FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Entwicklung einer App zur Fahrscheinoptimierung anhand des Beispiels Verkehrsverbund Rhein-Ruhr

BACHELORARBEIT

Zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)
im Studiengang Informatik

FRIEDRICH-SCHILLER-UNIVERSITÄT JENA

Fakultät für Mathematik und Informatik
Lehrstuhl für Softwaretechnik

eingereicht von
M[redacted] G[redacted]
geboren am 29.04.1996 in [redacted]

Gutachter:	Prof. Dr. W. Rossak, M. Mauch
Betreuer:	M. Mauch
Beginn der Arbeit:	25. Mai 2020
Ende der Arbeit:	30. September 2020

Kurzfassung

Bei einem Besuch in meiner Heimat stellte sich das Tarifsystem des Verkehrsverbundes Rhein-Ruhr für mehrere Fahrten als undurchsichtig heraus. Daraus entstand die Idee, eine App zu entwickeln, welche für den Nutzer die kostengünstigsten Fahrscheine ermittelt.
In dieser Arbeit wird die Optimierung von Fahrscheinen für den öffentlichen Personennahverkehr exemplarisch für den Verkehrsverbund Rhein-Ruhr erfolgen. Dabei wird der gesamte Verbundraum berücksichtigt. Daraus resultiert, dass auch verschiedene Preisstufen abgedeckt werden müssen.
Damit die Kosten der Fahrscheine optimiert werden können, muss der Nutzer zunächst die Fahrten planen, für welche die Fahrscheine benötigt werden. Da das Ziel der Arbeit der Entwurf der Optimierung ist, liegt der Fokus nicht auf dem Design der Anwendung. Zur Inspiration wurden verschiedene bereits existierende Anwendungen verglichen. Des Weiteren wurden zwei bestehende Fahrscheinberatungssysteme betrachtet, um Kriterien für die Optimierung zu entwickeln.
Die Entwicklung der App erfolgte nach einem angepassten inkrementellen Entwicklungsmodell. Dafür wurden zunächst im groben Systementwurf die ermittelten Kriterien und Anforderungen berücksichtigt. Anschließend wurden einzelne Komponenten detailliert entworfen, realisiert und getestet. Überprüft wurde einerseits die Funktionalität der App, andererseits wurde die Optimierung mehrerer realer Fahrten mit einer manuellen Optimierung verglichen. Außerdem wurde die Akzeptanz durch Nutzer mit einer kleinen Stichprobe überprüft.

Inhaltsverzeichnis

1 Einführung **1**
- 1.1 Motivation . . . 1
- 1.2 Grundidee . . . 1
- 1.3 Ziel der Arbeit . . . 2
- 1.4 Aufbau der Arbeit . . . 2

2 Grundlagen **3**
- 2.1 Verkehrsverbund Rhein Ruhr . . . 3
  - 2.1.1 Tarifsystem . . . 4
  - 2.1.2 Fahrscheine . . . 6
- 2.2 Vorhandene Anwendungen . . . 7
- 2.3 Vorhandene Ticketberater . . . 8
  - 2.3.1 Angebote des VRRs . . . 8
  - 2.3.2 FAIRTIQ . . . 9

3 Analyse **11**
- 3.1 Vergleich bestehender Fahrtenplaner für den ÖPNV . . . 11
  - 3.1.1 Routenoptionen . . . 11
  - 3.1.2 Übersichtlichkeit . . . 12
  - 3.1.3 Konsequenzen für die Anwendung . . . 17
- 3.2 Vorhandene Ticketberater . . . 18
  - 3.2.1 VRR online Ticketberater . . . 18
  - 3.2.2 FAIRTIQ . . . 19
  - 3.2.3 Schlussfolgerungen für die Ticketberatung . . . 20
- 3.3 Analyse der Fahrscheine . . . 20
  - 3.3.1 Rabattberechnung . . . 21
  - 3.3.2 Einzelfahrscheine . . . 21
  - 3.3.3 Zeitfahrscheine . . . 21
  - 3.3.4 Konsequenzen aus den Fahrscheinen . . . 23
- 3.4 Optimierungsansätze . . . 24
  - 3.4.1 Konsequenzen für die Optimierung . . . 24
- 3.5 Schlussfolgerungen . . . 25

4 Entwurf **27**
- 4.1 Anwendungsfalldiagramm . . . 27
- 4.2 Aktivitätsdiagramme . . . 30
  - 4.2.1 Mehrere Fahrten planen . . . 30

4.2.2 Einzelne Fahrt planen . . . 31
4.2.3 Fahrt bearbeiten . . . 32
4.3 Klassendiagramm . . . 36
4.3.1 Klassen aus Öffi . . . 36
4.3.2 Fahrscheine . . . 39
4.4 Layout Entwurf . . . 40
4.5 Geplantes Implementierungsvorgehen . . . 42
4.5.1 Komponententest . . . 43
4.5.2 Integrationstest . . . 44
4.5.3 Systemtest . . . 44

