in dieser Datei der *Debug Modus* aktiviert / deaktiviert werden. Um das Produkt im vollen Umfang lauffähig zu halten, sollte der Application zur Verfügung gestellte Cache erhöht werden. Eine Anpassung der Cache-Größe kann im Project Explorer unter *servers/#servername-config/context.xml* vorgenommen werden. Fügen Sie hierzu die folgenden Zeilen innerhalb des *context* Blocks ein:

```
1 <Resources cachingAllowed="true" cacheMaxSize="100000" />
```

Änderungen, die clientseitig - also in AngularJS vorgenommen werden, werden nach Speichern des Quelltextes und Neuladen der Webseite direkt übernommen. Änderungen, die serverseitig - also im Spring-Framework vorgenommen werden, benötigen meist einen Neustart des Servers. Hierzu muss ggf. der Server gestoppt und die Orderstruktur aktualisiert werden. Mit einem Rechtsklick auf den Server können die Optionen Clean Tomcat Working Directory und Clean vor einem Neustart erreicht werden.

7.5 Android-App

Um die Android-App benutzen zu können, muss die korrekte .apk-Daten installiert sein. Die zu installierende .apk-Daten befindet sich im Anhang. Weiterhin kann diese nicht aus dem GooglePlay-Store geladen werden. Sie muss auf dem Android-Gerät geöffnet werden. Anschließend muss „Installieren“ausgewählt werden. Da die Anwendung nicht für Tablet-Geräte angepasst ist, empfehlen wir die App auf einem Smartphone zu installieren. Auf dem Gerät muss die Android-Version 4.0 oder höher installiert sein. Des Weiteren müssen die GooglePlay-Services installiert sein, um die Funktionen der Karten nutzen zu können. Sind diese nicht installiert, wird der Nutzer bei der ersten Nutzung der App allerdings darauf hingewiesen. Er hat dann direkt die Möglichkeit, diese zu installieren. Unter Umständen muss auf dem Gerät die Option, Anwendungen aus fremden Quellen installieren zu können, aktiviert werden. Diese befindet in der Regel unter den Sicherheitseinstellungen des Gerätes. Um die App in vollem Funktionsumfang nutzen zu können, muss der Nutzer der App bestimmte Berechtigungen einräumen. Folgende Berechtigungen müssen der App gegeben werden:

- Allgemeiner (netzwerkbasierter) Standort
- Genauer Standort (GPS)
- Kontaktdaten lesen
- Ihre Profildaten lesen
- Bekannte Konten erkennen
- Google-Systemkonfiguration lesen
- Internetdaten erhalten
- Netzwerkstatus anzeigen
- Vollständiger Internetzugriff
- Ausgeführte Anwendungen abrufen
- Standby-Modus verhindern

Des Weiteren muss kontinuierlich eine Verbindung zum Internet bestehen, um den Funktionsumfang der App zu gewährleisten.
Die Adresse des anzusprechenden Server und dessen Ports können unter *PimpmyTaxiFleet/app/src/main/java/com/taxi/pg/pimpmytaxifleet/odysseus* in TCPConnection.java angepasst werden. Folgende Variablen können angepasst werden:

- IP ist die IP-Adresse der Servers.
- SEND_ORDER_PORT ist der Port, auf dem Aufträge gesendet werden.
- RECEIVE_ROUTE_PORT ist der Port, auf dem die Route aus Odysseus empfangen wird.
- RECEIVE_TAXI_POSITION_PORT ist der der Port, auf dem die Taxiposition empfangen wird. Diese dient dem Livetracking des zugewiesenen Taxis.

7.6 Facebook Schnittstelle

Die Facebook Schnittstelle zum Abfragen der Events ist ein externes Tool, welches separat gestartet werden muss. Es ist webbasiert und wurde in Node.js implementiert. Aufgrund dessen ist die Installation von Node.js notwendig.[46]

Das Facebook Tool ist im Projekt im Ordner *facebook_events* abgelegt. Das Starten ist über die Konsole möglich, indem in das Verzeichnis navigiert wird und der Befehl *npm start* ausgeführt wird.

Nun können die Events zum Beispiel über folgende URL abgerufen werden:
http://localhost:3000/events?lat=40.710803&lng=-73.964040&distance
=100&sort=venue&accessToken=YOUR_APP_ACCESS_TOKEN

Die URL kann auf die eigenen Bedürfnisse angepasst werden.
Die Variable *YOUR_APP_ACCESS_TOKEN* muss durch einen Facebook Access Token ersetzt werden.

WICHTIG: Der Access Token ist 3 Monate gültig und muss nach dieser Zeit im Masterscript erneuert werden.

[46] `https://nodejs.org/en/download/`

7.7 Projekt ausführen

Sind alle Komponenten erfolgreich installiert worden, kann das Projekt wie folgt gestartet werden:

1. Eclipse starten. *Eclipse for RCP and RAP Developers* muss gestartet werden. Odysseus und das Projekt der Projektgruppe müssen importiert sein.

2. Odysseus starten. Da es alle Imports bereits enthält, ist es empfehlenswert das Grand Challenge Produkt von Odysseus zu nutzen. Es muss unter dem Reiter *Contents* das Projekt *pgtaxi* hinzugefügt und die Option *Synchronize* gewählt werden. Anschließend kann das Produkt gestartet werden.

3. MasterScript starten. Im Ordner *Scripts* befindet sich das MasterScript. Es muss ausgeführt werden.

4. SUMO starten. SUMO ist gemäß Kapitel 7.2 zu konfigurieren und kann über *TraciRun* gestartet werden. Ein Klick auf *Play* startet die Simulation.

5. Start des Dashboards. Das Dashboard muss gemäß Kapitel 7.4 gestartet werden.

6. Überwachung durch das Dashboard. Das System ist aktiv und kann nun durch die Funktionen des Dashbaords analysiert werden. Des Weiteren können Aufträge nun per App (Kapitel 7.5) gesendet werden.

Die beschriebenen Schritte können auch auf der sich in der Abgabe befindlichen Virtuellen Maschine ausgeführt werden.

8 Benutzerhandbuch

Das Benutzerhandbuch richtet sich an die Anwender der jeweiligen Software, indem es mit einer Übersicht der Funktionsweise und einer kurzen Einführung unterstützend zur Seite steht. Um das Benutzerhandbuch praxisnah und verständlicher zu gestalten, wurden Screenshots der Prototypen beigefügt. Die dargestellten Datensätze wurden zu Testzwecken im System eingefügt. Sie weisen daher keinen Anspruch auf Vollständigkeit und Konsistenz auf. Den folgenden Handbüchern zur Smartphone App, dem webbasiertem Dashboard und der Verkehrssimulation wird, wie im Kapitel 7 beschrieben, eine laufende Umgebung aller Komponenten vorausgesetzt.

8.1 Android Applikation

Die App wurden für die Kunden des Systems erstellt. Sie soll eine einfach und zugleich schnelle Möglichkeit für den Kunden bieten sich ein Taxi zu bestellen. Dazu kann der Kunde sich die Applikation auf seinem Android-Smartphone installieren. Die Installation ist aktuell jedoch nur durch einen Computer und Android Studio möglich. Um die App in vollem Funktionsumfang nutzen zu können, muss der Nutzer der App bestimmte Berechtigungen einräumen. Folgende Berechtigungen müssen der App gegeben werden:

- Allgemeiner (netzwerkbasierter) Standort
- Genauer Standort (GPS)
- Kontaktdaten lesen
- Ihre Profildaten lesen
- Bekannte Konten erkennen
- Google-Systemkonfiguration lesen
- Internetdaten erhalten
- Netzwerkstatus anzeigen
- Vollständiger Internetzugriff

- Ausgeführte Anwendungen abrufen
- Standby-Modus verhindern

Weiterhin müssen um die App benutzen zu können die *Google Play Services* auf dem Smartphone installiert sein. Beim Start der Applikation wird dies jedoch automatisch geprüft. Sollte dies nicht der Fall sein, wird der Nutzer durch ein Pop-up darüber informiert, dass eine Installation der *Google Play Services* notwendig ist und direkt zu dem entsprechenden Download-Link geleitet.
Startet der Nutzer die Applikation auf seinem Smartphone sieht er zunächst den in Abbildung 34 dargestellten Startbildschirm. Durch einfaches Antippen des Bildschirms wird der Nutzer weitergeleitet zum „Einloggen und Registrieren“-Bildschirm. Wie im Kapitel 8.1.1 näher beschrieben, kann sich der Nutzer dort, unter der Bedingung, dass er bereits registriert ist, einloggen. Nach dem einloggen werden die Nutzerdaten von der App gespeichert und beim nächsten Öffnen der App ist ein erneutes Einloggen nicht mehr notwendig und der „Einloggen und Registrieren“-Bildschirm wird übersprungen.

Abbildung 34: Startbildschirm der App

8.1.1 Einloggen und Registrieren

Da beim erstmaligen Öffnen der App noch keine Daten des Nutzer in der App gespeichert sind wird ihm der Bildschirm zum Einloggen und Registrieren angezeigt. Ein Bild dieses Bildschirm ist in Abbildung 35 dargestellt. Wenn der Nutzer bereits ein Konto besitzt, kann er sich einfach mit seiner E-Mail-Adresse und seinem Passwort anmelden und gelangt dann zum nächsten Bildschirm.

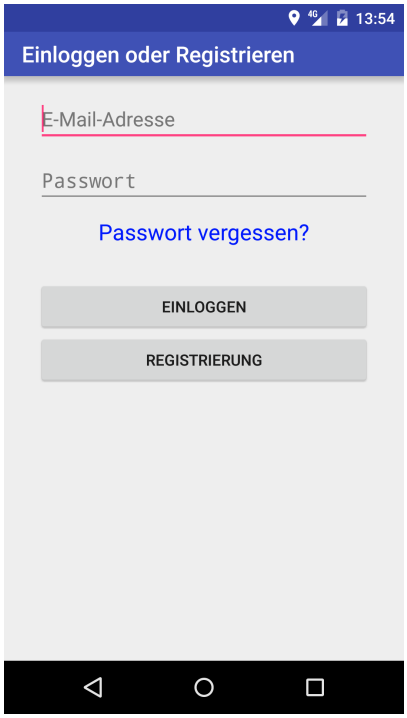
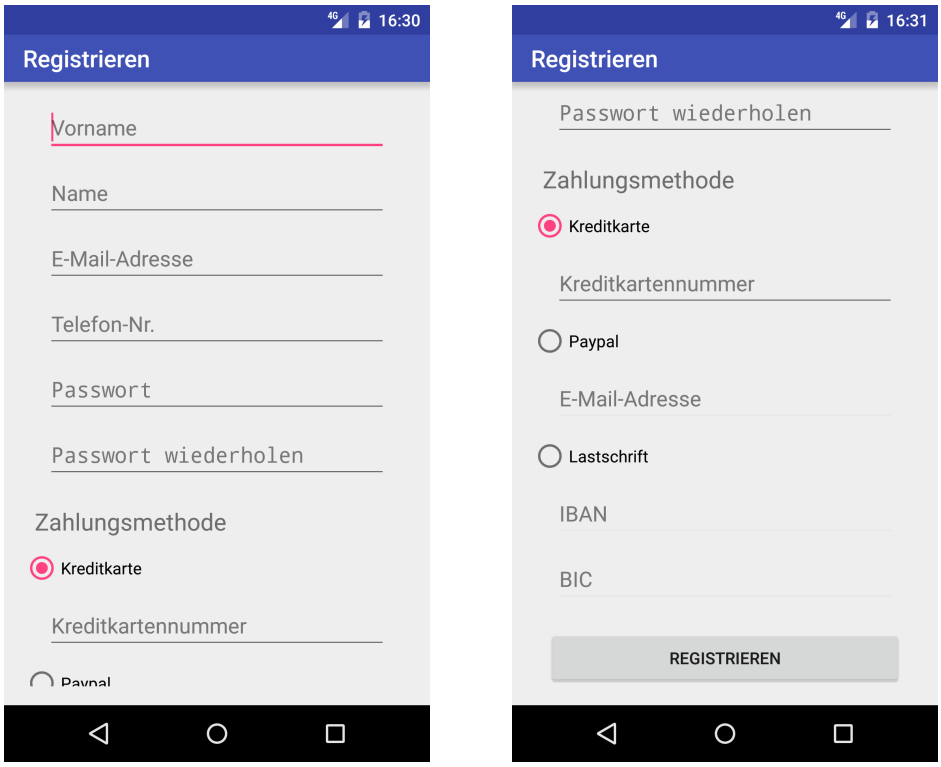


Abbildung 35: Login-Bildschirm der App

Hat der Nutzer jedoch noch kein Konto klickt er auf dem „Einloggen“-Bildschirm auf den Button „Registrieren“ um zum Bildschirm für die Registrierung zum kommen. Dieser sieht wie in der Abbildung 36 dargestellt aus.
Dort trägt der Nutzer nun sein Anmeldedaten ein. Notwendig sind die folgenden Daten:

- Vorname
- Name



(a) Obere Hälfte der Registrierung (b) Untere Hälfte der Registrierung

Abbildung 36: Bildschirm zur Registrierung der App

- E-Mail-Adresse
- Telefonnummer
- Passwort
- Wiederholung des Passworts
- Zahlungsart und -informationen

Der Nutzer hat die Wahl aus den drei Zahlungsarten *Kreditkarte*, *Paypal* und *Lastschrift*. Zur Anmeldung muss der Nutzer jedoch lediglich eine davon sowie die entsprechenden Zahlungsinformationen angeben.
Hat der Nutzer alle seine Daten eingetragen, kann er sich durch den Button „Registrieren“ mit seinen Daten registrieren und erhält eine Bestätigung über den

Erfolg. Anschließend ist der Nutzer in der Lage sich über den bereits beschriebenen „Einloggen“-Bildschirm einzuloggen. Danach ist der Nutzer in der Lage Fahrtinformationen einzugeben. Dieser Vorgang wird im folgenden Kapitel 8.1.2 beschrieben.

8.1.2 Auswahl von Abhol- und Zielort

Um mithilfe der App eine Taxifahrt zu buchen muss der Kunde im ersten Schritt seinen Abholort angeben. Dazu steht ihm der in Abbildung 37 darstellte Bildschirm zur Verfügung.

Abbildung 37: Auswahl des Abholortes in der App

Zur Auswahl des Abholortes hat der Nutzer vier verschiedene Möglichkeiten zur Auswahl. Der Nutzer kann den in der Karte angezeigten Marker an den Ort ziehen an dem er gerne abgeholt werden möchte, dabei ist anzumerken, dass der Marker standartmäßig an der aktuellen Position des Nutzers gesetzt ist. Weiterhin kann der Nutzer den Marker auf seine aktuelle Position setzen indem er den Button in der oberen rechte Ecke der Karte bestätigt. Außerdem hat der Nutzer die Möglichkeit

seinen Abholort in das Suchfeld unter der Karte einzutragen, wodurch automatisch auch der Marker an den eingebenen Ort gesetzt wird. Während der Suche werden dem Nutzer Vervollständigungsvorschläge angezeigt (siehe Abbildung 38).

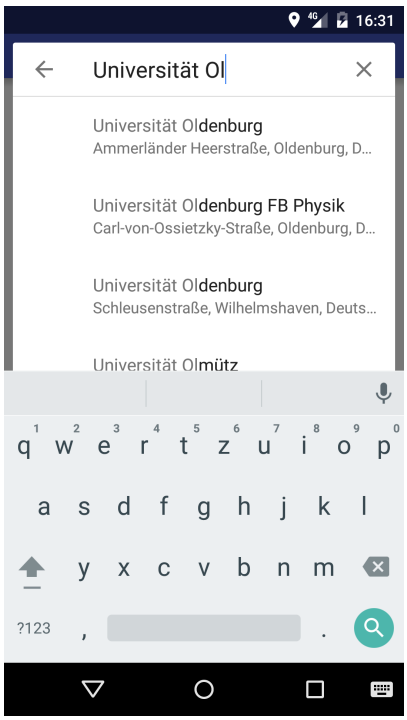


Abbildung 38: Auswahl des Abholortes durch die Suchfunktion in der App

Die letzte Möglichkeit steht dem Nutzer lediglich zur Verfüngung, wenn er bereits Adressen als seine Favoriten gespeichert hat. Ist dieser der Fall ist das Auswahlfeld in der oberen linken Ecke des Bildschirms aktiviert. Klick der Nutzer auf dieses Feld, so werden ihm seine gespeicherten Adressen angezeigt (siehe Abbildung 39). Wählt der Nutzer eine dieser Adressen per Klick aus, so wieder Marker auf der Karte automatisch an den korrekten Ort gesetzt. Der Nutzer ist jederzeit in der Lage eine Adresse als Favoriten zu speichern indem er auf den Stern auf dem Bildschirm klickt.

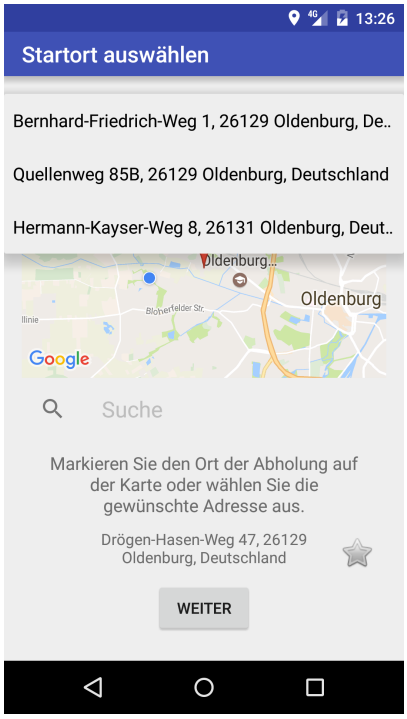


Abbildung 39: Auswahl des Abholortes durch die Favoritenfunktion in der App

Hat der Kunde einen Abholort ausgewählt und klickt auf den „Weiter“-Button, so wird der Abholort gespeichert und der Kunde gelangt zur Auswahl des Zielortes. Diese Auswahl erfolgt analog zur Auswahl des Abholortes und der Nutzer hat die selben Auswahlmöglichkeiten (siehe Abbildung 40). Klickt der Nutzer anschließend auf „Weiter“, so wird auch der Zielort gespeichert und er gelangt zu einem Bildschirm wo er weitere Informationen angeben kann. Dieser Bildschirm wird im Kapitel 8.1.3 näher erläutert.

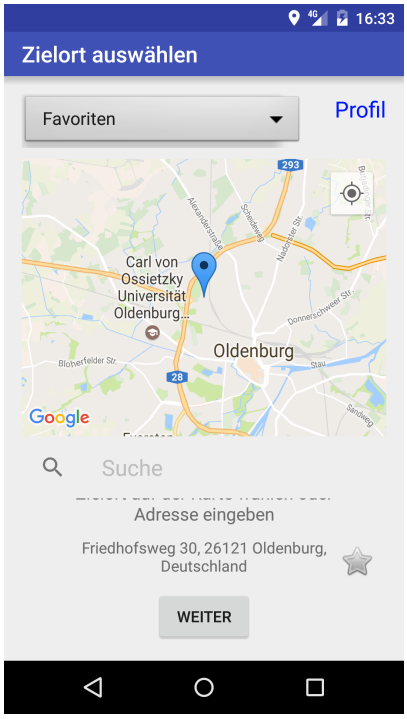


Abbildung 40: Auswahl des Zielortes in der App

8.1.3 Fahrtinformationen eingeben

Nachdem der Kunde seinen Abhol- und Zielort ausgewählt hat, muss er weitere Informationen angeben, beispielsweise Abholzeitpunkt, Personenanzahl usw. Zu diesem Zweck benutzt der Kunde den in Abbildung 41 aufgezeigten Bildschirm.

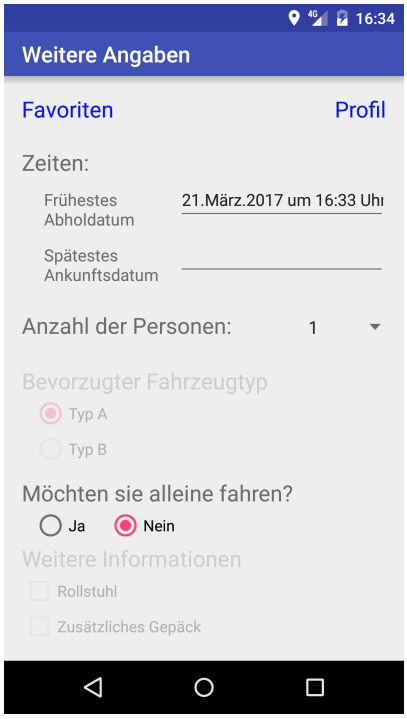


Abbildung 41: Bildschirm zur Angabe weiterer Informationen in der App

Dort kann der Nutzer zunächsten, durch den in Abbildung 42 dargestellten Datepicker, einen Abhol-, einen Ankunftszeitpunkt oder beiden auswählen.

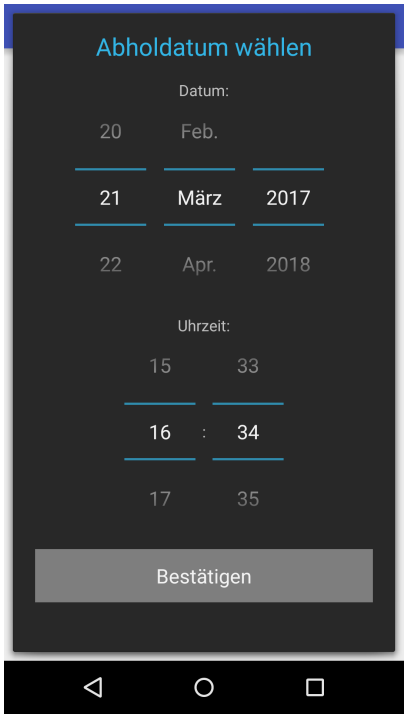


Abbildung 42: Datepicker zur Auswahl der Zeitpunkte in der App

Weiterhin muss der Nutzer noch angeben mit wie vielen Personen erfahren möchte und, ob er allein fahren möchte oder ob es für ihn in Ordnung ist wenn mehrere Aufträge zusammengelegt und in einem Taxis bearbeitet werden. Sind diese Angaben ausgefüllt, kann er den Auftrag abschicken und somit ein Taxi bestellen.

8.1.4 Route

Sobald der Nutzer ein Taxi bestellt hat wird er automatisch auf einen Wartebildschirm geleitet. der Ladebildschirm sieht wie in Abbildung 43 dargestellt.

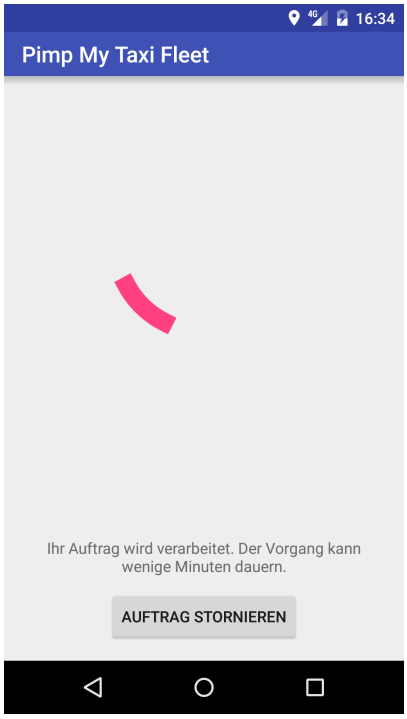


Abbildung 43: Wartebildschirm während der Routenberechnung in der App

Während der Nutzer auf dem Wartebildschirm ist hat er jederzeit die Möglichkeit seinen Auftrag zu stornieren. Dazu kann er entweder den „Auftrag stornieren“-Button oder den „Zurück“-Button benutzen. Bevor er den Auftrag stornieren kann wird er durch einen Bestätigungsdialog um Bestätigung der Stornierung gebeten (siehe 44).

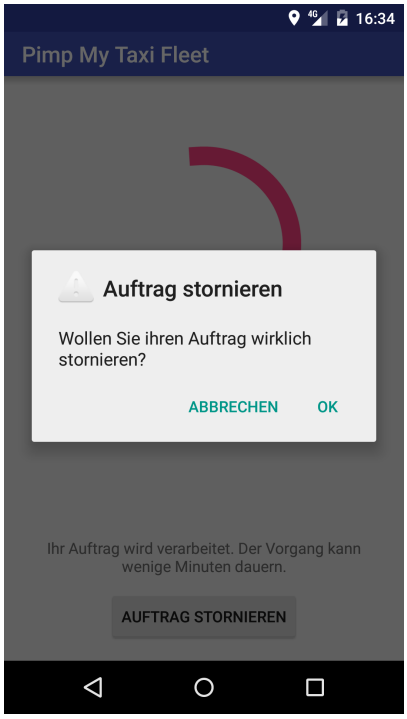


Abbildung 44: Stornieren eines Auftrags in der App

Außerdem wird in der Zeit in der der Nutzer auf dem Wartebildschirm ist die Routenberechnung vom System vorgenommen. Ist die Route berechnet, wird die App auf dem Smartphone des Kunden via Firebase Cloud Message darüber benachrichtigt, dass eine neue Route verfügbar ist. Die App empfängt daraufhin die Route und leitet den Nutzer auf den nächsten Bildschirm weiter. Dieser sieht wie in Abbildung 45 beispielhaft dargestellt aus. Auf dem Bildschirm kann der Nutzer seine Route erkennen, dabei wird der Startpunkt durch einen roten Marker, der Zielort durch einen blauen Marker und eventuelle Zwischenziele durch grüne Marker dargestellt. Weiterhin kann der Kunde noch die aktuelle Position seines Taxis anhand des schwarzen Taxi-Symboles auf der Karte erkennen.

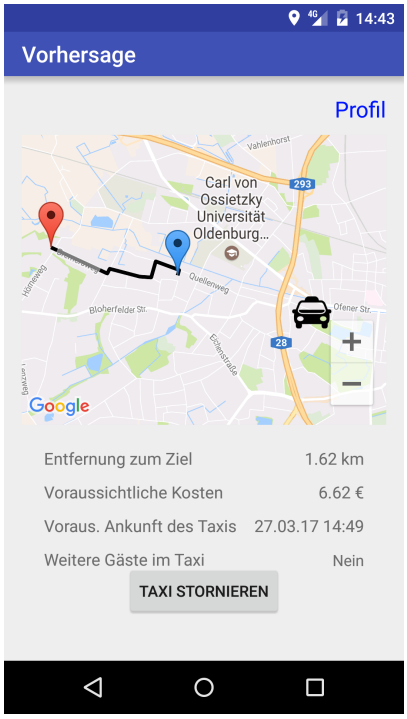
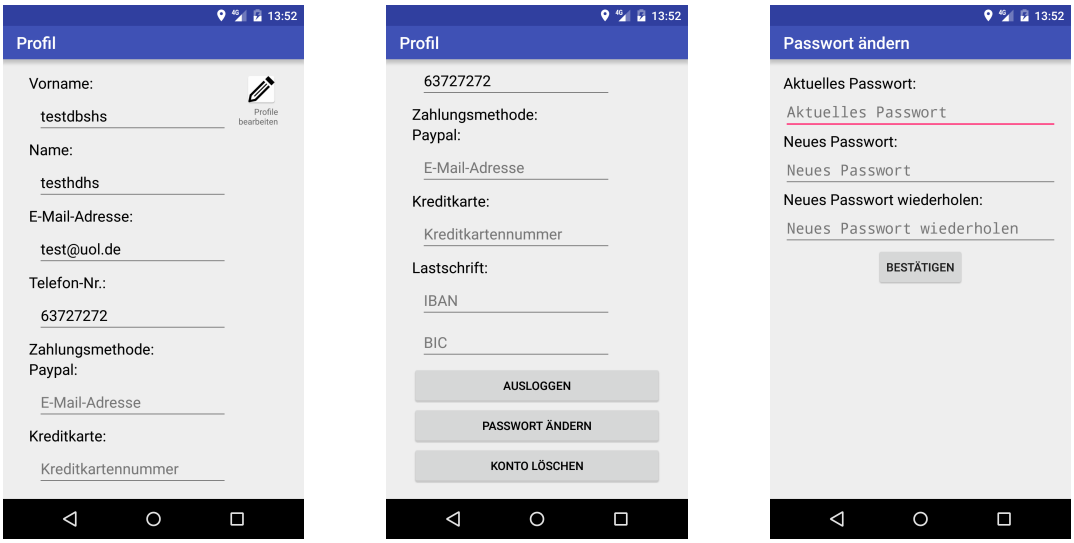


Abbildung 45: Route und weitere Informationen zur Fahrt in der App

8.1.5 Profil

Sofern der Nutzer eingeloggt ist, hat er dauerhaft die Möglichkeit auf sein Profil zu gehen. Im Profil kann er sich seine persönlichen Daten, wie beispielsweise Namen, E-Mail-Adresse usw., anzeigen lassen. Das Profil sieht aus wie in der Abbildung 46 dargestellt. Durch den „Profil bearbeiten“-Button ist der Nutzer in der Lage ausgewählte Informationen in seinem Profil zu verändern.



(a) Obere Hälfte des Kundenprofils

(b) Untere Hälfte des Kundenprofils

(c) Bildschim zum Ändern des Passworts

Abbildung 46: Kundenprofil in der App

Am unteren Ende des Profils befinden sich drei Button. Über den obersten Butten, den „Ausloggen“-Button, kann sich der Nutzer aus der App abmelden und gelangt dadurch zurück zum Anmeldebildschirm (siehe Abbildung 35) und kann sich neu anmelden. Über den „Passwort vergessen“-Button gelangt der Nutzer zu einem neuen Bildschirm, auf dem er ein neues Passwort für seinen Account setzen kann (siehe Abbildung 46). Über den „Konto löschen“-Button kann der Kunde sein Konto komplett löschen und gelang danach ebenfalls zurück zum Anmeldebildschirm (siehe Abbildung 35). Für eine erneute Anmeldung muss sich der Nutzer zunächst neu registrieren.

8.2 Dashboard

Das webbasierte Dashboard soll dem Taxi Management aktuelle Informationen zu den eingesetzten Fahrzeugen sowie den relevanten Einflüssen auf diese geben. Das Dashboard richtet sich daher ausschließlich an das Personal des Taxiunternehmens. Die Anwendung ist webbasiert und daher unabhängig vom Betriebssystem des verwendeten Endgeräts. Je nach Browser-Typ und -Version kann es jedoch zu Abweichungen in der Darstellung kommen. Es empfiehlt sich daher, den Einsatz auf Mozilla Firefox mit einer Bildschirmauflösung von mindestens 1024 x 767 Pixel zu beschränken und keine mobilen Endgeräte mit kleinerem Screen zu verwenden. Da Internet-Browser vom Benutzer konfigurierbar sind, wird nachfolgend aufgeführt, welche Einstellungen sich auf die Anwendung auswirken können.

- Cookies: Der Einsatz von Cookies wird empfohlen, um Inhalte zwischenspeichern zu können. Werden Cookies ausgeschaltet, kann dies zum Verlust von bereits geladene Inhalte führen.
- Local Storage: Der Einsatz von LocalStorage wird empfohlen, um Inhalte zwischenspeichern zu können. Werden LocalStorage ausgeschaltet, kann dies zum Verlust von bereits geladene Inhalte führen.
- Pop-up-Blocker: Pop-ups werden an wenigen Stellen für ein Benutzerfeedback verwendet.
- JavaScript: Der Einsatz wird für AngularJS zwingend vorausgesetzt.
- AdBlocker: Die Webseite greift z.B. für die Darstellung von Diagrammen auf externe Dienste wie Google Charts zu. Ein Blockieren dieser Inhalte, kann zu einem unerwünschten Erscheinungsbild führen und den Funktionsumfang einschränken.

Das Dashboard kann unter **http://localhost:8080/pgtaxi-spring-mvc/#/** [47] aufgerufen werden.

[47] Hier müssen Sie ggf. IP-Adresse und Port anpassen. "Localhost"kann ausschließlich nach einer lokalen Installation verwendet werden.

8.2.1 Menüleiste

Die Oberfläche des Dashboards besteht konsistent aus einer statischen Menüleiste, mit zwei Ebenen und dem jeweiligen folgenden Seiteninhalt. Alle wesentlichen Funktionen sind über die Menüleiste erreichbar. Die obere Menüebene enthält dabei die objektbezogenen Rubriken, wie Taxis, Aufträge und Kunden. Die untere Menüebene stellt über runde Buttons wichtige Funktionen zur Anzeige und Verwaltung bereit. Grundsätzlich teilt sich der Seiteninhalt in eine Karten oder Grafik auf der linken Seite und eine Auflistung der Einträge sowie detailliertere Informationen zu einem ausgewählten Eintrag auf der rechten Seite.

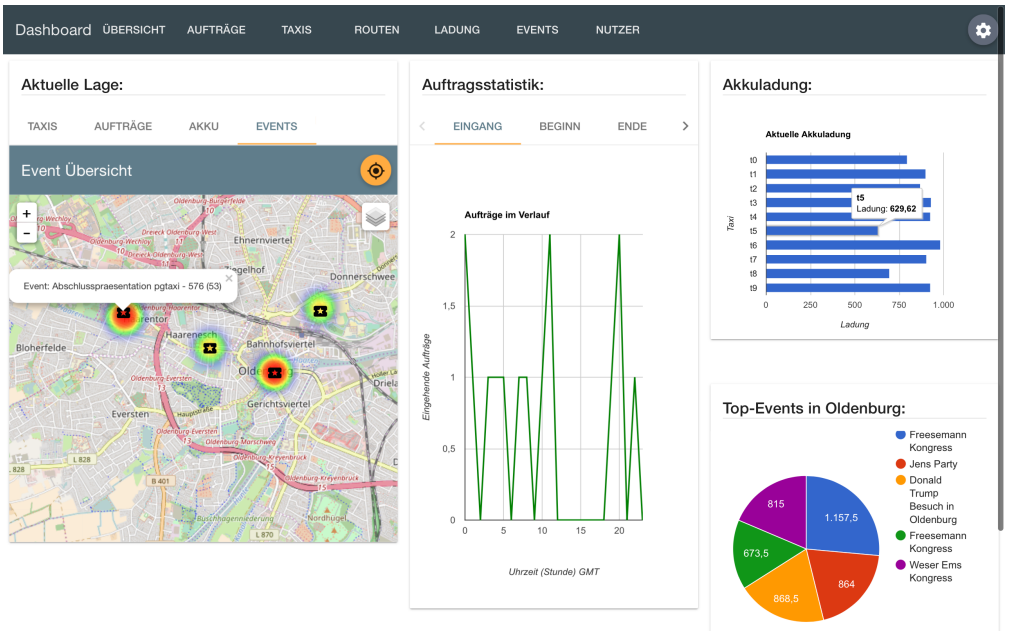


Abbildung 47: Übersichtsseite des Dashboards

8.2.2 Übersichtsseite

Die Home- bzw. Übersichtsseite wie in Abbildung 47 stellt eine Zusammenfassung aus den Übersichtsdaten für Events, Taxis, Ladestationen, und Kundenaufträgen dar. Sie wird bei Aufruf des Basispfades *../pgtaxi-spring-mvc/#/* angezeigt. So kann der Nutzer alle wichtigen Informationen auf einen Blick erhalten. Hierzu gehören die Heatmaps für Kundenaufträge und Veranstaltungen, eine Kartenansicht der aktuellen Taxi-Positionen und Diagramme, wie die aktuelle Ladung der

Taxis und die Top-Veranstaltungen in Oldenburg, die mit aktuellen Daten aus Odysseus versorgt werden. Aus der MySQL-Datenbank werden Kunden und Kundenaufträge in verschiedenen Diagrammen sowie Standorte der Ladestationen auf einer Karte visualisiert. Die Kartenansichten und die Diagramme sind in Tabs unterteilt, zwischen denen navigiert werden kann. Die Elemente der Übersichtsseite können über *Einstellungen* konfiguriert werden. Die Anzahl der Events im Diagramm *Top-Events in Oldenburg* kann über einen Slider konfiguriert werden, sodass das Diagramm entsprechen angepasst wird.

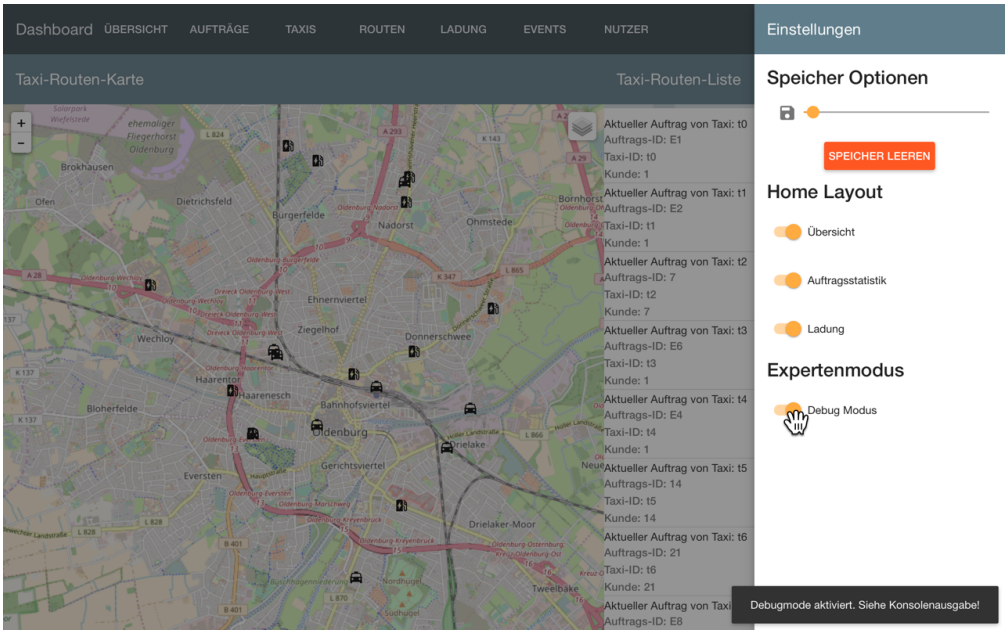


Abbildung 48: Einstellungen des Dashboards

8.2.3 Einstellungen

Über den Einstellungen-Button öffnet sich eine Sidebar (Abbildung 48). Dort kann der Nutzer Elemente der Übersichtsseite(/home.html) des Dashboards ein- und ausblenden. Hinzu kann ein Debug-Modus aktiviert bzw. deaktiviert werden, um die Konsolenausgaben im jeweiligen Webbrowser zu steuern. Außerdem ist es möglich, über den Slider die Anzahl der Cookies pro Objekt zu konfigurieren und über den Button *Speicher leeren* den LocalStorage und die Cookies zu leeren.

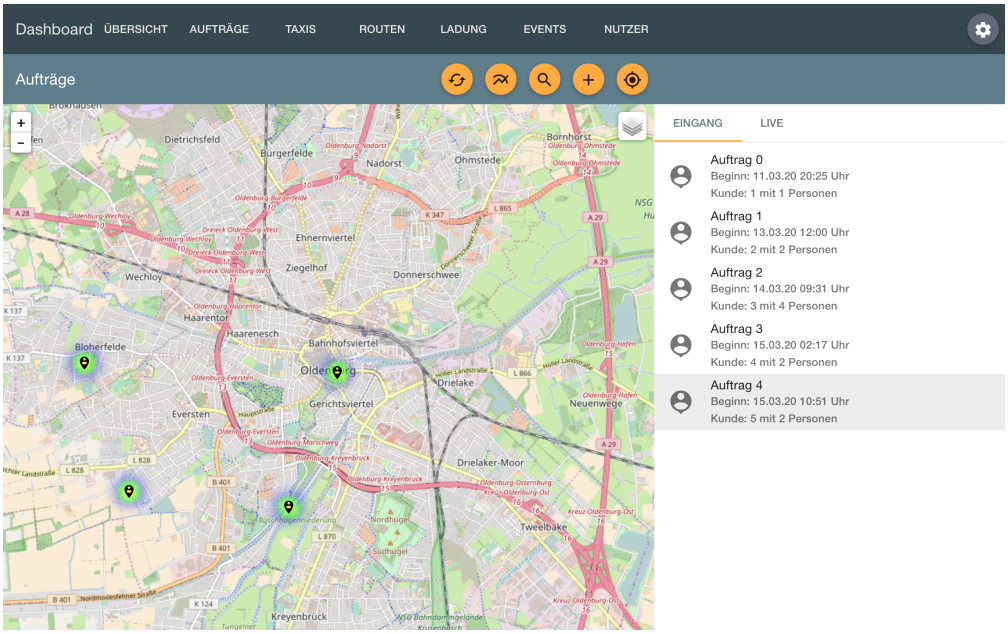


Abbildung 49: Auftrags Übersicht

8.2.4 Auftrags Übersicht

Die Auftrags Übersicht erreicht man über den Menüpunkt *Aufträge* in der Navigationsleiste. Wie in Abbildung 49 zu erkennen ist, beinhaltet die rechte Seite Listenansichten der Kundenaufträge, die sich in die zwei Tabs Eingang und Live unterteilen. Ersterer enthält alle gespeichert Kundenaufträge aus der MySQL-Datenbank. Unter dem Tab *Live-Aufträge* werden hingegen die Echtzeit-Aufträge aus Odysseus angezeigt, die von Websockets empfangen wurden. Die Kartenansicht zeigt mit den *Auftrags-Markern* die Abholorte der einzelnen Aufträge. Eine weitere Kartenebene ist die Heatmap, die aus den *Auftrags-Markern* generiert wird und auf der OSM-Karte visualisiert wird. Die Toolbar bietet mit verschiedenen Buttons folgende Möglichkeiten:

- Mit dem *Daten-Laden-Button* können die Daten aus der MySQL-Datenbank per REST-AJAX-Request neu geladen werden.
- Mit dem *Diagramm-Button* öffnen sich anstatt der Kartenansicht eine Tabansicht in der Diagramme hinterlegt, zur Zeit der Auftragserstellung und der Anzahl der Personen pro Auftrag.

- Mit dem *Karte-Button* schließt sich die Diagramm-Ansicht und switcht zur Kartenansicht.
- Mit dem *Suche-Button* öffnet sich auf der rechten Seite die Suchansicht
- Mit dem *Plus-Button* öffnet sich ein Overlayer, mit dem man Kundenaufträge erstellen kann
- Mit dem *Standort-Button* zentriert sich die Karte wieder mit den Default-Werten

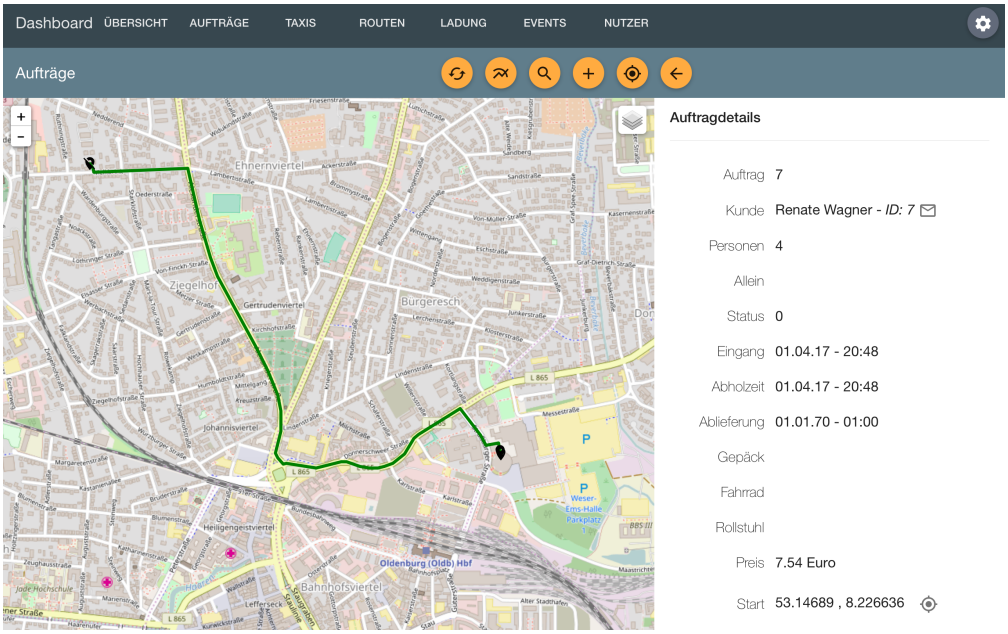


Abbildung 50: Auftrag Details

8.2.5 Auftrags Details

Abbildung 50 zeigt die Detailseiten der Aufträge, die mit einem Klick auf den jeweiligen Auftrag aus der Listenansicht auf der rechten Seite geöffnet werden können. Dort sieht man die jeweiligen Information, die man aus der MySQL-Datenbank erhält. Der Standort der Abholung des jeweiligen Auftrags kann über die *Standort-Buttons* in den Auftragsinformationen auf der Karte angezeigt werden. Die Detailseiten der Live-Aufträge können ebenfalls über die Listenansicht

im Tab *Live-Aufträge* erreicht werden. Mit einem Klick auf einem Listeneintrag öffnen sich auf der rechten Seite alle Auftragsinformationen und auf der Karte wird der Zielort und Abholort des Auftrags sowie die zugehörige Route angezeigt.