5 Organisatorische Umsetzung und Implementierung 45
5.1 Planung von Fahrten . . . 45
5.1.1 Planung einer Fahrt . . . 45
5.1.2 Planung mehrerer Fahrten . . . 47
5.2 Optimierung mit Einzelfahrscheinen . . . 48
5.2.1 Entwurf . . . 48
5.2.2 Realisierung . . . 49
5.3 Generalisierung . . . 52
5.4 Verwendung alter Fahrscheine . . . 54
5.4.1 Entwurf . . . 54
5.4.2 Realisierung . . . 55
5.5 Optimierung mit Zeitfahrscheinen . . . 57
5.5.1 Entwurf . . . 57
5.5.2 Generalisierbarkeit . . . 67
5.5.3 Realisierung . . . 71
5.6 Ablauf der Optimierung . . . 74
5.7 Aktualisieren von Verbindungsdaten . . . 78

6 Evaluation 81
6.1 Systemtests . . . 81
6.1.1 Abschließender Systemtest . . . 81
6.1.2 Vergleich der App mit den Anforderungen . . . 82
6.2 Optimierungsvergleich . . . 85
6.3 Akzeptanztest . . . 86

7 Fazit 87
7.1 Zusammenfassung der Entwicklung . . . 87
7.2 Resümee . . . 87
7.3 Ausblick . . . 88

A Analyse 91
A.1 Vergleich bestehender Anwendungen . . . 91
A.2 Ersparnisse Zeitfahrscheine . . . 93

B Optimierungsentwürfe **95**
B.1 Funktionsentwürfe 95
B.2 Vollständiges Klassendiagramm 100
B.3 Realisiertes Layout 103

C Testen **105**
C.1 Fragebogen für den Akzeptanztest 105
C.2 Überprüfung der Optimierung 108

Literaturverzeichnis **117**

Abbildungsverzeichnis **120**

Tabellenverzeichnis **121**

Abkürzungsverzeichnis

ÖPNV Öffentlicher Personennahverkehr

SPNV Schienenpersonennahverkehr

VMT Verkehrsverbund Mittelthüringen

VRR Verkehrsverbund Rhein-Ruhr

K Kurzstrecke (Preisstufe)

Kapitel 1

Einführung

1.1 Motivation

Bei einem Besuch in meiner Heimatstadt wollte ich mit meinem Freund von Herne nach Bochum fahren. Dort wollten wir in der Innenstadt einen Freund treffen und danach gemeinsam nach Bochum-Wattenscheid fahren. Von dort aus wollten wir anschließend wieder über die Bochumer Innenstadt zurückfahren. Für die Fahrt von Herne nach Bochum und zurück hätten wir eine andere Preisstufe benötigt, als von Herne nach Bochum-Wattenscheid. Anstatt die kostengünstigste Kombination aus verschiedenen Fahrscheinen zu ermitteln, wählten wir das Auto. Um nicht für jede Verbindung die günstigste Ticketkonstellation manuell zu ermitteln, kam die Idee einer App auf, welche diese Aufgabe für den Nutzer übernimmt. Von dieser würden mehr Menschen profitieren als nur die, die sich mit den genauen Fahrscheinvorschriften und Preisen beschäftigt haben. Dadurch könnte die Anzahl der Personen, welche den öffentlichen Personennahverkehr (ÖPNV) nutzen, steigen und so die Umwelt entlastet werden.

1.2 Grundidee

Das Tarifsystem des Verkehrsverbundes Rhein-Ruhr (VRR) ist für einzelne Fahrten leicht verständlich. Reisen jedoch mehrere Personen oder werden mehrere Fahrten unternommen, so ist dies nicht mehr gegeben. In diesen Fällen sind die Preisgestaltung sowie die Regeln der Fahrscheine komplexer. Aus diesen Gründen soll mithilfe einer App die Suche nach einer geeigneten und kostengünstigen Fahrscheinkombination für den Nutzer vereinfacht werden.

Dabei soll der Nutzer im Voraus Fahrten planen, für welche dann die kostengünstigste Kombination an Fahrscheinen ermittelt wird. Dadurch soll die Beratung nicht mehr an die Öffnungszeiten von Kundenzentren oder Hotlines gebunden sein. Auch kann der Nutzer weitere Fahrten hinzufügen oder Fahrten entfernen, um zu sehen, inwieweit diese den Preis beeinflussen.