8.2.6 Auftrags Diagramme

Die Diagramm-Ansicht ist über dem *Diagramm-Button* in der Toolbar der Übersichtsseite zu erreichen. Dort werden Diagramme zur Auftragserstellung in Bezug auf die Tageszeit und ein Kreisdiagramm, welches die Personenzahl pro Auftrag anzeigt, in verschiedenen Tabs angezeigt.

8.2.7 Auftrags Suchfunktion

Die Suchfunktion erreicht man über dem *Suchen-Button*. Auf der rechten Seite öffnet sich ein Such-Input-Feld, wo mit einer Auftrags- oder Kunden-ID oder der Personenanzahl nach Aufträgen gesucht werden kann.

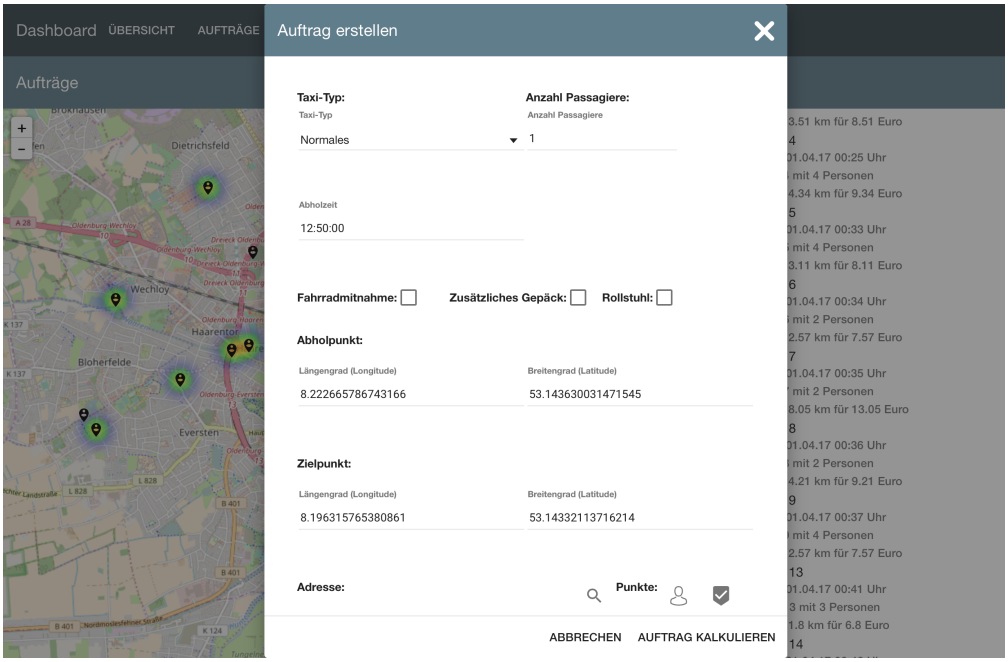


Abbildung 51: Auftrag Hinzufügen

8.2.8 Auftrag Hinzufügen

Neben dem Auslesen der Aufträge bietet das Dashboard auch die Möglichkeit, neue Aufträge zu erstellen. Hierzu können, wie in Abbildung 51 zu sehen, alle Informationen mit Unterstützung von Karte und Suchfunktionen angegeben werden.

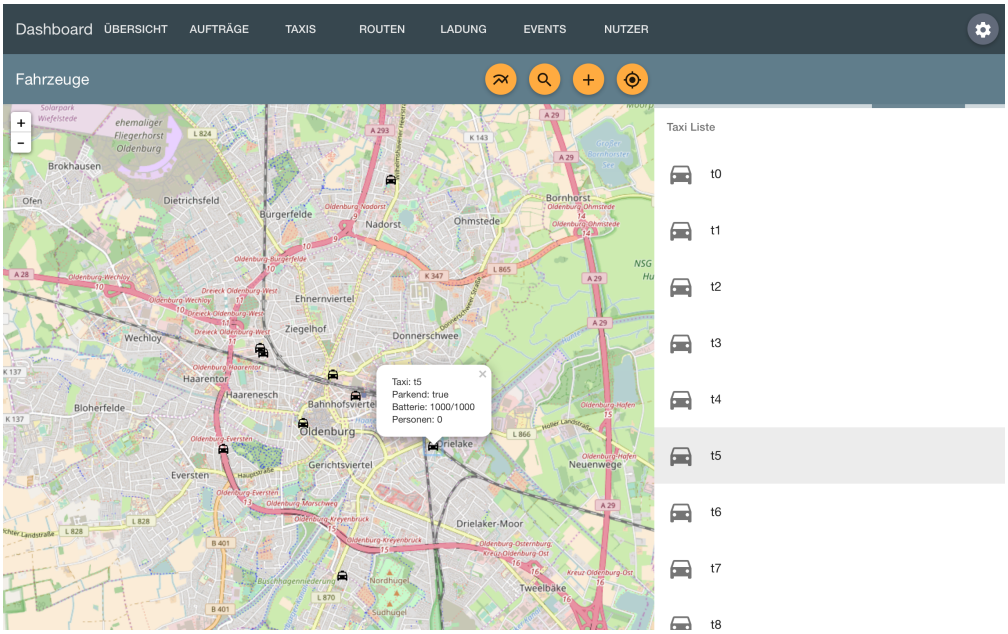


Abbildung 52: Taxi Übersicht

8.2.9 Taxi Übersicht

Wie auf der Abbildung 52 zu erkennen ist, werden unter dem Menüpunkt *Taxis* alle Taxis aufgelistet und ihr Standort auf der nebenstehenden Karte angezeigt. Die Menüpunkte der zweiten Ebene ermöglichen weitere Anpassungen auf dieser Seite, über die die Icons beim Fokussieren mit dem Cursor detailliertere Auskunft in Textform geben. So kann zwischen der Karte und Diagrammen, wie auch in der folgenden Grafik abgebildet, gewechselt werden. Weitere Button ermöglichen die Suche nach bestimmten Taxis, das Hinzufügen neuer Fahrzeuge und ein zentrieren der Karte auf Oldenburg. Der dargestellte Kartenausschnitt kann mit Halten der linken Maustaste und gleichzeitiger Mausbewegung verändert werden. Der Zoom-Faktor ist über die *Plus/Minus Buttons* der LeafletJS-Funktionalität auf der Karte

anpassbar. In der Karte angezeigte Markierungen (*Marker*) können angeklickt werden, um weitere Informationen innerhalb von Tooltips über die jeweiligen *Marker* zu erhalten.

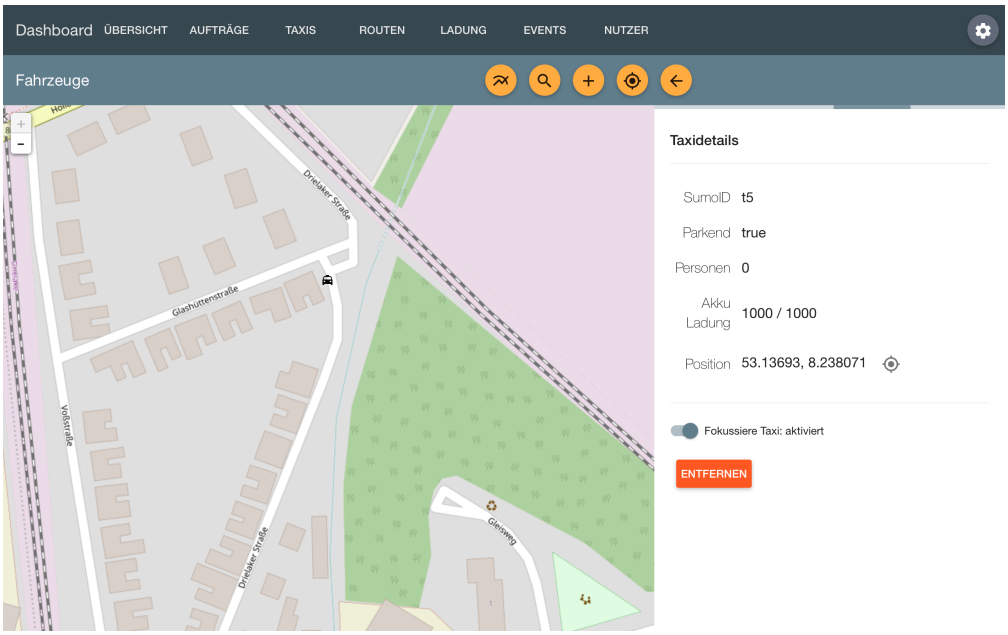


Abbildung 53: Taxi Details

8.2.10 Taxi Details

Die Auswahl eines Taxis aus der Auflistung auf der rechten Seite, fokussiert das Fahrzeug auf der Karte und führt zu einer Detailansicht (Abbildung 53) mit ausführlichen Informationen. Diese Daten werden ebenso wie die Liste in Echtzeit aktualisiert. Die Aktualität der Daten wird durch die Ladeleiste unterhalb des Menüs symbolisiert. Das untere Menü wurde in dieser Anzeige um einen *Zurück-Button* ergänzt, der zurück zur Liste führt. Um das Taxi auch bei aktualisiertem Aufenthaltsort nicht aus den Augen zu verlieren, kann über den Schalter *Fokussiere Taxi* die Karte auf das Taxi dauerhaft zentriert werden. Manche Inhalte, wie bspw. Ladestationen oder Benutzer, sind vom Dashboard Nutzer veränderbar. In diesen Fällen führt ein *Ändern Button* zu einer weiteren Detailseite, die dargestellte Inhalte editierbar macht. Vorgenommene Veränderungen müssen über ein *Aktualisieren Button* bestätigt werden. Der nebenstehende *Abbrechen Button* ver-

wirft hingegen vorgenommene Änderungen und führt zurück zur Liste.

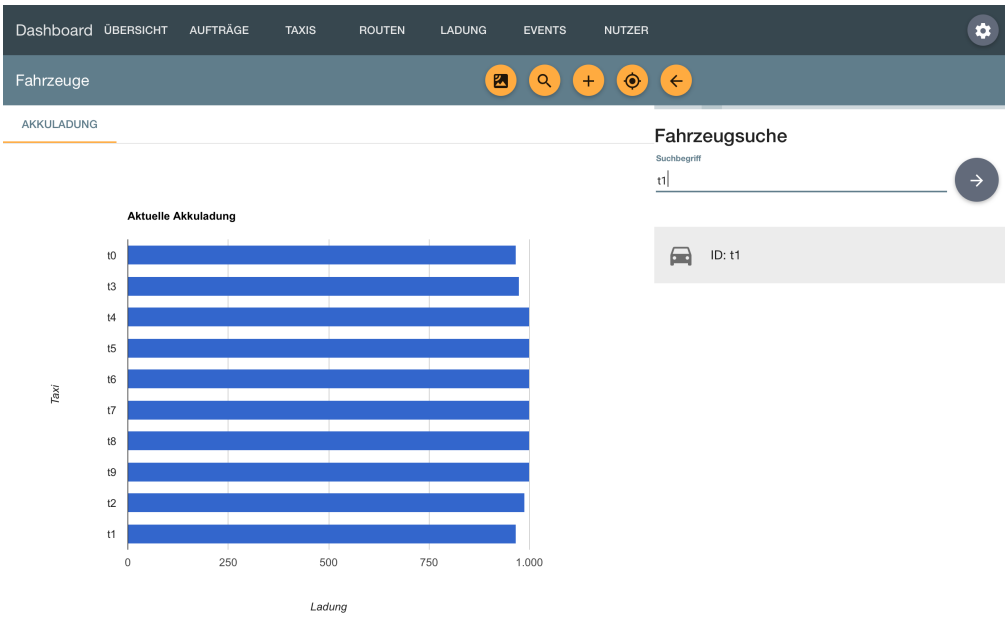


Abbildung 54: Taxi Akkuladungen

8.2.11 Taxi-Diagramme

Diagramme, wie in Abbildung 54 die Akku-Kapazität aller eingesetzten Taxis, sollen dem Nutzer einen schnellen Eindruck über aktuelle Zahlen geben. Über den oben stehenden Reiter, kann meist zwischen verschiedenen Diagrammtypen gewechselt werden. Weitere Details zu den Datenreihen können mit Mausklick über diese angezeigt werden. Da einige Diagramme mit in Echtzeit eingehenden Daten erstellt werden, kann es einen Augenblick in Anspruch nehmen, bis ein erstes Diagramm erscheint.

8.2.12 Taxi-Suchfunktion

Über den *Lupen Button* in der zweiten Menüleiste, kann die in Abbildung 54 dargestellte Suchfunktion erreicht werden. Diese ermöglicht dem Nutzer ein schnelleres Auffinden aus der jeweiligen Liste. Eingegebene Suchwörter, die mit den Kriterien aus der unterstehenden Erklärung übereinstimmen müssen, können mit der

Enter-Taste oder dem *Pfeil Button* gesucht werden. Einträge aus der untenstehenden Liste können zuvor angewählt werden, um eine Detailansicht zu erhalten.

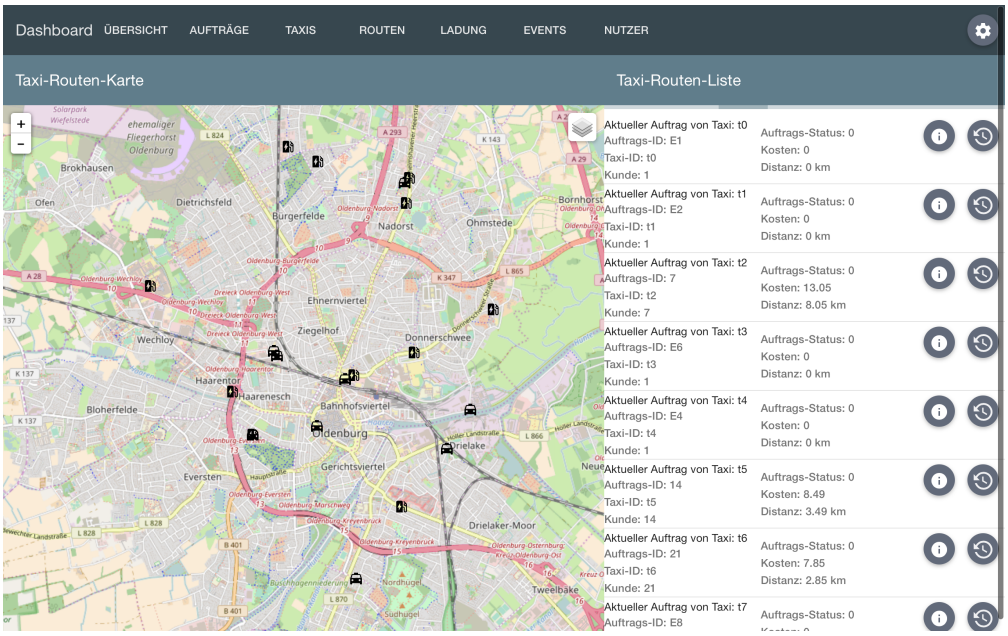


Abbildung 55: Routen Übersicht

8.2.13 Routen Übersicht

Um auf die Routen-Übersichtsseite (Abbildung 55) zu gelangen, muss in der Navigation auf den Menüpunkt *Routen*gewählt werden. Auf der Routen-Übersichtsseite werden auf der rechten Seite die aktuell vergebenen Taxi-Routen für jedes erstellte Taxi aufgelistet, die zuletzt Live über die Websockets empfangen wurden. Innerhalb der Liste kann zum einen über den *Info-Button* die aktuelle Route angezeigt oder über den *History-Button* des jeweiligen Taxis alle älteren Taxi-Routen aufgelistet werden, die Live über die Websockets empfangen. In der Kartenansicht auf der linken Seite werden durch *Marker* die Ladestationen aus der MySQL-Datenbank angezeigt sowie die aktuellen Positionen der Taxis aus den Websockets. In der Karte angezeigte Markierungen (*Marker*) können angeklickt werden, um weitere Informationen innerhalb von Tooltips über die jeweiligen *Marker* zu erhalten. Die Ebenen der Kartenansicht können über den *Layer-Button* oben rechts in der Karte ein- und ausgeschaltet werden. Somit können *Marker-Gruppen*, wie z.B.

die *Marker* der Ladestationen vollständig ein- bzw. ausgeblendet werden.

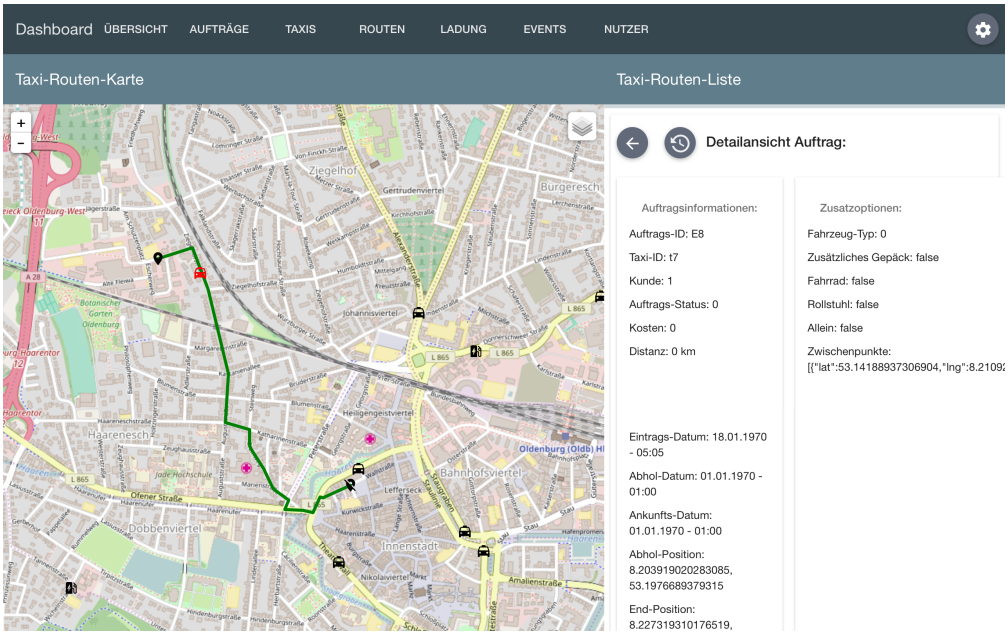


Abbildung 56: Routen Details

8.2.14 Routen Detailseite

Auf die Routen-Detailseiten (Abbildung 56) gelangt man über jeweiligen *Info-Buttons* in Taxi-Routen-Liste oder der jeweiligen History-Liste der einzelnen Taxis. Die Taxi-Detailseiten werden auf der rechten Seite angezeigt und beinhalten alle verfügbaren Informationen zur Taxi-Route die wir aus Odysseus erhalten. Zusätzlich werden in der Kartenansicht die Routen in einem Pfad visualisiert. Zur Route gehören noch die Start- und Endpunkte sowie Zwischenpunkte, die aus den Abhol- und Lieferpunkten der Kundenaufträge bestehen. Sobald die Detailansicht einer Taxi-Route geöffnet wird, erscheint ein roter *Taxi-Marker* auf der Karte, der das betroffene Taxi visualisiert und somit klarer von anderen *Taxi-Markers* differenziert werden kann. Von der von der Routen-Detailseite kann man über den *Zurück-Button* auf die Listenansicht der Taxi-Routen zurückgelangen oder über den *History-Button* auf die History-Ansicht des gewählten Taxis.

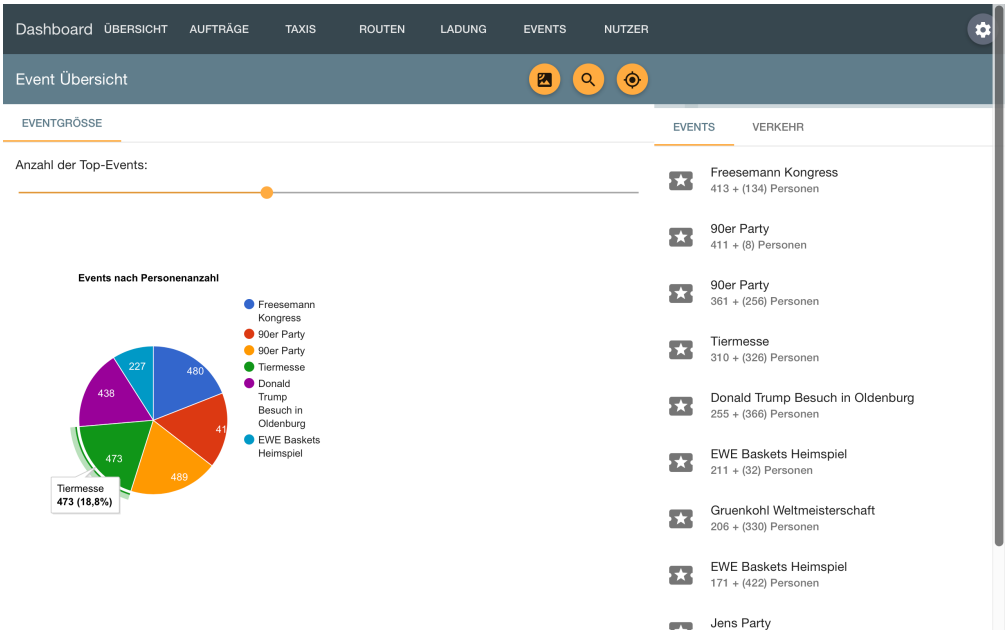


Abbildung 57: Event Liste und Diagramm

8.2.15 Events-Übersichtsseite

Die Eventübersichtsseite (Abbildung 57) erreicht man über den Menüpunkt *Events* in der Navigationsleiste. In der Übersichtseite findet man auf der rechten Seite die Listenansichten für Veranstaltungen und Verkehrsevents (Straßensperrungen oder Geschwindigkeitsbegrenzungen), die in Tabs unterteilt sind. Die Daten der Listenansicht werden in Echtzeit aus den Websockets bezogen und lokal in den Cookies gespeichert. Auf der linken Seite befindet sich die Kartenansicht auf der die Standorte der Veranstaltungen und Verkehrsevents in Form von *Markern* visualisiert werden. Auf der Karte wird zusätzlich aus den Teilnehmerzahlen der Veranstaltungen eine Heatmap generiert. Oben rechts in der Kartenansicht sind die Ebenen-Optionen eingeblendet, wo man die *Marker* auf der Karte filtern kann. In der Toolbar über der Kartenansicht kann über folgende Buttons folgende Funktionen auslösen:

- Mit dem *Diagramm-Button* öffnen sich anstatt der Kartenansicht eine Tabansicht in der Diagramme hinterlegt, zur Zeit der Auftragserstellung und der Anzahl der Personen pro Auftrag.

- Mit dem *Karte-Button* schließt sich die Diagramm-Ansicht und switcht zur Kartenansicht
- Mit dem *Standort-Button* zentriert sich die Karte wieder mit den Default-Werten

8.2.16 Events-Detailseite

Die Event-Detailseiten erreicht man über einen Klick auf einem Listenelement. Die Detailseite zeigt weitere Informationen zum zu den Events auf der rechten Seite an und in der Kartenansicht wird der Standort des Events angezeigt.

8.2.17 Events-Diagramme

Über den *Diagramm-Button* in der Toolbar öffnet sich die Diagrammansicht an Stelle der Kartenansicht. Dort wird ein Kreisdiagramm der Top-Events in Oldenburg angezeigt, also die Veranstaltungen mit den meisten Teilnehmern.

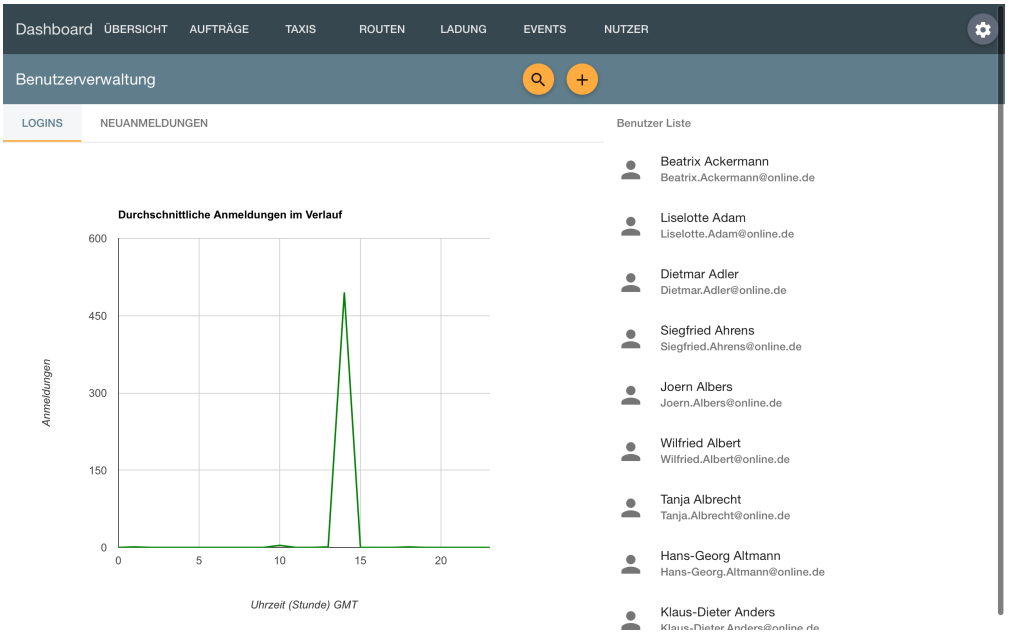


Abbildung 58: Kunden Übersicht

8.2.18 Kunden Übersicht

Die Kundenübersichtsseite erreicht man über den Menüpunkt *Kunden* in der Navigationsleiste. Die Übersichtsseite teilt sich auf unter einer Diagrammansicht auf der man sich die Anzahl der Logins/Neuanmeldungen in Bezug auf den Monatstag ansehen. Auf der rechten Seite befindet sich die Listenansicht der Kundendaten. Die Daten werden per REST-Schnittstelle aus der MySQL-Datenbank bezogen. In der Toolbar über der Diagrammansicht sind folgende Buttons mit folgenden Funktionen hinterlegt:

- Mit dem *Hinzufügen-Button* öffnet sich ein Formular mit dem ein Kunde hinzugefügt werden kann
- Mit dem *Suche-Button* wird die Suchansicht anstatt der Kundenliste angezeigt

8.2.19 Kunden Details

Die Kunden-Detailseiten erreicht man über einen Klick auf einem Listenelement. Die Detailseite zeigt weitere Informationen zum zu den Kunden.

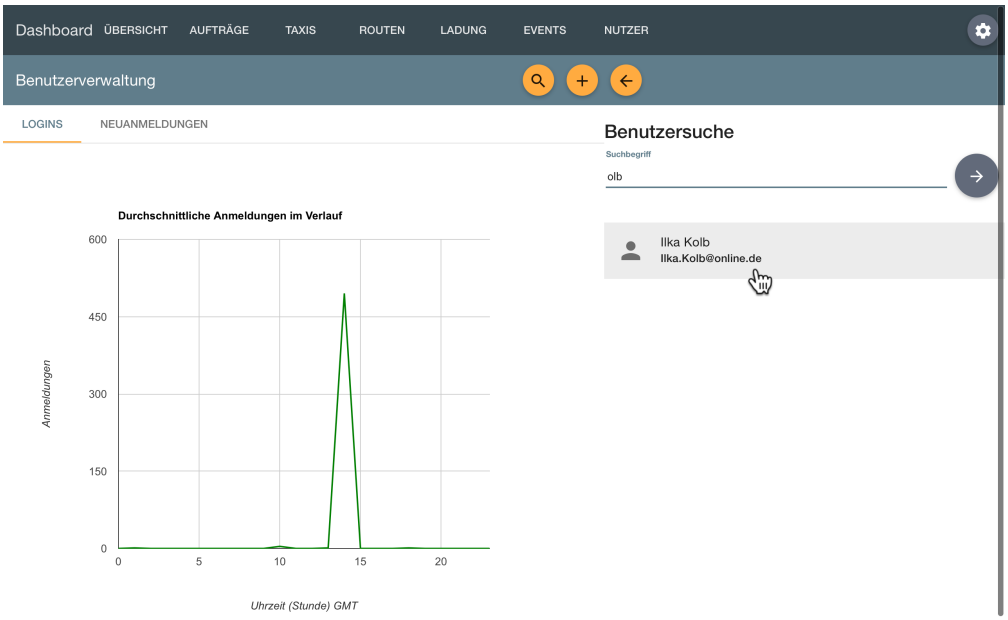


Abbildung 59: Kunden Suchfunktion

8.2.20 Kunden Suchfunktion

Die Kunden-Suchfunktion erreicht man über einen Klick auf dem *Suchen-Button*. Auf der rechten Seite öffnet sich ein Such-Input-Feld, in dem man mit einer E-Mail Adresse oder dem Kundennamen die gewünschten Kunden suchen kann.

8.2.21 Kunden Hinzufügen

Die Kunden-Hinzufügen erreicht man über einen Klick auf dem *Hinzufügen-Button*. Auf der rechten Seite öffnet sich ein ein Formular, in dem man mit den gewünschten Daten den Kunden erstellen kann.

8.3 Verkehrssimulation

Zur Konfigurierung der Simulation wurde im Java Projekt eine Konfigurierungsdatei erstellt, die vom Server Administrator angepasst werden kann. Diese ist im Ordner config zu finden und heißt StaticParams.java. In der Datei gibt es verschiedene Abschnitte, die für unterschiedliche Komponenten verantwortlich sind. Im folgenden werden die einzelnen Möglichkeiten zur Anpassung erläutert.
Bei Verwendung des GUI sind folgende Aktionen relevant.

- Paths: Es können die Pfade angepasst werden.
- GUI: Durch den Boolean Wert lässt sich die GUI an- und ausschalten.
- Traffic Generator: Es lassen sich die Longitude und Latitude Grenzen für den Generator anpassen.
- General Sumo: Die minimale und maximale Anzahl von neuen Fahrzeugen pro Zeitschritt lässt sich anpassen. Zudem kann die Vorlaufzeit angepasst werden.
- Taxi Data: Anpassen der Adresse, Port und Interval zum Senden der Daten.
- Route Receiver: Anpassen der Adresse und Port zum Empfangen der Routen.
- Order Generator: Es lässt sich das Interval zum generieren von neuen Aufträgen einstellen. Zudem der Port und durch den Boolean Wert kann der Generator an- und ausgeschaltet werden.
- Event Generator: Es lässt sich das Interval zum generieren von neuen Events einstellen. Zudem der Port und durch den Boolean Wert kann der Generator an- und ausgeschaltet werden.
- Roadblock Generator: Es lässt sich das Interval zum generieren von neuen Straßenereignissen einstellen. Zudem der Port und durch den Boolean Wert kann der Generator an- und ausgeschaltet werden.
- Station Handler: Die Adresse und Port kann angepasst werden. Das Senden der Daten kann an- und ausgeschaltet werden.
- Person Checker: Hat keine Auswirkung mehr.

- Command Data Receiver: Anpassen von Adresse und Port zum Empfangen von Kommandos.
- Traffic Data Sender: Anpassen von Adresse und Port zum Senden der Verkehrslage.

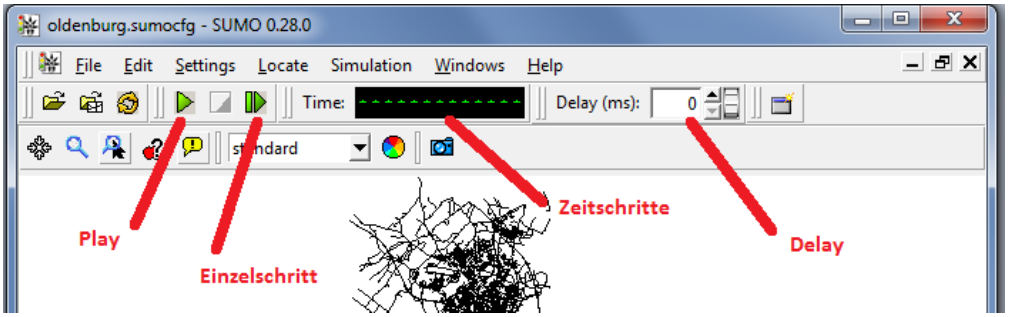


Abbildung 60: SUMO mit grafischer Oberfläche

Wenn die Simulation mit der Konsole gestartet wird gibt es keine Möglichkeit mehr einzugreifen. Bei Verwendung der GUI sind folgende Aktionen relevant.

- Play: Startet die Simulation.
- Einzelschritt: Führt einen einzelnen Zeitschritt aus.
- Zeitschritte: Zeigt die aktuelle Zeit an.
- Delay: Die Simulation wird durch den angegebenen Wert verlangsamt.

9 Evaluation

Unter einer Evaluation wird - im Kontext der Projektgruppe - das systematische Sammeln, Auswerten und Interpretieren von Daten verstanden, um eine valide Bewertung des entwickelten Softwaresystems zu erhalten.[48] Dabei müssen die Ergebnisse nachvollziehbar und aus empirisch gewonnen Daten gewonnen worden sein.[49]

Zur Bewertung von Software kann zwischen zwei Arten von Evaluationen unterschieden werden. **Formative Evaluationen** begleiten einen iterativen Entwicklungsprozess und beinhalten eine möglichst frühzeitige Benutzerbeteiligung. Dies soll zu Verbesserungen im untersuchten System schon während des eigentlichen Entwicklungsprozesses führen und erfolgt in der Regel anhand von Prototypen des Zielsystems. Neben quantitativen Daten dienen besonders qualitative Daten zur Verbesserung des Softwaresystems. Hierunter fallen vor allem Usability-Tests, die die Benutzbarkeit des Softwaresystems verbessern sollen. Die Entwickler haben so die Möglichkeit, iterativ Verbesserungen an ihrem Produkt vorzunehmen, bevor das Endprodukt fertiggestellt ist.[50]

Summative Evaluationen dienen der Bewertung des Softwareentwicklungsprozesses anhand von vorher formulierten Evaluationskriterien. Ergebnis ist eine finale Bewertung des untersuchten Softwaresystems, was u. a. den Vergleich von Alternativen ermöglicht. Die Ergebnisse können im Gegensatz zur formativen Evaluation nur in Nachfolgeprodukten bzw. späteren Versionen des Softwaresystems einfließen. Im Rahmen der Projektgruppe ist vor allem die Bewertung des entwickelten Softwaresystems hinsichtlich der definierten Evaluationskriterien interessant. Durch die mit quantitativen Methoden erzielten Ergebnisse können zuvor nur theoretisch erwartete Ergebnisse bzw. Verhaltensweisen überprüft und entsprechende Erkenntnisse gewonnen werden.[51]

[48] http://wirtschaftslexikon.gabler.de/Archiv/55834/evaluation-v6.html

[49] Vgl. http://www.gesis.org/fileadmin/upload/forschung/publikationen/gesis_reihen/iz_arbeitsberichte/ab_29.pdf

[50] Vgl. Ebd.

[51] Vgl. Ebd.

Das **Evaluationsziel** ist es, mögliche Schwachstellen des entwickelten Softwaresystems zu erkennen (z. B. lange Berechnungszeiten und Skalierbarkeit) und den Wertbeitrag der zentralen Komponenten zur dynamischen Zusammenlegung von Routen quantitativ zu belegen. Dafür eignet sich die **Analysierende Evaluation**, die das Softwaresystem in potentiellen Einsatzfällen untersucht. Abbildung61 sind verschiedene Verfahren zur Evaluation nach Grad der Benutzbeteiligung und Objektivität der Datenerhebung klassifiziert. Die Analysierende Evaluation kennzeichnet ihr quantitativer Charakter und der niedrige Grad der Benutzerbeteiligung. Innerhalb der Projektgruppe wurden potentielle Benutzereingaben (Anfragen an unseren Fahrdienst) mit Hilfe von möglichst realistischen Szenarien simuliert. Dabei sind unterschiedliche Verhaltensweisen der Benutzer (z. B. mit Car-Sharing und ohne) berücksichtigt worden, um das Systemverhalten unter den entsprechenden Szenarien beurteilen zu können. Hierfür wird ein Vergleich der Gesamtstrecke und der Dauer für alle Routen aufgestellt, um quantifizierte Aussagen über die Auswirkung von zusammengelegten Routen treffen zu können. Das Car-Sharing ist ein wesentlicher Kern der Projektgruppe, sodass belastbare Aussagen nicht nur über die Berechnungszeiten, sondern auch über unternehmerisch interessante Werte (z. B. kürzere Strecken durch Car-Sharing) gewonnen werden sollen. Aus unternehmerischer Sicht, sind diese Werte für die eigene Geschäftstätigkeit interessant, weil der Wertbeitrag für den Kunden unmittelbar ersichtlich wird. So kann das Car-Charing-Angebot für Kunden günstigere Preise, als über heutzutage noch normale Taxi-Dienstleistungen (ohne Mitnahme fremder Personen) ermöglichen und dabei zusätzlich einen höheren Komfort, als die öffentlichen Nahverkehrsmittel, bieten.

Außerdem sind vor der Durchführung der Evaluation zwei wesentliche Komponenten herausgestellt worden, die der Einschätzung der Projektmitglieder nach, am meisten Berechnungszeit benötigen. Dabei handelt es sich zum einen um die **Berechnung des Dynamic-Dial-a-Ride-Problems** (DDARP), die durch Meta-Heuristiken von *jSprit* erfolgt und zum anderen um die **Umwandlung der Routen** aus *GraphHopper* nach *SUMO*. Erwartungsgemäß sollte letzteres immer länger dauern und mit längeren Routen auch mehr Berechnungszeit benötigen. In den Routen, die durch den Auftragsgenerator erzeugt wurden, liegt der Erwartungswert der Berechnungszeit zwischen 25 Sek. und 30 Sek.

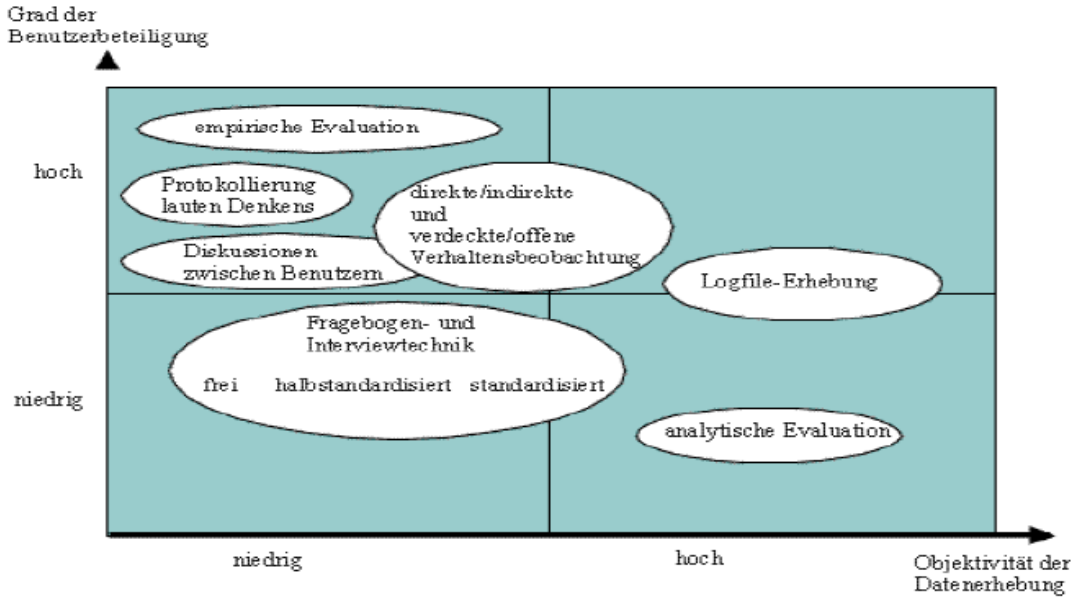


Abbildung 61: Verfahren zur Evaluation. Quelle: Görner, C., Ilg, R. (1993).Evaluation der Mensch-Rechner-Schnittstelle. [10]

9.1 Routenverarbeitung in SUMO

In der Evaluation werden Untersuchungen zur Performanz des Gesamtsystems durchgeführt. Im ersten Teil wurde untersucht wie viel Zeit zwischen dem Eintreffen einer Route bei SUMO und dem Starten des Taxis liegt. Dazu wurden die im Anhang befindlichen Auftragslisten verwendet. Außerdem wurde SUMO immer als Konsolenanwendung gestartet und eine Berechnung bis Zeitschritt 1000 abgewartet. Die Verkehrssimulation wurde mit der Standardkonfiguration aus dem GIT gestartet.

Erläuterung: Die Spalte „Erstmals erhalten“ zeigt die Uhrzeit wann der Auftrag erstmals bei Sumo eingegangen ist. Die Spalte Letzte Änderungen zeigt die Uhrzeit wann die letzte Korrektur der Route eingegangen ist und in Klammern die Zeit nach erstmaligen Erhalt. Die Spalte Start zeigt den Startzeitpunkt und in Klammern die Zeit nach erstmaligen Erhalt des Auftrages.
Im ersten Durchlauf wollen alle Passagiere alleine fahren. Es werden 50 Aufträge nahezu zeitgleich in das System überführt, sodass nicht alle abgearbeitet werden können. Aufgrund der hohen Auslastung beträgt die Rechenzeit für fast alle Taxis

Fahrzeugstatistiken mit Auftragsliste „evaluationorders“			
Fahrzeug	**Erstmals erhalten**	**Letzte Änderung**	**Start**
t0	17:18:25	17:18:57 (32 Sek.)	17:25:52 (7 Min. und 27 Sek.)
t1	nie	nie	nie
t2	17:18:21	17:19:03 (42 Sek.)	17:26:13 (7 Min. und 52 Sek.)
t3	17:18:26	17:18:52 (26 Sek.)	17:25:59 (7 Min. und 33 Sek.)
t4	17:18:18	17:19:13 (55 Sek.)	17:25:27 (7 Min. und 9 Sek)
t5	17:18:43	17:19:09 (26 Sek.)	17:26:00 (7 Min. und 17 Sek.)
t6	17:18:33	17:18:41 (8 Sek.)	17:25:38 (7 Min und 5 Sek.)
t7	17:18:27	17:19:00 (33 Sek.)	17:25:11 (6 Min und 44 Sek.)
t8	nie	nie	nie
t9	17:18:29	17:18:49 (20 Sek.)	17:24:49 (6 Min und 18 Sek.)

Abbildung 62: Evaluation SUMO 1

über sieben Minuten.

Im zweiten Durchlauf können Taxis auch geteilt werden. Es ist zu erkennen, dass der Unterschied jedoch nicht erheblich ist. Wieder beträgt die Rechenzeit für fast alle Aufträge über 7 Minuten.

Fahrzeugstatistiken mit Auftragsliste „evaluationorders_alone_false“			
Fahrzeug	**Erstmals erhalten**	**Letzte Änderung**	**Start**
t0	17:57:21	17:57:54 (33 Sek.)	18:04:56 (7 Min. und 33 Sek.)
t1	nie	nie	nie
t2	17:57:17	17:57:59 (42 Sek.)	18:04:50 (7 Min. und 33 Sek.)
t3	17:57:22	17:57:48 (26 Sek.)	18:04:58 (7 Min. und 36 Sek.)
t4	17:57:15	17:58:09 (54 Sek.)	18:04:36 (7 Min. und 21 Sek.)
t5	17:57:40	17:58:06 (26 Sek.)	18:04:51 (7 Min. und 11 Sek.)
t6	17:57:28	17:57:38 (10 Sek.)	18:04:39 (7 Min und 11 Sek.)
t7	17:57:24	17:57:56 (32 Sek.)	18:03:40 (6 Min. und 16 Sek.)
t8	nie	nie	nie
t9	17:57:25	17:57:46 (21 Sek.)	18:04:58 (7 Min und 33 Sek.)

Abbildung 63: Evaluation SUMO 2

Mit GUI		Konsolenanwendung	
Empfangen	**Start**	**Empfangen**	**Start**
14:07:40	14:08:00 (20 Sek.	14:01:05	14:01:21 (16 Sek.)

Abbildung 64: Vergleich SUMO mit und ohne GUI

Im nächsten Test wurde ein einzelner Auftrag in das System eingelesen. Dieser Test wird zweimal ausgeführt. Einmal mit grafischer Oberfläche und einmal als Konsolenanwendung.
Es ist zu erkennen, dass die Berechnung mit GUI ein bisschen länger dauert. Der Zeitvorteil der Konsolenanwendung summiert sich jedoch bei mehr Aufträgen auf.

9.2 Routenberechnung mit Graphhopper und JSprit

In einer zweiten Evaluation wurde gemessen, wieviel Zeit die Routenberechnung im CalcRoute-Operator nach dem Eintreffen eines Auftrags in Anspruch nimmt. Hierzu wurde die Latenz zwischen dem Eintreffen der Auftragsdaten und dem Senden der Route zur App mit Hilfe des Operators **CalcLatency** von Odysseus gemessen. Damit die Latenz gemessen werden kann, ist ein Setzen von **#METADATA Latency** in der Source für die eintreffenden Aufträge notwendig. In **CalcLatency** kann dann der Output von CalcRoute als Source angegeben werden. Die Messungen von **CalcLatency** werden anschließend in eine CSV-Datei geschrieben. Die folgende Hardware wurde für die Messungen verwendet:

- Betriebssystem: Microsoft Windows 10 pro
- CPU: Intel Pentium N3540: 2.16 GHz
- Speicher: 8 GB

Es wurden insgesamt drei Messungen durchgenommen, bei denen der OrderGenerator mit unterschiedlichen Zeitintervallen Aufträge generierte. Diese Zeitintervalle betrugen jeweils eine Sekunde, 10 Sekunden sowie eine Minute. Die Messergebnisse lassen sich im Anhang finden (A). Zu beobachten ist, dass bei allen Messungen eine steigende Latenz vorliegt. Dies lässt sich damit erklären, dass mit jedem neuen Auftrag insgesamt mehr Aufträge bei der Routenberechnung berücksichtigt werden müssen. Ebenfalls lässt sich beobachten, dass das Intervall, in dem Aufträge eintreffen, keinen erkennbaren Einfluss auf die Latenz hat.

9.3 Routenoptimierung mit Graphhopper und JSprit

Für die Evaluation der Routenoptimierung wurden 25 Aufträge generiert. Die Abhol- und Ablieferpunkte wurden zu 90% zufällig über das gesamte Stadtgebiet verteilt und zu 10% an signifikanten Orten in Oldenburg (z.B. Bahnhof, Waffenplatz, EWE-Arena) erzeugt. Die Distanz zwischen Abhol-und Ablieferpunkt ist ein zufälliger Wert zwischen 1000 und 4000 Meter Luftlinie und die Anzahl Passagiere ein Wert zwischen 1 und 4 Personen. Anschließend wurden die Aufträge in einem Abstand von jeweils 5 Sekunden an den CalcRoute-Operator zur Routenoptimierung gesendet. Im ersten Durchgang wurden alle Aufträge über ein Flag

als Einzelfahrt gekennzeichnet. In Abbildung 65 wird die Routenoptimierung aus JSprit für die Einzelfahrten dargestellt.

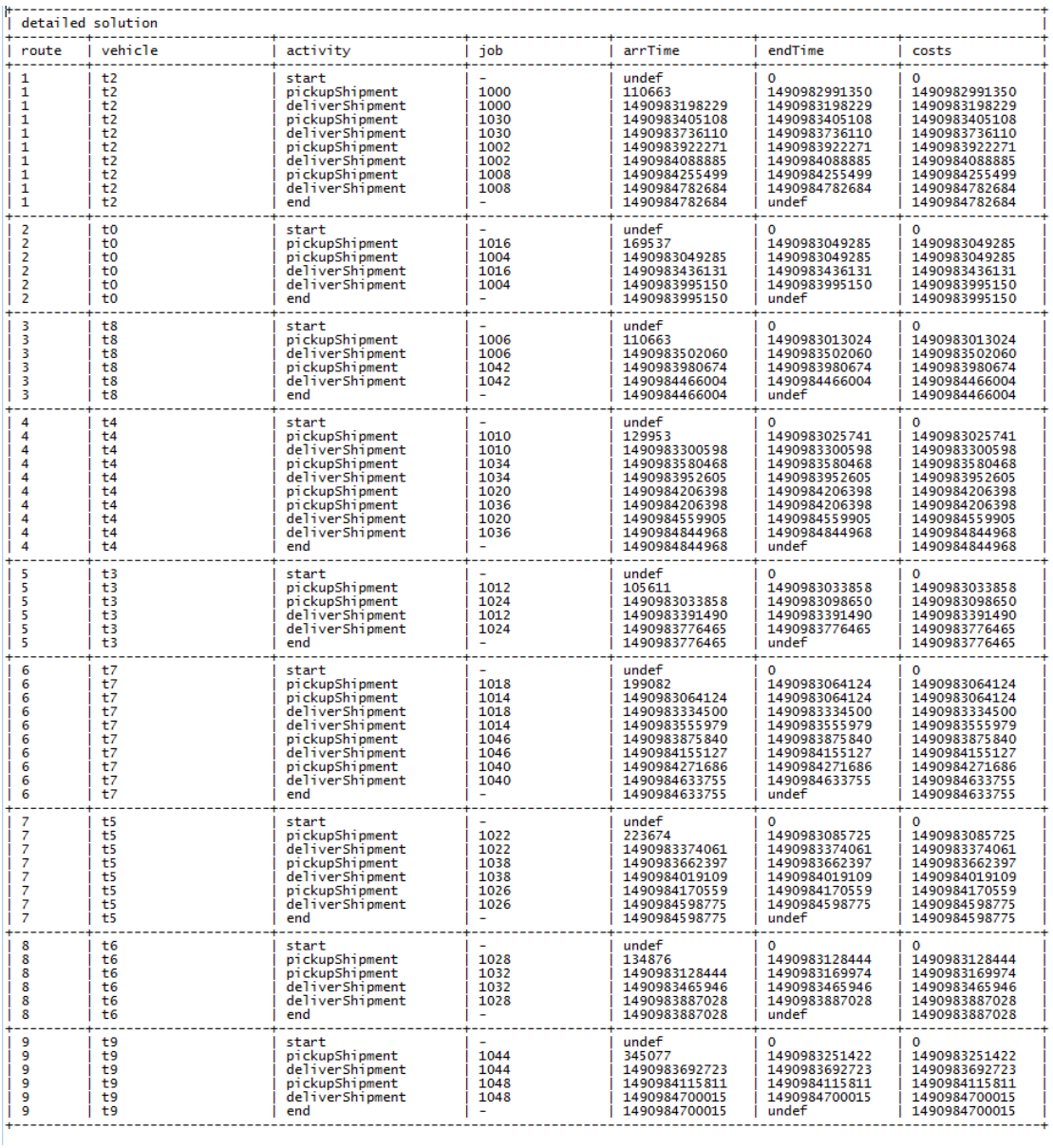
detailed solution

route	vehicle	activity	job	arrTime	endTime	costs
1	t2	start	-	undef	0	0
1	t2	pickupShipment	1000	110663	1490982991350	1490982991350
1	t2	deliverShipment	1000	1490983198229	1490983198229	1490983198229
1	t2	pickupShipment	1002	1490983405108	1490983405108	1490983405108
1	t2	deliverShipment	1002	1490983571722	1490983571722	1490983571722
1	t2	pickupShipment	1022	1490983738336	1490983738336	1490983738336
1	t2	deliverShipment	1022	1490984026672	1490984026672	1490984026672
1	t2	pickupShipment	1008	1490984285148	1490984285148	1490984285148
1	t2	deliverShipment	1008	1490984812333	1490984812333	1490984812333
1	t2	end	-	1490984812333	undef	1490984812333
2	t0	start	-	undef	0	0
2	t0	pickupShipment	1004	131228	1490983007747	1490983007747
2	t0	deliverShipment	1004	1490983554230	1490983554230	1490983554230
2	t0	pickupShipment	1048	1490984106494	1490984106494	1490984106494
2	t0	deliverShipment	1048	1490984690698	1490984690698	1490984690698
2	t0	end	-	1490984690698	undef	1490984690698
3	t8	start	-	undef	0	0
3	t8	pickupShipment	1006	110663	1490983013024	1490983013024
3	t8	deliverShipment	1006	1490983502060	1490983502060	1490983502060
3	t8	pickupShipment	1046	1490983980674	1490983980674	1490983980674
3	t8	deliverShipment	1046	1490984259961	1490984259961	1490984259961
3	t8	pickupShipment	1044	1490984388443	1490984388443	1490984388443
3	t8	deliverShipment	1044	1490984829744	1490984829744	1490984829744
3	t8	end	-	1490984829744	undef	1490984829744
4	t4	start	-	undef	0	0
4	t4	pickupShipment	1010	129953	1490983025741	1490983025741
4	t4	deliverShipment	1010	1490983300598	1490983300598	1490983300598
4	t4	pickupShipment	1030	1490983580468	1490983580468	1490983580468
4	t4	deliverShipment	1030	1490983911470	1490983911470	1490983911470
4	t4	pickupShipment	1024	1490984119071	1490984119071	1490984119071
4	t4	deliverShipment	1024	1490984656215	1490984656215	1490984656215
4	t4	end	-	1490984656215	undef	1490984656215
5	t3	start	-	undef	0	0
5	t3	pickupShipment	1012	105611	1490983033858	1490983033858
5	t3	deliverShipment	1012	1490983324976	1490983324976	1490983324976
5	t3	pickupShipment	1038	1490983617816	1490983617816	1490983617816
5	t3	deliverShipment	1038	1490983974528	1490983974528	1490983974528
5	t3	pickupShipment	1026	1490984125978	1490984125978	1490984125978
5	t3	deliverShipment	1026	1490984554194	1490984554194	1490984554194
5	t3	end	-	1490984554194	undef	1490984554194
6	t7	start	-	undef	0	0
6	t7	pickupShipment	1014	206244	1490983041534	1490983041534
6	t7	deliverShipment	1014	1490983377336	1490983377336	1490983377336
6	t7	pickupShipment	1040	1490983697197	1490983697197	1490983697197
6	t7	deliverShipment	1040	1490984059266	1490984059266	1490984059266
6	t7	pickupShipment	1028	1490984402845	1490984402845	1490984402845
6	t7	deliverShipment	1028	1490984920411	1490984920411	1490984920411
6	t7	end	-	1490984920411	undef	1490984920411
7	t6	start	-	undef	0	0
7	t6	pickupShipment	1016	247689	1490983049285	1490983049285
7	t6	deliverShipment	1016	1490983474440	1490983474440	1490983474440
7	t6	pickupShipment	1042	1490983899595	1490983899595	1490983899595
7	t6	deliverShipment	1042	1490984384925	1490984384925	1490984384925
7	t6	end	-	1490984384925	undef	1490984384925
8	t5	start	-	undef	0	0
8	t5	pickupShipment	1032	253232	1490983169974	1490983169974
8	t5	deliverShipment	1032	1490983465946	1490983465946	1490983465946
8	t5	pickupShipment	1018	1490983629899	1490983629899	1490983629899
8	t5	deliverShipment	1018	1490983903197	1490983903197	1490983903197
8	t5	pickupShipment	1020	1490984178036	1490984178036	1490984178036
8	t5	deliverShipment	1020	1490984531543	1490984531543	1490984531543
8	t5	end	-	1490984531543	undef	1490984531543
9	t9	start	-	undef	0	0
9	t9	pickupShipment	1034	345033	1490983180525	1490983180525
9	t9	deliverShipment	1034	1490983552662	1490983552662	1490983552662
9	t9	pickupShipment	1036	1490983923200	1490983923200	1490983923200
9	t9	deliverShipment	1036	1490984399220	1490984399220	1490984399220
9	t9	end	-	1490984399220	undef	1490984399220

Abbildung 65: JSprit Lösung: Einzelfahrt

Nach der Routenoptimierung der Einzelfahrten wurden die gleichen Aufträge in einem weiteren Durchgang über ein Flag als geteilte Fahrt gekennzeichnet und erneut an den CalcRoute-Operator gesendet. Die Routenoptimierung aus JSprit

lässt sich in Abbildung 66 ablesen.



detailed solution

route	vehicle	activity	job	arrTime	endTime	costs
1	t2	start	-	undef	0	0
1	t2	pickupShipment	1000	110663	1490982991350	1490982991350
1	t2	deliverShipment	1000	1490983198229	1490983198229	1490983198229
1	t2	pickupShipment	1030	1490983405108	1490983405108	1490983405108
1	t2	deliverShipment	1030	1490983736110	1490983736110	1490983736110
1	t2	pickupShipment	1002	1490983922271	1490983922271	1490983922271
1	t2	deliverShipment	1002	1490984088885	1490984088885	1490984088885
1	t2	pickupShipment	1008	1490984255499	1490984255499	1490984255499
1	t2	deliverShipment	1008	1490984782684	1490984782684	1490984782684
1	t2	end	-	1490984782684	undef	1490984782684
2	t0	start	-	undef	0	0
2	t0	pickupShipment	1016	169537	1490983049285	1490983049285
2	t0	pickupShipment	1004	1490983049285	1490983049285	1490983049285
2	t0	deliverShipment	1016	1490983436131	1490983436131	1490983436131
2	t0	deliverShipment	1004	1490983995150	1490983995150	1490983995150
2	t0	end	-	1490983995150	undef	1490983995150
3	t8	start	-	undef	0	0
3	t8	pickupShipment	1006	110663	1490983013024	1490983013024
3	t8	deliverShipment	1006	1490983502060	1490983502060	1490983502060
3	t8	pickupShipment	1042	1490983980674	1490983980674	1490983980674
3	t8	deliverShipment	1042	1490984466004	1490984466004	1490984466004
3	t8	end	-	1490984466004	undef	1490984466004
4	t4	start	-	undef	0	0
4	t4	pickupShipment	1010	129953	1490983025741	1490983025741
4	t4	deliverShipment	1010	1490983300598	1490983300598	1490983300598
4	t4	pickupShipment	1034	1490983580468	1490983580468	1490983580468
4	t4	deliverShipment	1034	1490983952605	1490983952605	1490983952605
4	t4	pickupShipment	1020	1490984206398	1490984206398	1490984206398
4	t4	pickupShipment	1036	1490984206398	1490984206398	1490984206398
4	t4	deliverShipment	1020	1490984559905	1490984559905	1490984559905
4	t4	deliverShipment	1036	1490984844968	1490984844968	1490984844968
4	t4	end	-	1490984844968	undef	1490984844968
5	t3	start	-	undef	0	0
5	t3	pickupShipment	1012	105611	1490983033858	1490983033858
5	t3	pickupShipment	1024	1490983033858	1490983098650	1490983098650
5	t3	deliverShipment	1012	1490983391490	1490983391490	1490983391490
5	t3	deliverShipment	1024	1490983776465	1490983776465	1490983776465
5	t3	end	-	1490983776465	undef	1490983776465
6	t7	start	-	undef	0	0
6	t7	pickupShipment	1018	199082	1490983064124	1490983064124
6	t7	pickupShipment	1014	1490983064124	1490983064124	1490983064124
6	t7	deliverShipment	1018	1490983334500	1490983334500	1490983334500
6	t7	deliverShipment	1014	1490983555979	1490983555979	1490983555979
6	t7	pickupShipment	1046	1490983875840	1490983875840	1490983875840
6	t7	deliverShipment	1046	1490984155127	1490984155127	1490984155127
6	t7	pickupShipment	1040	1490984271686	1490984271686	1490984271686
6	t7	deliverShipment	1040	1490984633755	1490984633755	1490984633755
6	t7	end	-	1490984633755	undef	1490984633755
7	t5	start	-	undef	0	0
7	t5	pickupShipment	1022	223674	1490983085725	1490983085725
7	t5	deliverShipment	1022	1490983374061	1490983374061	1490983374061
7	t5	pickupShipment	1038	1490983662397	1490983662397	1490983662397
7	t5	deliverShipment	1038	1490984019109	1490984019109	1490984019109
7	t5	pickupShipment	1026	1490984170559	1490984170559	1490984170559
7	t5	deliverShipment	1026	1490984598775	1490984598775	1490984598775
7	t5	end	-	1490984598775	undef	1490984598775
8	t6	start	-	undef	0	0
8	t6	pickupShipment	1028	134876	1490983128444	1490983128444
8	t6	pickupShipment	1032	1490983128444	1490983169974	1490983169974
8	t6	deliverShipment	1032	1490983465946	1490983465946	1490983465946
8	t6	deliverShipment	1028	1490983887028	1490983887028	1490983887028
8	t6	end	-	1490983887028	undef	1490983887028
9	t9	start	-	undef	0	0
9	t9	pickupShipment	1044	345077	1490983251422	1490983251422
9	t9	deliverShipment	1044	1490983692723	1490983692723	1490983692723
9	t9	pickupShipment	1048	1490984115811	1490984115811	1490984115811
9	t9	deliverShipment	1048	1490984700015	1490984700015	1490984700015
9	t9	end	-	1490984700015	undef	1490984700015

Abbildung 66: JSprit Lösung: Geteilte Fahrt

Diese beiden Lösungen aus der Routenoptimierung wurden in der weiteren Evaluation genauer betrachtet. In der Abbildung 67 kann man erkennen, dass die Taxis für alle Aufträge als Einzelfahrt eine Strecke von 198,23 km bewältigen müssen und dafür eine Gesamtfahrzeit 4:50:14 benötigen. Wenn hingegen die Aufträge

sich auch die Taxis teilen können, dann beträgt die Gesamtstrecke 177,60 km und die Gesamtfahrzeit 4:26:59.

	Fahrt (Einzel)			Fahrt (Geteilt)		
Taxi	Aufträge	Distanz (m)	Dauer (ms)	Aufträge	Distanz (m)	Dauer (ms)
t0	2	21954,03	1851368	2	15665,40	1259466
t1	0	0,00	0	0	0,00	0
t2	4	29160,67	2452082	4	26944,16	2608730
t3	3	19761,86	1729748	2	11652,24	1029324
t4	3	19936,06	1901965	4	25316,25	2521543
t5	3	20461,14	1935006	3	21378,75	1893450
t6	2	22861,50	1829556	2	8288,58	1007527
t7	3	20414,69	2068788	4	22923,04	2010621
t8	3	22632,53	1991956	2	20699,74	1611732
t9	2	21042,96	1654021	2	24731,23	2077125
Σ	**25**	**198225,44**	**17.414.490**	**25**	**177599,39**	**16.019.518**

Abbildung 67: Auswertung Taxis

Wenn die beiden Lösungen aus Auftragssicht verglichen werden, lässt sich in Abbildung 68 erkennen, dass bei einer Einzelfahrt die durchschnittliche Wartezeit 0:14:07, die durchschnittliche Fahrzeit 0:06:08 und die durchschnittliche Fahrdistanz 4,14 km ist. Bei der geteilten Fahrt reduziert sich die durchschnittliche Wartezeit auf 0:12:19, jedoch erhöht sich die durchschnittliche Fahrzeit auf 0:08:03 und die durchschnittliche Fahrdistanz auf 5,18 km.

	Fahrt (Einzel)			Fahrt (Geteilt)		
Auftrag	Wartezeit (ms)	Fahrzeit (ms)	Distanz (m)	Wartezeit (ms)	Fahrzeit (ms)	Distanz (m)
1	83235	206879	2169,19	83235	206879	2169,20
2	837548	166614	1366,35	1329172	166614	1366,36
3	131228	552265	8749,99	313613	945866	12442,26
4	83235	478614	5201,32	83235	478614	5201,32
5	1924897	309017	3650,63	2081545	309017	3650,62
6	129953	279870	2559,79	129953	279870	2559,79
7	105611	292840	3553,75	105611	511715	4720,89
8	202293	319861	3773,48	234193	475914	4793,89
9	257513	425155	4566,12	169537	530922	5115,25
10	703983	274839	2946,41	195131	293497	3040,52
11	1581499	327710	3429,19	1283892	926791	9901,66
12	1378085	288336	3597,15	223840	288336	3597,14
13	1346138	555827	6667,86	333582	695742	8911,21
14	1301544	428216	4211,05	1465246	428216	4211,04
15	1547663	521125	3448,74	134876	711137	6084,28
16	810793	327744	3288,75	815267	327744	3288,75
17	245070	294960	3212,63	291485	975610	6958,90
18	368899	370538	4258,82	659561	370538	4258,82
19	1179600	474422	5434,32	1698202	851862	9775,70
20	790094	360000	4095,75	953796	360000	4095,74
21	857956	346128	3944,58	1664493	346128	3944,62
22	1343954	485614	6927	1126118	485614	6927,00
23	1568868	423088	4592,42	354067	423088	4592,42
24	1143297	297089	3198,15	1250845	297089	3198,16
25	1267165	396992	4759,19	1492921	396992	4759,19
Ø	**847604,84**	**368149,72**	**4144,1052**	**738936,64**	**483351,8**	**5182,5892**

Abbildung 68: Auswertung Aufträge

Schlussfolgernd kann man sagen, dass sich eine Routenoptimierung mit geteilten Fahrten für ein Taxiunternehmen lohnt, da sich in unserem Szenario die Fahrdistanz für die Aufträge um 20,62 km und die Gesamtfahrzeit um 0:23:15 (hh:mm:ss) verringert hat. Die Kunden haben auf der einen Seite eine kürzere Wartezeit, aber dafür eine längere Fahrzeit. Hier müssten Anreize geschaffen werden, die den Kunden zu einer geteilten Fahrt motivieren. So könnte z.B. ein Teil des monetären Gewinns durch die geteilte Fahrt vom Unternehmen auf den Kunden umgelagert werden. Als Wertbeitrag für den Kunden ergibt sich so ein höherer Komfort, als

über den Transport mit dem öffentlichen Personennahverkehr (ÖPNV), zu einem günstigeren Preis als bei normalen Taxi-Fahrdiensten (ohne Car-Sharing).

9.4 Zusammenfassung

In diesem Kapitel wurde die Evaluation des von der Projektgruppe entwickelten Softwaresystems beschrieben. Dabei sind die Kernkomponenten des Softwaresystems zuerst identifiziert, mit Erwartungswerten verbunden und anschließend quantitativ innerhalb einer Summativen Evaluation bewertet worden. Das Evaluationsziel wurde primär durch das Finden von Schwachstellen des Systems und die Ermittlung des Wertbeitrages der zentralen Komponente zur dynamischen Zusammenlegung von Routen beschrieben.
Die Entscheidung zur Evaluation der Routenberechnung und -umrechnung beruht darauf, dass im Verlauf der Projektgruppe vor allem die Routenumrechnung für *SUMO* als erwartete Schwachstelle erkannt worden ist. Klassifiziert wurde die gewählte Methode als Analysierende Evaluation. Diese ermöglicht es, das Softwaresystem in potentiellen Einsatzfällen zu untersuchen und erlaubt durch ihren quantitativen Charakter eine valide Bewertung des Softwaresystems unter den gegebenen Rahmenbedingungen.
Für die Evaluation der Routenberechnung wurde der CalcLatency-Operator von Odysseus verwendet und die Latenz zwischen dem Eintreffen der Auftragsdaten durch den Ordergenerator und dem Senden der Route zur App gemessen. Beobachtet wurde, dass sich die Latenz mit zunehmender Anzahl an Aufträgen erwartungsgemäß vergrößert hat. Dies lässt sich damit erklären, dass mehr Aufträge beim Aufbau und der Lösung des dynamischen Dial-A-Ride-Problems (DDARP) längere Berechnungszeiten, für das Finden einer optimalen Lösung, mit sich bringen.
Neben Daten zur Berechnungsdauer der Routengenerierung und -umwandlung ist zudem der Wertbeitrag der zentralen Komponente für die potentiellen Anwender untersucht worden. Ein wesentlicher Kern der Projektgruppe lag in der Bereitstellung einer Car-Sharing-Option, die es ermöglicht, Kundenrouten zusammen zu legen und so Aufträge gemeinsam zu bedienen. Erwartungsgemäß ergab die Evaluation dieser Komponente, dass Aufträge mit aktivierter Car-Sharing-Option

deutlich kürzer sind, als per Einzelfahrt. Auch die Gesamtfahrzeit hat sich mit aktivierter Car-Sharing-Option deutlich verringert. So ließ sich der Wertbeitrag durch die Car-Sharing-Option auch quantitativ belegen und ermöglichte eine valide Aussage zu dieser Komponente.
Insgesamt konnten die zuvor nur subjektiv erwarteten Verhaltensweisen, des innerhalb der Projektgruppe entwickelten Softwaresystems, mit Hilfe der Evaluation quantitativ belegt werden. Damit ließen sich valide Aussagen zu den Berechnungszeiten der Routengenerierung, -umrechnung und zum Wertbeitrag der zentralen Komponente zur dynamischen Zusammenlegung von Routen treffen.

10 Zusammenfassung und Ausblick

10.1 Fazit

Die Projektgruppe „Pimp My Taxi Fleet“ war ein sehr umfangreiches Projekt, welches den Projektteammitgliedern in den unterschiedlichsten Bereichen ihr Können abverlangte. Bei keinem Teammitglied bestand Vorwissen zu Themenbereichen wie Odysseus, Verkehrssimulation oder SpringMVC. Dennoch wurde ein lauffähiger Prototyp entwickelt und die wesentlichen Ziele der Projektgruppe erreicht.
In dem Prototyp wurde die Routengenerierung und -optimierung als DDARP umgesetzt, wodurch das "Carsharing"beim Taxifahren ermöglicht wurde. Auch die Standortplanung wurde implementiert, sodass die Taxis anhand von Events und historischen Aufträgen sinnvoll platziert werden können. Ein weiterer wichtiger Aspekt war die Einbeziehung von elektrischen und selbstfahrenden Fahrzeugen, wodurch ein Zukunftsszenario erzeugt wurde. Es werden die aktuell in Oldenburg befindlichen Ladesäulen und die Akkuladung der Fahrzeuge simuliert, die jedoch einer vereinfachten Berechnung unterliegen.
Auch externe Einflüsse sollten bei der Routenplanung eine Rolle spielen. Dazu zählen Verkehrsereignisse wie Staus oder Baustellen sowie Events aus sozialen Medien, wie beispielsweise Facebook. Als Quelle für die Verkehrsereignisse dient derzeit die Verkehrssimulation SUMO, die jedoch leicht ausgetauschbar gestaltet wurde. Für die Einbeziehung von Events können reale Veranstaltungen in Oldenburg mit einem implementierten Tool abgerufen und verarbeitet werden. Weiterhin wurde jedoch auch ein Generator realisiert, der beispielhafte Veranstaltungen erstellt.
Insbesondere Odysseus als Datenstrommanagementsystem hat am Anfang viel Zeit zur Einarbeitung benötigt. Es musste die Funktionsweise von Operatoren und die Abfragesprache PQL erlernt werden. Auch die Verkehrssimulation hat anfangs viel Zeit in Anspruch genommen, weil zunächst eine Java Bibliothek als Schnittstelle verwendet wurde, die sich im Nachhinein jedoch als Sackgasse erwiesen hat. Daraufhin musste eine Python Bibliothek verwendet und ein eigenes Skript gechrieben werden.
Insgesamt hat sich die Verkehrssimulation als relativ unstabil herausgestellt, wodurch viele Probleme aufgetreten sind. Es mussten einige Änderungen an der Pla-

nung vorgenommen werden, wie zum Beispiel eine realistische Simulation des Akkuladestands. Bei einem Nachfolgerprojekt sollte noch einmal evaluiert werden, ob weitere Verkehrssimulationen existieren, die stabiler sind. Insgesamt hat die Projektgruppe mit vielen Fehlern in allen Systemkomponente zu kämpfen gehabt. Dadurch hat es sehr lange gedauert bis ein lauffähiges und stabiles Produkt entstanden ist.
Zum Schluss wurde das Ziel erreicht externe Anwendungen wie die App und das Dashboard zu entwickeln. Die App wurde so weit entwickelt, dass ein Kunde bereits ein Taxi bestellen kann. Zudem wurden weitere Funktionen wie das Tracking der Taxis implementiert. Auch das Dashboard bietet zahlreiche Funktionen zur Verwaltung der Taxis und zur Auswertung von statistischen Daten über Aufträge und Events.

10.2 Ausblick

Insgesamt ist das Projekt gelungen, indem ein lauffähiger Prototyp implementiert wurde. Dennoch gibt es weiteren Raum für Verbesserungen.
Für das Projekt wurde eine Verkehrssimulation eingebunden, die Taxi- und Verkehrsdaten liefert. In der Realität werden diese Daten von realen Taxis und Verkehrsinformationsdiensten geliefert. Diese können noch an Odysseus angebunden werden, sodass nicht mehr auf simulierte Daten zurückgegriffen werden muss. Ein weiterer Aspekt stellt die Betrachtung Kostenvorteilen bei der Routenplanung dar. Derzeit werden Kostenvorteile nicht an die Kunden weitergegeben, sodass trotz „Taxi Sharing“ der gleiche Preis wie bei einer Einzelfahrt bezahlt werden muss.
Weiterhin gab es im Laufe des Projekts immer wieder Performance-Probleme. Dadurch wurden zum Beispiel Abstriche beim Senden von Verkehrsdaten in Kauf genommen, indem nur Daten für ausgewählte Straßenabschnitte gesendet werden. Auch das Berechnen und Abfahren der Routen hat sich als sehr performance-lastig herausgestellt. Vor allem die Umwandlung von GPS-Punkten zu Straßenabschnitten dauert im Schnitt circa 30 Sekunden pro Auftrag. Aufgrund dessen können dafür noch einige Verbesserungen zur Optimierung der Performance durchgeführt werden.
Neben der Performance können Sicherheitsaspekte, auch unter Beachtung rechtlicher Grundlagen überdacht werden. Da die App mit personenbezogenen Daten,

insbesondere ihrer Standort- und Zahlungsdaten agiert, muss dessen Verarbeitung rechtskonform stattfinden. Hierzu sollten die Serverkonfigurationen angepasst und die Übertragungen verschlüsselt werden. Ebenso kann ein Rechte-Management unberechtigten Zugriff auf das Dashboard einschränken.
Ein weiterer Aspekt stellt die Verbesserung des Ladekonzepts dar. Zum einen kann die Belegung von Ladesäulen mit einbezogen werden, sodass eine Ladesäulen nur dann angefahren wird, wenn sie frei ist. Zum anderen kann die Ladeplanung dynamisiert werden, was bedeutet, dass die Taxis nicht nur dann geladen werden wenn der Akku leer ist, sondern auch in speziellen Situationen. Ein einfaches Beispiel für so eine Situation besteht, wenn das Taxi keine Aufträge oder Events abfährt und somit an einer Ladestation auf die nächste Zuweisung wartet.
Auch die dynamische Standortplanung kann noch erweitert werden, indem die Schwellwerte und Grids dynamisiert werden. So kann ein Algorithmus entwickelt werden, der die Grenzwerte der einzelnen Quarter intelligent an das Auftragsaufkommen in Oldenburg anpasst. Weiterhin können statt der aktuell gleich großen Quarters unterschiedliche größen verwendet werden, um die Position der Taxis noch genau planen zu können. So ist es sinnvoll in Gebieten mit vielen Menschen und vielen Aufträgen, z. B. in der Oldenburgen Innenstadt, die Quarter kleiner zu gestalten und in Gebieten mit weniger Menschen und weniger Aufträgen die Quarter so zu belassen bzw. sogar zu vergrößern.
Als letzten Aspekt könnte eine Funktion zur Integration von privaten Fahrzeugen implementiert werden. Dabei kann eine Person für eine Provision sein Fahrzeug für den Taxi Pool bereitstellen. Dazu muss das Fahrzeug jedoch selbstfahrend und elektrisch sein. Zudem muss es Fahrzeugdaten wie GPS Daten senden und die von der Routenplanung bereitgestellte Route abfahren können. Dabei könnte diese Funktion ebenfalls durch die App realisiert werden, sodass ein weiterer Kundenzugang geschaffen wird, über den der Kunde sein Fahrzeug zur Verfügung stellen kann.

Literatur

[1] Mathias Bertram and Stefan Bongard. *Elektromobilitat im motorisierten Individualverkehr - Grundlagen, Einflussfaktoren und Wirtschaftlichkeitsvergleichlichkeitsvergleich*. Springer-Verlag, Wiesbaden, 2014.

[2] Fabian Blechschmidt. Taxi 2.0 - eine verkehrssimulation für das jahr 2035. Bachelor thesis. Berlin.

[3] Jean-Francois Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54:573–58611, 2003.

[4] Jörg Dallmeyer. *Simulation des Straßenverkehrs in der Großstadt*. Springer Verlag, Wiesbaden, 2014.

[5] D. Carney et. al. Monitoring streams - a new class of data management applications. In *Proc. 28th Intl. Conference on Very Large Data Bases*, Hong Kong, China, 2002.

[6] Justus Haucap et al. Chancen der digitalisierung auf märkten für urbane mobilität: Das beispiel uber, 2015.

[7] Moreira-Matias et al. Predicting taxi-passenger demand using streaming data. *IEEE Transactions onIntelligent Transportation Systems*, 14(3):1393–1402, 2013.

[8] Tali Tzoreff et. al. The vehicle routing problem with pickups and deliveries on some special graphs. *Discrete Applied Mathematics*, 116(3):193–229, 2002.

[9] S. Geisler. Data stream management systems. *Data Exchange, Integration, and Streams*, pages 275–304, 2013.

[10] C. Görner. *Evaluation der Mensch-Rechner-Schnittstelle*. Oldenbourg Verlag, München, Wien, 1993.

[11] Achim Kampker, Dirk Vallee, and Armin Schnettler. *Elektromobilität - Grundlagen einer Zukunftstechnologie*. Springer-Verlag, Heidelberg, 2013.

[12] Anton Karle. *Elektromobilitat - Grundlagen und Praxis*. Carl Hanser Verlag, München, 2015.

[13] J. Krämer. *Continuous Querys over Data Streams – Semantics and Implementation*. Philipps-Universität Marburg, 2007.

[14] Wolf-Ulrich Raffel. *Agentenbasierte Simulation als Verfeinerung der Diskreten-Ereignis-Simulation unter besonderer Berücksichtigung des Beispiels Fahrerloser Transportsysteme*. PhD thesis, 2005.

[15] Martin Treiber and Arne Kesting. *Verkehrsdynamik und -simulation*. Springer-Verlag, Heidelberg, 2010.

[16] Nils Woitzik. Das dial-a-ride-problem, 2007.

[17] Jeffrey M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley & Sons, Hoboken, 2009.

Glossar

Browser Programm zur Darstellung von Webseiten im World Wide Web.

DARP (engl. dail-a-ride problem) ist ein mathematisches Problem, bei dem eine Reihe von Lieferungen über eine Anzahl verfügbarer Routen zwischen Start und Zielpunkt innerhalb einer begrenzten Zeitspanne erfolgen muss.

DDARP (engl. dynamic dial-a-ride problem) erweitert das DARP um dynamisch eingehende Lieferungen.

Mobile Application oder App (Kurzform, engl. application) wird eine Anwendungssoftware für mobile Betriebssysteme bezeichnet. Zu unterscheiden sind native Apps, die nur für eine Plattform entwickelt wurden und auf dieser laufen, von plattformunabhängigen Web-, Hybrid- oder Cross-Plattform-Apps.

Odysseus (Oldenburger DynaQuest Datastream Query System) ist ein flexibles Framework zur Verarbeitung von Datenströmen.

Scrum bezeichnet ein modernes Vorgehensmodell des Projekt- und Produktmanagements, insbesondere zur agilen Softwareentwicklung.

A Anhang

Testfallmappe

Die Projektgruppe hat die erstellen *JUnit*-Testfälle ein einer Testfallmappe dokumentiert. Für die Dokumentation wurde eine Vorlage erstellt, um die Testfälle einheitlich zu beschreiben. Der erarbeitete Prozess zur Auswahl, Implementierung und Dokumentation von Testfällen ist im Abschnitt Testkonzept dokumentiert.

UpdateGraph

Klasse UpdateGraphTest

Package de.uniol.inf.is.odysseus.pgtaxi.test.traffic.UpdateGraphTest

Signatur public void setUp(), public void testGetEdge(), public void testSetSpeed(), public void testSetDistance()

Beschreibung Testfall für die Aktualisierung von Attributen des Graphen für das Straßennetz von Oldenburg. Es wird überprüft, ob die Geschwindigkeit bzw. die Distanz einer Kante erfolgreich aktualisiert wurde.

Vorbedingung Der Graph für das Straßennetz von Oldenburg muss aufgebaut sein.

public void testGetEdge() :

Nach der Übergabe von Breiten- und Längengrad soll die nächstliegende Kante des Graphen vom Oldenburger Straßennetz zurückgegeben werden.

Vorbedingung Längen- und Breitengrad müssen innerhalb von Oldenburg liegen.

Nachbedingung Es muss die nächstliegende Kante zurückgegeben worden sein.

public void testSetSpeed() :

Das Attribut mit der Geschwindigkeit der aktualisierten Kante muss dem neuen Wert entsprechen.

Vorbedingung Die neue Geschwindigkeit sollte nicht höher, als die maximal erlaubte Geschwindigkeit auf dieser Kante sein. Sonst wird ist die neue Geschwindigkeit automatisch die maximal erlaubte Geschwindigkeit auf dieser Kante.

Nachbedingung Die Geschwindigkeit auf der gewählten Kante muss aktualisiert worden sein.

public void testSetDistance() :

Das Attribut mit der Distanz der aktualisierten Kante muss dem neuen Wert der Distanz entsprechen.

Vorbedingung Die Distanz darf nicht negativ gewählt werden.

Nachbedingung Die Distanz auf der gewählten Kante muss aktualisiert worden sein.

Testhistory

Erfolgreich: Michael Kroeger, 08.08.2016.

RequestBuilder

Klasse RequestBuilderTest

Package App/[..]/src/test/java/com/taxi/pg/pimpmytaxifleet/rest

Signatur public void testBuildLoginUserRequest(), public void testBuildRegisterUserRequest(), public void testBuildGetUserByIDRequest(), public void testBuildDeleteUserByIDRequest(), public void testBuildUpdateUserRequest()

Beschreibung Testfall zur Überprüfung der Erzeugung der korrekten HTTP-URLs und Requestbodys durch die die User der App angelegt, gelöscht, aktualisiert usw. werden. In dieser Klassen werden die Requests lediglich gebaut und nicht gefeuert.

Vorbedingung Keine.

public void testBuildLoginUserRequest() :

Nach Übergabe der E-Mail-Adresse des Nutzers und dem zugehörigen Passwort, soll ein GET-Request mit korrektem Link gebaut werden.

Vorbedingung E-Mail-Adresse und Passwort des Nutzers müssen als Strings übergeben werden.

Nachbedingung Es wird ein Request zurückgeben. Dabei handelt es sich um ein GET-Request mit HTTP-URL. In dieser URL sind die Nutzerdaten eingefügt.

public void testBuildRegisterUserRequest() :

Nach Übergabe von Name, Vorname, Mobiletelefon-Nummer, E-Mail-Adresse, Passwort, Kreditkartennummer, Paypal-Account, IBAN und BIC des Nutzers soll ein POST-Request mit korrektem Link gebaut werden. Dabei sind lediglich Name, Vorname, Mobiletelefon-Nummer und E-Mail-Adresse als Werte übergeben, alle Zahlungenmethoden sind *null*.

Vorbedingung Name, Vorname, Mobiletelefon-Nummer, E-Mail-Adresse müssen als String übergeben werden.

Nachbedingung Es wird ein Request zurückgeben. Dabei handelt es sich um ein POST-Request mit HTTP-URL und einem Requestbody. In diesem Requestbody werden die Nutzerdaten übergeben.

public void testBuildGetUserByIDRequest() :

Nach Übergabe der ID des Nutzers soll ein GET-Request mit korrektem Link gebaut werden. In diesem Testfall wird beispielhaft die ID 1 benutzt.

Vorbedingung Die ID des Nutzers muss als Integer übergeben werden.

Nachbedingung Es wird ein Request zurückgeben. Dabei handelt es sich um ein GET-Request mit HTTP-URL. In dieser URL ist die ID des Nutzers

eingefügt.

public void testBuildDeleteUserByIDRequest() :
Nach Übergabe der ID des Nutzers soll ein DELETE-Request mit korrektem Link gebaut werden. In diesem Testfall wird beispielhaft die ID 1 benutzt.
Vorbedingung Die ID des Nutzers muss als Integer übergeben werden.
Nachbedingung Es wird ein Request zurückgeben. Dabei handelt es sich um ein DELETE-Request mit HTTP-URL. In dieser URL ist die ID des Nutzers eingefügt.

public void testBuildUpdateUserRequest() :
Nach Übergabe von Name, Vorname, Mobiletelefon-Nummer, E-Mail-Adresse, Passwort, Kreditkartennummer, Paypal-Account, IBAN und BIC des Nutzers soll ein PUT-Request mit korrektem Link gebaut werden. Dabei ist lediglich die E-Mail-Adresse unveränderlich.
Vorbedingung Die ID des Nutzer muss als Integer übergeben werden, alle anderen Attribute als Strings.
Nachbedingung Es wird ein Request zurückgeben. Dabei handelt es sich um ein PUT-Request mit HTTP-URL und einem Requestbody. In diesem Requestbody werden die Nutzerdaten übergeben.

Usermanager

Klasse UsermanagerTest

Package App/[..]/src/androidTest/java/com/taxi/pg/pimpmytaxifleet/rest

Signatur public void testGetUserByEMailAddressTrue(), public void testGetUserByEMailAddressFalseWrongPassword(), public void testGetUserByEMailAddressFalseWrongEMail(), public void testRegisterAndDeleteUser()

Beschreibung Testet, ob die Usermanager-Klasse, welche für alle interaktionen der Nutzerdaten mit der Datenbank zuständig ist, korrekt funktioniert.

Vorbedingung Muss als *androidTest* gestartet werden und somit entweder auf einem Android-Emulator oder einem physischen Gerät durchgeführt werden. Weiterhin muss eine Internetverbindung vorhanden sein, damit die Usermanager-Klasse auf die Datenbank zugreifen kann.

public void testGetUserByEMailAddressTrue() Bei dieser Methode wird ein spezifischer Nutzer anhand seiner E-Mail-Adresse und des dazugehörigen Passworts aus der Datenbank geholt.

Vorbedingung Es muss eine Instanz der Klasse Usermanager vorhanden sein. Weiterhin müssen sowohl E-Mail-Adresse als auch Passwort als String gegeben sein und das Passwort muss zu der E-Mail-Adresse in der Datenbank passen.

Nachbedingung Man erhält ein User-Objekt mit den Nutzerdaten des gewünschten Users.

public void testGetUserByEMailAddressFalseWrongPassword() Bei dieser Methode wird versucht einen Nutzer aus der Datenbank zu holen. Dabei ist die E-Mail-Adresse korrekt, das Passwort aber falsch und somit sollte kein Nutzer zurückgeliefert werden.

Vorbedingung Es muss eine Instanz der Klasse Usermanager vorhanden sein. Weiterhin müssen sowohl E-Mail-Adresse als auch Passwort als String gegeben sein und das Passwort darf nicht zu der E-Mail-Adresse in der Datenbank passen.

Nachbedingung Man erhält kein User-Objekt zurück.

public void testGetUserByEMailAddressFalseWrongEMail() Bei dieser Methode wird versucht einen Nutzer aus der Datenbank zu holen. Dabei ist die E-Mail-Adresse nicht in der Datenbank vorhanden und somit sollte kein Nutzer zurückgegeben werden.

Vorbedingung Es muss eine Instanz der Klasse Usermanager vorhanden sein. Weiterhin müssen sowohl E-Mail-Adresse als auch Passwort als String

gegeben sein und die E-Mail-Adresse darf nicht in der Datenbank vorhanden sein.
Nachbedingung Man erhält kein User-Objekt zurück.

public void testRegisterAndDeleteUser() Bei dieser Methode werden direkt zwei Methoden der Usermanager-Klasse getestet. Dabei wird zunächst ein neuer Nutzer mit seinen Nutzerdaten in der Datenbank registriert und dieser am Ende des Testfalls wieder aus der Datenbank gelöscht.
Vorbedingung Es muss eine Instanz der Klasse Usermanager vorhanden sein. Weiterhin müssen Vorname, Nachname, E-Mail-Adresse, Passwort und die Informationen über eine der Zahlungsmethoden als String gegeben sein.
Nachbedingung Zunächst wird ein User in der Datenbank angelegt und der Vorgang mit einem *Boolean true* bestätigt. Anschließend wird der User wieder gelöscht und dieser Vorgang ebenfalls mit einem *Boolean true* bestätigt.

User

Klasse UserTest

Package App/[..]/src/test/java/com/taxi/pg/pimpmytaxifleet/user

Signatur public void testGetIdClient(), public void testSetIdClient(), public void testGetForename(), public void testSetForename(), public void testGetSurname(), public void testSetSurname(), public void testGetMobile(), public void testSetMobile(), public void testGetEmail(), public void testSetEmail(), public void testGetCreditcard(), public void testSetCreditcard(), public void testGetPaypal(), public void testSetPaypal(), public void testGetIban(), public void testSetIban(), public void testGetBic(), public void testSetBic(), public void testEqualsSuccessful(), public void testEqualsFailed(), public void testEqualsSuccessfulAlthoughDifferentPassword()

Beschreibung In diesem Testfall wird die Funktion der Klasse *User* der Applikation getestet.

Vorbedingung Zu Beginn des Tests wird ein Testuser angelegt.

public void testGetIdClient() Bei dieser Methode ob die User-Klasse die korrekte UserID als Integer zurückgibt.
Vorbedingung Es muss ein Testnutzer angelegt sein.
Nachbedingung Es wird die ID als Integer zurückgegeben.

public void testSetIdClient() Bei dieser Methode wird für den Testuser eine neue ID gesetzt.
Vorbedingung Es muss ein Testnutzer angelegt sein und die neue ID des Nutzers muss als Integer übergeben werden.
Nachbedingung Es wird die neue ID des Nutzers als Integer zurückgegeben.

public void testGetForename() Bei dieser Methode ob die User-Klasse den korrekten Vornamen als String zurückgibt.
Vorbedingung Es muss ein Testnutzer angelegt sein.
Nachbedingung Es wird der Vorname als String zurückgegeben.

public void testSetForename() Bei dieser Methode wird für den Testuser ein neuer Vorname gesetzt.
Vorbedingung Es muss ein Testnutzer angelegt sein und der neue Vorname des Nutzers muss als String übergeben werden.
Nachbedingung Es wird der neue Vorname des Nutzers als String zurückgegeben.

public void testGetSurname() Bei dieser Methode ob die User-Klasse den korrekten Nachnamen als String zurückgibt.
Vorbedingung Es muss ein Testnutzer angelegt sein.
Nachbedingung Es wird der Nachname als String zurückgegeben.

public void testSetSurname() Bei dieser Methode wird für den Testuser ein neuer Nachname gesetzt.

Vorbedingung Es muss ein Testnutzer angelegt sein und der neue Nachname des Nutzers muss als String übergeben werden.

Nachbedingung Es wird der neue Nachname des Nutzers als String zurückgegeben.

public void testGetMobile() Bei dieser Methode ob die User-Klasse die korrekte Mobiltelefonnummer als String zurückgibt.

Vorbedingung Es muss ein Testnutzer angelegt sein.

Nachbedingung Es wird die Mobiltelefonnummer als String zurückgegeben.

public void testSetMobile() Bei dieser Methode wird für den Testuser eine neue Mobiltelefonnummer gesetzt.

Vorbedingung Es muss ein Testnutzer angelegt sein und die neue Mobiltelefonnummer des Nutzers muss als String übergeben werden.

Nachbedingung Es wird die neue Mobiltelefonnummer des Nutzers als String zurückgegeben.

public void testGetEmail() Bei dieser Methode ob die User-Klasse die korrekte E-Mail-Adresse als String zurückgibt.

Vorbedingung Es muss ein Testnutzer angelegt sein.

Nachbedingung Es wird die Mobiltelefonnummer als String zurückgegeben.

public void testSetEmail() Bei dieser Methode wird für den Testuser eine neue E-Mail-Adresse gesetzt.

Vorbedingung Es muss ein Testnutzer angelegt sein und die neue E-Mail-Adresse des Nutzers muss als String übergeben werden.

Nachbedingung Es wird die neue E-Mail-Adresse des Nutzers als String zurückgegeben.

public void testGetCreditcard() Bei dieser Methode ob die User-Klasse die korrekte Kreditkartennummer als String zurückgibt.

Vorbedingung Es muss ein Testnutzer angelegt sein und die Kreditkartennummer muss im User hinterlegt sein.

Nachbedingung Es wird die Kreditkartennummer als String zurückgegeben.

public void testSetCreditcard() Bei dieser Methode wird für den Testuser eine neue Kreditkartennummer gesetzt.

Vorbedingung Es muss ein Testnutzer angelegt sein und die neue Kreditkartennummer des Nutzers muss als String übergeben werden.

Nachbedingung Es wird die neue Kreditkartennummer des Nutzers als String zurückgegeben.

public void testGetPaypal() Bei dieser Methode ob die User-Klasse die korrekte PayPal-E-Mail-Adresse als String zurückgibt.

Vorbedingung Es muss ein Testnutzer angelegt sein und die PayPal-E-Mail-Adresse muss im User hinterlegt sein.

Nachbedingung Es wird die PayPal-E-Mail-Adresse als String zurückgegeben.

public void testSetPaypal() Bei dieser Methode wird für den Testuser eine neue PayPal-E-Mail-Adresse gesetzt.

Vorbedingung Es muss ein Testnutzer angelegt sein und die neue PayPal-E-Mail-Adresse des Nutzers muss als String übergeben werden.

Nachbedingung Es wird die neue PayPal-E-Mail-Adresse des Nutzers als String zurückgegeben.

public void testGetIban() Bei dieser Methode ob die User-Klasse den korrekte IBAN als String zurückgibt.

Vorbedingung Es muss ein Testnutzer angelegt sein und der IBAN muss im User hinterlegt sein.

Nachbedingung Es wird der IBAN als String zurückge-geben.

public void testSetIban() Bei dieser Methode wird für den Testuser ein neuer IBAN gesetzt.
Vorbedingung Es muss ein Testnutzer angelegt sein und der neue IBAN des Nutzers muss als String übergeben werden.
Nachbedingung Es wird der neue IBAN des Nutzers als String zurückgegeben.

public void testGetBic() Bei dieser Methode ob die User-Klasse den korrekte BIC als String zurückgibt.
Vorbedingung Es muss ein Testnutzer angelegt sein und der BIC muss im User hinterlegt sein.
Nachbedingung Es wird der BIC als String zurückgegeben.

public void testGetBic() Bei dieser Methode wird für den Testuser ein neuer BIC gesetzt.
Vorbedingung Es muss ein Testnutzer angelegt sein und der neue BIC des Nutzers muss als String übergeben werden.
Nachbedingung Es wird der neue BIC des Nutzers als String zurückgegeben.

public void testEqualsSuccessful() Bei dieser Methode wird die *equals-Methode* der Klasse *User* getestet. Dazu wird ein neuer User angelegt, der identisch mit dem Testuser ist. Anschließend werden beide Nutzer verglichen und als der identisch erkannt.
Vorbedingung Es muss ein Testnutzer angelegt sein.
Nachbedingung Es wird ein *Boolean true* zurückgeben.

public void testEqualsFailed() Bei dieser Methode wird die *equals-Methode* der Klasse *User* getestet. Dazu wird ein neuer User angelegt, der anders

ist als der Testuser. Anschließend werden beide Nutzer verglichen und als der unterschiedlich erkannt.
Vorbedingung Es muss ein Testnutzer angelegt sein.
Nachbedingung Es wird ein *Boolean false* zurückgeben.

public void testEqualsSuccessfulAlthoughDifferentPassword() Bei dieser Methode wird die *equals-Methode* der Klasse *User* getestet. Dazu wird ein neuer User angelegt, der identisch mit dem Testuser ist. Jedoch erhält der neu geschaffene User ein anderes Passwort als der Testuser. Anschließend werden beide Nutzer verglichen und als der identisch erkannt, trotz des unterschiedlichen Passworts.
Vorbedingung Es muss ein Testnutzer angelegt und die Passwörter der zwei Nutzer müssen unterschiedlich sein.
Nachbedingung Es wird ein *Boolean true* zurückgeben.

Traci

Klasse TraciTest

Package pgtaxi.test (Testpfad)

Signatur public void setUpBeforeClass(), public void afterTests(), public void convertGPSTest(), public void dataHandlerTest(), public void getVehiclesTest(), public void generateTripTest(), public void generateTrafficTest(), public void randomPositionTest(), public void realTimeTest(), public void vehicleHandlingTest(), public void getDistanceBetweenTest(), public void getGPSPositionTest(), public void otherTests(), public void addRouteTest(), public void setRouteTest(), public void getTeleportedTest(), public void stopVehicleTest()

Beschreibung Testet verschiedene Komponenten der Simulation, wobei die meisten TraCI Funktionen sind.

Vorbedingung Es muss dringend vorher die Methode setUpBeforeClass() ausgeführt werden, welche den TraCI Manager initialisiert und Sumo startet.

setUpBeforeClass() Startet Sumo und initialisiert den TraCI Manager. Zudem wird ein Zeitschritt durchgeführt.
Vorbedingung Keine.
Nachbedingung Sumo muss laufen. Manager darf keine Fehler erzeugen.

afterTests() Lässt die Simulation nach den Tests noch 500 Zeitschritte laufen, damit mögliche nachfolgende Fehler erkannt werden können.
Vorbedingung Die Simulation darf nicht vorher abgebrochen sein.
Nachbedingung Fehlerfreies Ende der Simulation.

convertGPSTest() Konvertiert einen GPS Punkt zu einer Edge von Sumo.
Vorbedingung Es müssen gültige GPS-Punkte sein.
Nachbedingung Die zurückgegebene Edge darf nicht null sein. Zudem muss die zurückgegebene Edge der Edge "35215789#6ëntsprechen.

dataHandlerTest() Testet die Zusammenstellung eines Info-Pakets, welches im realen Betrieb an Odysseus gesendet wird.
Vorbedingung Es müssen Taxis in der Simulation vorhanden sein.
Nachbedingung Gibt die ID und Position des ersten gefundenen Taxi aus.

getVehiclesTest() Gibt die Anzahl der derzeit geladenen Fahrzeuge aus.
Vorbedingung keine
Nachbedingung Der Test ist bestanden wenn ein beliebiger Integer Wert ausgegeben wird.

generateTripTest() Testet die Generierung eines zufälligen Trips.
Vorbedingung keine
Nachbedingung Es werden 5 unterschiedliche Trips erzeugt, die gültig sein

müssen.

generateTrafficTest() Testet die Generierung von Traffic innerhalb von Sumo.
Vorbedingung keine
Nachbedingung Es müssen 5 Fahrzeuge innerhalb von Sumo generiert werden. Es darf kein Abbruch der Simulation geschehen.

randomPositionTest() Testet den Generator für zufällige Positionen.
Vorbedingung keine
Nachbedingung Gibt eine zufällige Position zurück.

realTimeTest() Testet die Simulation in der Realzeit.
Vorbedingung Darf nur einzeln ausgeführt werden!
Nachbedingung Kein Abbruch der Simulation.

vehicleHandlingTest() Testet verschiedene Operationen um Werte eines Fahrzeuges zu verändern oder eins der Simulation hinzufügen oder zu löschen.
Vorbedingung Das Taxi t10 darf vorher nicht vorhanden sein.
Nachbedingung Das Taxi wurde erfolgreich hinzugefügt, verändert und dann wieder gelöscht.

getDistanceBetweenTest() Testet die Funktion um eine Distanz zwischen zwei Edges wiederzugeben.
Vorbedingung Edges müssen vorhanden sein.
Nachbedingung Die zurückgegebene Distanz darf nicht null sein.

getGPSPositionTest() Testet die Funktion eine GPS-Position eines Fahrzeuges zurück zu geben.
Vorbedingung Fahrzeug t0 muss vorhanden sein.
Nachbedingung Zurückgegebene GPS-Punkte müssen größer als 0.0 sein.

otherTests() Testet verschiedene kleine Funktionen.

Vorbedingung Angegeben Edges müssen vorhanden sein.

Nachbedingung Es gibt Ausgaben.

addRouteTest() Testet das hinzufügen einer Route zur Simulation.

Vorbedingung Der Punkt hq muss vorhanden sein.

Nachbedingung Kein Abbruch der Simulation.

setRouteTest() Testet das setzen einer Route für ein Fahrzeug.

Vorbedingung Fahrzeug und Edges müssen vorhanden sein.

Nachbedingung Kein Abbruch und Fahrzeug t0 verändert seine Route.

getTeleportedTest() Testet die Funktion um Fahrzeuge zu erfassen, die teleportiert wurden.

Vorbedingung Keine.

Nachbedingung Kein Abbruch der Simulation.

stopVehicleTest() Testet das anhalten und weiterfahren eines Fahrzeuges.

Vorbedingung Das Fahrzeug t0 und die Edges müssen vorhanden sein.

Nachbedingung Fahrzug stoppt.

Testhistory

Erfolgreich: Frederik Preuß, 07.10.2016.

Traci Langzeit

Klasse TraciTest

Package pgtaxi.test (Testpfad)

Signatur public void setUpBeforeClass(), public void afterTests(), public void longTermRunTest()

Beschreibung Testet einen Langzeit Lauf der Verkehrssimulation.

Vorbedingung Es muss dringend vorher die Methode setUpBeforeClass() ausgeführt werden, welche den TraCI Manager initialisiert und Sumo startet.

setUpBeforeClass() Startet Sumo und initialisiert den TraCI Manager. Zudem wird ein Zeitschritt durchgeführt. **Vorbedingung** Keine.
Nachbedingung Sumo muss laufen. Manager darf keine Fehler erzeugen.

afterTests() Lässt die Simulation nach den Tests noch 500 Zeitschritte laufen, damit mögliche nachfolgende Fehler erkannt werden können. **Vorbedingung** Die Simulation darf nicht vorher abgebrochen sein.
Nachbedingung Fehlerfreies Ende der Simulation.

longTermRunTest() Testet die Simulation in einem Langzeit Test. **Vorbedingung** keine
Nachbedingung Kein Abbruch der Simulation.

Testhistory

Erfolgreich: Frederik Preuß, 07.10.2016.

Traci Tracking

Klasse TraciTest

Package pgtaxi.test (Testpfad)

Signatur public void setUpBeforeClass(), public void afterTests(), public void startRunning()

Beschreibung Damit in der App getestet werden kann, ob die Tracking Funktion von Taxis funktioniert wurde dieser Testfall entworfen.

Vorbedingung Es muss dringend vorher die Methode setUpBeforeClass() ausgeführt werden, welche den TraCI Manager initialisiert und Sumo startet.

setUpBeforeClass() Startet Sumo und initialisiert den TraCI Manager. Zudem wird ein Zeitschritt durchgeführt. **Vorbedingung** Keine.
Nachbedingung Sumo muss laufen. Manager darf keine Fehler erzeugen.

afterTests() Lässt die Simulation nach den Tests noch 500 Zeitschritte laufen, damit mögliche nachfolgende Fehler erkannt werden können. **Vorbedingung** Die Simulation darf nicht vorher abgebrochen sein.
Nachbedingung Fehlerfreies Ende der Simulation.

startRunning() Initialisiert eine abgespeckte Version von TraciRun mit Kommunikation zu Odysseus. **Vorbedingung** Keine.
Nachbedingung Kein Abbruch. In Odysseus müssen Daten ankommen.

Testhistory

Erfolgreich: Frederik Preuß, 15.10.2016.

Traci Traffic

Klasse TraciTest

Package pgtaxi.test (Testpfad)

Signatur public void setUpBeforeClass(), public void afterTests(), public void generateTrafficToFile()

Beschreibung Hierbei handelt es sich weniger um einen test, sondern um ein Tool, dass einen Verkehrsfluss erstellt.

Vorbedingung Es muss dringend vorher die Methode setUpBeforeClass() ausgeführt werden, welche den TraCI Manager initialisiert und Sumo startet.

setUpBeforeClass() Startet Sumo und initialisiert den TraCI Manager. Zudem wird ein Zeitschritt durchgeführt. **Vorbedingung** Keine.
Nachbedingung Sumo muss laufen. Manager darf keine Fehler erzeugen.

afterTests() Lässt die Simulation nach den Tests noch 500 Zeitschritte laufen, damit mögliche nachfolgende Fehler erkannt werden können. **Vorbedingung** Die Simulation darf nicht vorher abgebrochen sein.
Nachbedingung Fehlerfreies Ende der Simulation.

generateTrafficToFile() Generiert die angegebene Anzahl von Fahrzeugen mit den angegebenen Zeitschritten. **Vorbedingung** Keine.
Nachbedingung Kein Abbruch. Die Datei muss erstellt worden sein.

Testhistory

Erfolgreich: Frederik Preuß, 15.10.2016.

FleetManager

Klasse FleetManagerTest

Package de.uniol.inf.is.odysseus.pgtaxi.test.vehiclefleet;

Signatur public void setUp(), public void testVehicleContainsPassenger(), public void testVehicleContainsNoPassenger()

Beschreibung Testfall für die Abfrage, ob ein Taxi Passagiere transportiert oder nicht.

Vorbedingung Es muss ein Taxi-Objekt erstellt und dem FleetManager zugeordnet worden sein.

public void testVehicleContainsPassenger() :

Vorbedingung Das Taxi muss mindestens einen Passagier transportieren.
Nachbedingung Der Wahrheitswert *true* muss zurückgeliefert worden sein.

public void testVehicleContainsNoPassenger() :

Vorbedingung Das Taxi darf keinen Passagier transportieren.
Nachbedingung Der Wahrheitswert *false* muss zurückgeliefert worden sein.

Testhistory

Erfolgreich: Michael Kroeger, 25.10.2016.

RoutingRequests

Klasse RoutingRequestsTest

Package de.uniol.inf.is.odysseus.pgtaxi.test.vehiclefleet;

Signatur public void testSetup(), public void testStartRoutingRequest(), public void testGetArrTimePickup(), public void testGetArrTimeDeliver(), testSetVehicles(), public void testUpdateTaxiLocation(), public void testGetStepOversPassenger(), public void testGetStopOversVehicle(), public void testGetVehicleName(), public void testNumberOfJobs(), public void testGetFreeVehicles()

Beschreibung Testet, ob Routing-Anfragen korrekt bearbeitet und die richtigen Informationen über die berechneten Routen und Auftragsdaten bereitgestellt werden.

Vorbedingung Die setUp-Methode muss zuerst ausgeführt worden sein.

public void testSetup() :

Erstellt die Testumgebung.

Nachbedingung Die Fahrzeugflotte wurde erstellt.

public void testStartRoutingRequest() :

Vorbedingung testSetup() wurde ausgeführt.

Nachbedingung Kundenanfragen wurden mit allen notwendigen Informationen gestellt und die Routen/Auftragsinformationen wurden berechnet.

public void testGetArrTimePickup() :

Vorbedingung testSetup() und testStartRoutingRequest() wurden ausgeführt.

Nachbedingung Die Ankunftszeit am Abholort des Kunden muss zurückgelifert worden sein.

public void testGetArrTimeDeliver() :

Vorbedingung testSetup() und testStartRoutingRequest() wurden ausgeführt.

Nachbedingung Die Ankunftszeit am Ziel des Kunden muss zurückgeliefert worden sein.

public void testSetVehicles() :

Vorbedingung Keine.

Nachbedingung Die Fahrzeuge müssen zum *FleetManager* hinzugefügt worden sein.

public void testUpdateTaxiLocation() :

Vorbedingung testSetVehicles() muss ausgeführt worden sein.

Nachbedingung Die Fahrzeuge im *FleetManager* müssen entsprechend der

Angaben aktualisiert worden sein.

public void testGetStepOversPassenger() :

Vorbedingung testSetup() und testStartRoutingRequest() wurden ausgeführt.

Nachbedingung Die Zwischenpunkte der Passagier-Route müssen ausgegeben worden sein.

public void testGetStopOversVehicle() :

Vorbedingung testSetup() und testStartRoutingRequest() wurden ausgeführt.

Nachbedingung Die Zwischenpunkte der Taxi-Route müssen ausgegeben werden.

public void testGetVehicleName() :

Vorbedingung testSetup() und testStartRoutingRequest() wurden ausgeführt.

Nachbedingung Der Name, des dem Kunden zugeordneten Taxis, muss zurückgelieferrt worden sein.

public void testNumberOfJobs() :

Vorbedingung testSetup() und testStartRoutingRequest() wurden ausgeführt.

Nachbedingung Die Anzahl, der in der Routenoptimierung parallel betrachteten Aufträge, muss zurückgelieferrt worden sein.

public void testGetFreeVehicles() :

Vorbedingung testSetup() und testStartRoutingRequest() wurden ausgeführt.

Nachbedingung Die Anzahl, der freien (nicht für einen Auftrag eingeplanten) Fahrzeuge, muss zurückgeliefert worden sein.

public void testGetTaxiPositions() :

Vorbedingung testSetup() und testStartRoutingRequest() wurden ausgeführt.

Nachbedingung Die Positionen (Längen- und Breitengrad) aller Taxis aus der Fahrzeugflotte müssen zurückgeliefert worden sein.

public void testRemoveTaxi() :

Vorbedingung testVehicles() muss ausgeführt worden sein.

Nachbedingung Das angegebene Taxi wurde aus der Fahrzeugflotte entfernt.

Testhistory

Erfolgreich: Michael Kroeger, 21.11.2016.

Quarter

Klasse QuarterTest

Package de.uniol.inf.is.odysseus.pgtaxi.test;

Signatur public void testIfPointIsInQuarterIsTrue(), public void testIfPointIsInQuarterIsFalse(), public void testIfQuarterManagerCreatesCorrectNumberOfQuarters(), public void testIfReveivedOrderIsSortedCorrectly(), public void testIfQuarterManagerDecrementsQuarterWhenorderLeavesTimeWin -dow(), public void testIfQuarterIsActivatedCorrectly

Beschreibung Testet, ob bei Klassen *Quarter* und *QuarterManager* zu dynamischen Standortplanung korrekt funktionieren.

Vorbedingung Keine

public void testIfPointIsInQuarterIsTrue() Testet ob ein gegebener Punkt der in einem Quarter liegt auch so erkannt wird.

Vorbedingung Der QuarterManager mit Quartern ist angelegt und es werden Koordinaten übergeben, die in das Quarter fallen.

Nachbedingung Das Quarter erkennt den Punkt korrekt.

public void testIfPointIsInQuarterIsFalse() Testet ob ein gegebener Punkt der nicht in einem Quarter liegt auch so erkannt wird.

Vorbedingung Der QuarterManager mit Quartern ist angelegt und es werden Koordinaten übergeben, die nicht in das Quarter fallen.

Nachbedingung Das Quarter erkennt dass der Punkt nicht innerhalb seiner Fläche liegt.

public void testIfQuarterManagerCreatesCorrectNumberOfQuarters() Testet ob beim anlegen des QuarterManagers die korrekte Anzahl an Quartern erstellt wird.

Vorbedingung keine

Nachbedingung Bei der Erstellung des QuarterManagers werden ingesamt 15 Quarter angelegt.

public void testIfReveivedOrderIsSortedCorrectly() Testet ob ein neuer Auftrag in das korrekte Quarter des QuarterManager einsortiert und der jeweilige Zähler inkrementiert wird.

Vorbedingung Es muss bereits eine Instanze des QuarterManagers angelegt sein.

Nachbedingung Der Zähler des Quarters steht nun auf 1 und das Quarter ist nicht aktiviert.

public void testIfQuarterManagerDecrementsQuarterWhenorder[..] Testet ob der Zählen in dem Quarter dekrementiert wird sobald ein Auftrag das Zeitfenster verlässt.

Vorbedingung Es muss bereits eine Instanze des QuarterManagers angelegt sein. Zunächst muss der Zähler im Quarter inkrementiert werden.
Nachbedingung Der Zähler des Quarters wird dekrementiert und steht nun auf 0.

public void testIfQuarterIsActivatedCorrectly() Testet ob ein Quarter inkrementiert wird sobald der Schwellwert erreicht wird.
Vorbedingung Es muss bereits eine Instanze des QuarterManagers angelegt sein.
Nachbedingung Der Zähler des Quarters steht auf 5 und stehtdas Quarter ist aktiviert.

Testhistory

Evaluation

Auftragsliste für erste Sumo Evaluation (gekürzt)

id	entrydate	startdate	startPosLng	startPosLat	endPosLong	endPosLat	clientID	amount
1	1490787741028	1490787741028	8.21277488118714	53.12952573927131	8.19769581597788	53.14225610334158	1	4
2	1490787746784	1490787746784	8.222671	53.14463	8.211483332248088	53.164627887989816	2	2
3	1490787750641	1490787750641	8.224491396990803	53.11964203964271	8.238858700161774	53.096007872773086	3	3
5	1490787764194	1490787764194	8.216645	53.138616	8.22994599340492	53.15879137997389	5	4
6	1490787767593	1490787767593	8.210098	53.139881	8.225177767667562	53.11878387842672	6	3
7	1490787772714	1490787772714	8.212560773212452	53.16241404636765	8.199007651281525	53.13867779943485	7	3
8	1490787775530	1490787775530	8.226636	53.146896	8.243157890732139	53.13037013589138	8	4
9	1490787788598	1490787788598	8.222671	53.14463	8.24004497260828	53.12490313957525	9	3
10	1490787797020	1490787797020	8.180293911298891	53.15323453093097	8.192690295029891	53.16833223797771	10	2
11	1490787801951	1490787801951	8.209736525946681	53.16242695844837	8.201548700450155	53.18219329888802	11	2
12	1490787809424	1490787809424	8.20351566847562	53.122858599484	8.216562695031785	53.14128236243486	12	4
13	1490787826491	1490787826491	8.208886238044029	53.13381070676388	8.185895025554712	53.123804345687134	13	4
14	1490787838885	1490787838885	8.208769351742825	53.158756126155836	8.21704572796038	53.14987542311172	14	1
15	1490787848587	1490787848587	8.162285025250757	53.14712912430411	8.1718712196948	53.1232080389174	15	1
16	1490787857217	1490787857217	8.210098	53.139881	8.193967697360327	53.16003783579469	16	1
17	1490787872085	1490787872085	8.188318967377555	53.1219864496168	8.179888198161171	53.14575886069717	17	3
18	1490787880145	1490787880145	8.222671	53.14463	8.20247616996053	53.12694180171575	18	3
19	1490787895093	1490787895093	8.210330234097682	53.14025908104885	8.192067424539218	53.11625975863617	19	4
20	1490787902415	1490787902415	8.203053388520322	53.110992683239644	8.222453482446253	53.091169329765805	20	1
21	1490787927087	1490787927087	8.206116229743158	53.14833491753589	8.217888390431492	53.15962912258533	21	3
22	1490787930093	1490787930093	8.226636	53.146896	8.244580930105663	53.1665124325484	22	1
23	1490787932797	1490787932797	8.209312022301342	53.110034284918	8.228305736150507	53.09694042203945	23	2
24	1490787945301	1490787945301	8.214990052798111	53.1480356059918	8.203136174293158	53.16900373772718	24	3
25	1490787978539	1490787978539	8.226636	53.146896	8.21347608651256	53.15793984544483	25	3
26	1490787987533	1490787987533	8.190591194053711	53.1502532656398	8.210674462025166	53.13824477472156	26	2
27	1490788001065	1490788001065	8.180791734475067	53.11548320398703	8.169200275551429	53.09396466168896	27	4
28	1490788008636	1490788008636	8.198845182972986	53.134438277815846	8.189905017771352	53.11515093134205	28	1
29	1490788042694	1490788042694	8.173596533670048	53.163420419700884	8.16006466944354	53.153718008559466	29	4
30	1490788055840	1490788055840	8.20428275536419	53.156843827329595	8.215319506413305	53.17313847694899	30	1
31	1490788086264	1490788086264	8.210098	53.139881	8.218583347933514	53.16284981198877	31	4
32	1490788096523	1490788096523	8.218398465325734	53.113791111955145	8.23088387019513	53.13559288305415	32	2
33	1490788104780	1490788104780	8.194864644001932	53.144713187940006	8.18027040860879	53.167437604172214	33	2
34	1490788130588	1490788130588	8.226636	53.146896	8.2032929419597	53.1270410396253	34	2
35	1490788135999	1490788135999	8.216071873468287	53.11245872316337	8.23745011211178	53.129544015432664	35	2
36	1490788142570	1490788142570	8.173413568252855	53.13346486447091	8.190605573687435	53.14461523436317	36	4
37	1490788148160	1490788148160	8.227398757659095	53.135807087719805	8.237898315215832	53.11231016433825	37	3
38	1490788165551	1490788165551	8.229011687483885	53.14534683048027	8.216794366736321	53.16710540577509	38	2
39	1490788171664	1490788171664	8.231456875364762	53.14463490170987	8.240101347338035	53.13432525029997	39	4
40	1490788181460	1490788181460	8.211319	53.141989	8.231530151540507	53.120465672852056	40	1
41	1490788184032	1490788184032	8.206801336012738	53.13806257425914	8.230511391493755	53.115937116819474	41	3
42	1490788213900	1490788213900	8.161340238608847	53.12987044009156	8.140051224757173	53.10606640549975	42	2
43	1490788238751	1490788238751	8.194544412827097	53.10361530796577	8.182764583861234	53.12003159928883	43	3
44	1490788260110	1490788260110	8.222671	53.14463	8.205714987658725	53.136576631866994	44	3
45	1490788262613	1490788262613	8.226636	53.146896	8.21342972462736	53.15854345145474	45	1
46	1490788264727	1490788264727	8.202006732678848	53.11777374718992	8.18477728951033	53.137256392969576	46	1
47	1490788288200	1490788288200	8.180995573265397	53.14367023929614	8.172672621755622	53.12766693114342	47	1
48	1490788292083	1490788292083	8.227696506604255	53.12796850109746	8.243437275172134	53.141990457055705	48	3
49	1490788300857	1490788300857	8.213152193102594	53.11186090156406	8.194162823021406	53.09649070953478	49	2
50	1490788311175	1490788311175	8.220779887301443	53.11874557476298	8.206763284937368	53.10859565756768	50	4

Auftragsliste für zweiten Sumo Evaluation (gekürzt)

id	entrydate	startdate	startPosLng	startPosLat	endPosLong	endPosLat	clientID	amount
1	1490787741028	1490787741028	8.21277488118714	53.12952573927131	8.19769581597788	53.14225610334158	1	4
2	1490787746784	1490787746784	8.222671	53.14463	8.211483332248088	53.164627887989816	2	2
3	1490787750641	1490787750641	8.224491396990803	53.11964203964271	8.238858700161774	53.096007872773086	3	3
4	1490787755558	1490787755558	8.197173257019198	53.149468546033724	8.211611584448873	53.14029554112185	4	4
5	1490787764194	1490787764194	8.216645	53.138616	8.22994599340492	53.15879137997389	5	4
6	1490787767593	1490787767593	8.210098	53.139881	8.225177767667562	53.11878387842672	6	3
7	1490787772714	1490787772714	8.212560773212452	53.16241404636765	8.199007651281525	53.13867779943485	7	3
8	1490787775530	1490787775530	8.226636	53.146896	8.243157890732139	53.13037013589138	8	4
9	1490787788598	1490787788598	8.222671	53.14463	8.24004497260828	53.12490313957525	9	3
10	1490787797020	1490787797020	8.180293911298891	53.15323453093097	8.192690295029891	53.16833223797771	10	2
11	1490787801951	1490787801951	8.209736525946681	53.16242695844837	8.201548700450155	53.18219329888802	11	2
12	1490787809424	1490787809424	8.20351566847562	53.122858599484	8.216562695031785	53.14128236243486	12	4
13	1490787826491	1490787826491	8.208886238044029	53.13381070676388	8.185895025554712	53.123804345687134	13	4
14	1490787838885	1490787838885	8.208769351742825	53.158756126155836	8.21704572796038	53.14987542311172	14	1
15	1490787848587	1490787848587	8.162285025250757	53.14712912430411	8.1718712196948	53.1232080389174	15	1
16	1490787857217	1490787857217	8.210098	53.139881	8.193967697360327	53.16003783579469	16	1
17	1490787872085	1490787872085	8.188318967377555	53.1219864496168	8.179888198161171	53.14575886069717	17	3
18	1490787880145	1490787880145	8.222671	53.14463	8.20247616996053	53.12694180171575	18	3
19	1490787895093	1490787895093	8.210330234097682	53.14025908104885	8.192067424539218	53.11625975863617	19	4
20	1490787902415	1490787902415	8.203053388520322	53.110992683239644	8.222453482446253	53.091169329765805	20	1
21	1490787927087	1490787927087	8.206116229743158	53.14833491753589	8.217888390431492	53.15962912258533	21	3
22	1490787930093	1490787930093	8.226636	53.146896	8.244580930105663	53.1665124325484	22	1
23	1490787932797	1490787932797	8.209312022301342	53.110034284918	8.228305736150507	53.09694042203945	23	2
24	1490787945301	1490787945301	8.214990052798111	53.1480356059918	8.203136174293158	53.16900373772718	24	3
25	1490787978539	1490787978539	8.226636	53.146896	8.21347608651256	53.15793984544483	25	3
26	1490787987533	1490787987533	8.190591194053711	53.1502532656398	8.210674462025166	53.13824477472156	26	2
27	1490788001065	1490788001065	8.180791734475067	53.11548320398703	8.169200275551429	53.09396466168896	27	4
28	1490788008636	1490788008636	8.198845182972986	53.134438277815846	8.189905017771352	53.11515093134205	28	1
29	1490788042694	1490788042694	8.173596533670048	53.163420419700884	8.16006466944354	53.153718008559466	29	4
30	1490788055840	1490788055840	8.20428275536419	53.156843827329595	8.215319506413305	53.17313847694899	30	1
31	1490788086264	1490788086264	8.210098	53.139881	8.218583347933514	53.16284981198877	31	4
32	1490788096523	1490788096523	8.218398465325734	53.113791111955145	8.23088387019513	53.13559288305415	32	2
33	1490788104780	1490788104780	8.194864644001932	53.144713187940006	8.18027040860879	53.167437604172214	33	2
34	1490788130588	1490788130588	8.226636	53.146896	8.2032929419597	53.1270410396253	34	2
35	1490788135999	1490788135999	8.216071873468287	53.11245872316337	8.23745011211178	53.129544015432664	35	2
36	1490788142570	1490788142570	8.173413568252855	53.13346486447091	8.190605573687435	53.14461523436317	36	4
37	1490788148160	1490788148160	8.227398757659095	53.135807087719805	8.237898315215832	53.11231016433825	37	3
38	1490788165551	1490788165551	8.229011687483885	53.14534683048027	8.216794366736321	53.16710540577509	38	2
39	1490788171664	1490788171664	8.231456875364762	53.14463490170987	8.240101347338035	53.13432525029997	39	4
40	1490788181460	1490788181460	8.211319	53.141989	8.231530151540507	53.120465672852056	40	1
41	1490788184032	1490788184032	8.206801336012738	53.13806257425914	8.230511391493755	53.115937116819474	41	3
42	1490788213900	1490788213900	8.161340238608847	53.12987044009156	8.140051224757173	53.10606640549975	42	2
43	1490788238751	1490788238751	8.194544412827097	53.10361530796577	8.182764583861234	53.12003159928883	43	3
44	1490788260110	1490788260110	8.222671	53.14463	8.205714987658725	53.136576631866994	44	3
45	1490788262613	1490788262613	8.226636	53.146896	8.21342972462736	53.15854345145474	45	1
46	1490788264727	1490788264727	8.202006732678848	53.11777374718992	8.18477728951033	53.137256392969576	46	1
47	1490788288200	1490788288200	8.180995573265397	53.14367023929614	8.172672621755622	53.12766693114342	47	1
48	1490788292083	1490788292083	8.227696506604255	53.12796850109746	8.243437275172134	53.141990457055705	48	3
49	1490788300857	1490788300857	8.213152193102594	53.11186090156406	8.194162823021406	53.09649070953478	49	2
50	1490788311175	1490788311175	8.220779887301443	53.11874557476298	8.206763284937368	53.10859565756768	50	4

Auftragsliste für dritten Sumo Evaluation (gekürzt)

id	entrydate	startdate	startPosLng	startPosLat	endPosLong	endPosLat	clientID	amount
1	1490787746784	1490787746784	8.222671	53.14463	8.211483332248088	53.164627887989816	2	2

Erste Latenzmessung (Auftrag pro Sekunde)

amount	start	end	minIstart	maxIstart	Iend	latency	Measurements
3	1491052418759		134431986183060	134431986183060	134441999317541	10013134481	[]
2	1491052423837		134437064192173	134437064192173	134443934958062	6870765889	[]
4	1491052440601		134453828574480	134453828574480	134460483186908	6654612428	[]
1	1491052454627		134467853427012	134467853427012	134474830517406	6977090394	[]
2	1491052508236		134521463174441	134521463174441	134528407090001	6943915560	[]
4	1491052520058		134533285835072	134533285835072	134540387440976	7101605904	[]
2	1491052547429		134560655034002	134560655034002	134568041848761	7386814759	[]
3	1491052567051		134580278246744	134580278246744	134588171604168	7893357424	[]
2	1491052585268		134598494163424	134598494163424	134606708114964	8213951540	[]
3	1491052616278		134629504803909	134629504803909	134637560781537	8055977628	[]
3	1491052624481		134637707622498	134637707622498	134646167100607	8459478109	[]
3	1491052684682		134697908735888	134697908735888	134706771360960	8862625072	[]
4	1491052689140		134702367016859	134702367016859	134710791746794	8424729935	[]
1	1491052748534		134761761531012	134761761531012	134771830996639	10069465627	[]
2	1491052778636		134791863146840	134791863146840	134801824400805	9961253965	[]
1	1491052823709		134836935045969	134836935045969	134847419959818	10484913849	[]
4	1491052848046		134861273379268	134861273379268	134871596683839	10323304571	[]
1	1491052917407		134930634168934	134930634168934	134941565320544	10931151610	[]
3	1491052936648		134949874629527	134949874629527	134961138864142	11264234615	[]
4	1491052982040		134995266948377	134995266948377	135006765923981	11498975604	[]

Zweite Latenzmessung (Auftrag pro 10 Sekunden)

amount	start	end	minIstart	maxIstart	Iend	latency	Measurements
3	1491053299459		135312685291375	135312685291375	135322208311809	9523020434	[]
1	1491053343212		135356438769949	135356438769949	135363324025784	6885255835	[]
1	1491053411763		135424989470579	135424989470579	135431760004934	6770534355	[]
4	1491053495707		135508933812912	135508933812912	135515278432955	6344620043	[]
2	1491053598038		135611264438784	135611264438784	135618664786913	7400348129	[]
1	1491053641501		135654727912864	135654727912864	135662126796355	7398883491	[]
3	1491053719106		135732332585811	135732332585811	135740004517739	7671931928	[]
2	1491053839457		135852684057373	135852684057373	135860475150067	7791092694	[]
3	1491053873918		135887145070540	135887145070540	135895366018221	8220947681	[]
2	1491053937462		135950688888458	135950688888458	135959208976780	8520088322	[]
1	1491054004101		136017328163540	136017328163540	136026072682253	8744518713	[]
1	1491054110741		136123968277989	136123968277989	136132803979051	8835701062	[]
3	1491054248894		136262120321984	136262120321984	136271458943106	9338621122	[]
2	1491054375168		136388395401726	136388395401726	136398061809714	9666407988	[]
2	1491054480368		136493595611473	136493595611473	136503631260926	10035649453	[]

Dritte Latenzmessung (Auftrag pro Minute)

amount	start	end	minIstart	maxIstart	Iend	latency	Measurements
1	1491056400187		138413414020448	138413414020448	138422942777972	9528757524	[]
2	1491056573815		138587042227327	138587042227327	138593990548145	6948320818	[]
3	1491056764268		138777494968762	138777494968762	138784524600863	7029632101	[]
2	1491056988706		139001932642714	139001932642714	139008994418873	7061776159	[]
4	1491057156345		139169572285965	139169572285965	139176550869354	6978583389	[]
4	1491057357140		139370367068847	139370367068847	139377154097985	6787029138	[]
2	1491057542467		139555694175145	139555694175145	139562516344728	6822169583	[]
1	1491057729815		139743042673950	139743042673950	139750313674067	7271000117	[]
3	1491057941331		139954558206250	139954558206250	139961696978232	7138771982	[]
3	1491058136040		140149266324996	140149266324996	140157055952106	7789627110	[]
1	1491058331391		140344618203481	140344618203481	140353076859709	8458656228	[]
1	1491058582015		140595241816529	140595241816529	140603824233971	8582417442	[]
1	1491058829430		140842656935338	140842656935338	140851217550987	8560615649	[]
4	1491059801648		141814874738229	141814874738229	141824522463692	9647725463	[]
2	1491060637592		142650818898605	142650818898605	142662249515450	11430616845	[]