1.3 Ziel der Arbeit

Ziel dieser Arbeit ist der Entwurf eines Algorithmus zur Minimierung der Kosten von Fahrscheinen. Dabei sollen Fahrten in verschiedenen Städten und über Städtegrenzen hinweg im gesamten Verbundgebiet berücksichtigt werden. Auch sollen alle Preisstufen des exemplarisch betrachteten Verkehrsverbundes (VRR) berücksichtigt werden. Des Weiteren sollen die Vorschriften aller Tickets eingehalten werden. Dazu gehört, dass wenn ein Fahrschein einem Gültigkeitsbereich zugewiesen ist, alle zugehörigen Fahrten in diesem Bereich liegen müssen. Die möglichen Gültigkeitsbereiche sind vom Verkehrsverbund vorgegeben. Will der Nutzer mit weiteren Personen reisen, so wird sichergestellt, dass keine Person ohne gültigen Fahrschein fährt. Um rechtliche Konsequenzen zu vermeiden, soll in der Anwendung kein Ticketkauf möglich sein. Stattdessen sollen dem Nutzer die entsprechenden Fahrscheine angezeigt werden. Für den Kauf und das Entwerten der Fahrscheine ist der Nutzer selbst verantwortlich.
In der hier zu entwickelnden Anwendung soll der Nutzer die Möglichkeit haben, Fahrten zu planen. Auf Basis dieser Fahrten wird die Optimierung durchgeführt.
Die Bedienung der App soll intuitiv erfolgen, weshalb die Oberfläche der Anwendung vor allem übersichtlich und leicht verständlich sein soll. Dafür werden verschiedene, bereits vorhandene, Anwendungen zur Fahrplanauskunft betrachtet. Anhand dieser werden Merkmale gesammelt, welche in der App ebenfalls umgesetzt werden sollen. Für die Ticketberatung werden aus diesem Grund zwei bereits vorhandene Fahrscheinberater betrachtet, die Funktionen werden dabei auch unter dem Gesichtspunkt der notwendigen Funktionalität einer App sowie Fahrscheinberatung im Speziellen betrachtet. Zusätzlich soll die Anwendung leicht zu erweitern und anpassbar für andere Verkehrsverbünde sein.

1.4 Aufbau der Arbeit

In diesem Kapitel wurde die Idee, Motivation und das Ziel für diese Arbeit vorgestellt. Im folgenden Kapitel werden der hier betrachtete Verkehrsverbund inklusive dessen Fahrscheine erläutert. Des Weiteren werden bereits existierende Anwendungen für die Fahrplanauskunft und für die Fahrscheinberatung vorgestellt. Diese Anwendungen und Fahrscheine werden im darauf folgenden Kapitel analysiert. Basierend auf den Ergebnissen der Analyse wird in Kapitel 4 die App entworfen. Danach wird die Realisierung des Entwurfs vorgestellt. Dann wird die dadurch entstandene Anwendung evaluiert. Dabei wird überprüft, inwieweit die im Verlauf der vorherigen Analyse erstellten Anforderungen erfüllt werden. Außerdem wird die Befragung der Nutzer in Kapitel 6 ausgewertet. Abschließend wird die Arbeit resümiert und ein Ausblick auf mögliche Erweiterungen und Anpassungen der Optimierung sowie der App gegeben.

Kapitel 2

Grundlagen

In diesem Kapitel wird kurz der Verkehrsverbund Rhein-Ruhr vorgestellt, einschließlich dessen Tarifsystem und die angebotenen Fahrscheine. Danach werden verschiedene Anwendungen zur Fahrplanauskunft, welche in der Analyse betrachtet werden, eingeführt. Im Anschluss werden die Beratungsangebote des VRRs für den Ticketkauf sowie eine App zur Fahrscheinoptimierung präsentiert.

2.1 Verkehrsverbund Rhein Ruhr

Gegründet wurde der Verkehrsverbund Rhein-Ruhr, kurz VRR, 1980, um die Tarife von mehr als 20 kommunalen Verkehrsbetrieben zu vereinheitlichen [1]. Der Zuständigkeitsbereich des VRRs liegt im westlichen Teil von Nordrhein-Westfalen an der Grenze zu den Niederlanden [2, 3]. In Abbildung 2.1 sind die einzelnen Tarifgebiete des VRRs eingezeichnet. Durch die Farbe eines Gebiets ist die Zuordnung zur Region gekennzeichnet. Im Westen liegt der nördliche und zentrale Teil des Niederrheins [4]. In der Abbildung ist diese Region rot eingezeichnet. Das Ruhrgebiet fällt fast gänzlich in das Gebiet des VRRs [5–7]. In der Abbildung ist dies die blaue Fläche, welche die nördliche und östliche Grenze beinhaltet. Der Kreis Wesel, in der Abbildung lila eingefärbt, fällt sowohl in die Region Niederrhein als auch in das Ruhrgebiet [4, 8]. Das Bergische Land im Südosten wird nur im nördlichen Bereich abgedeckt [2, 9]. Die Region ist hier grün eingefärbt. Westlich davon liegt die Stadt Düsseldorf, hier gelb dargestellt. Im Süden liegt der Rhein-Kreis-Neuss [10]. In der Abbildung ist dieser Kreis rosa eingefärbt. Zwei weitere Städte im Zuständigkeitsbereich sind die Stadt Krefeld (türkis) im Nordwesten vom Rhein-Kreis-Neuss sowie die Stadt Mönchengladbach (orange) im Westen von diesem.

Das Gebiet hat eine Nord-Süd-Ausdehnung von etwa 65 km und eine West-Ost-Ausdehnung von rund 95 km. Insgesamt deckt der Verkehrsbund eine Fläche von 7305 km^2 ab. Auf dieser Fläche wohnten 2015 etwa 7,8 Millionen Menschen. Es handelt sich damit, nach eigenen Angaben, um den „einwohnerstärkste[n] Verkehrsverbund und [den] größte[n] Nahverkehrsballungsraum Europas“[11].
Zu den Aufgaben des VRRs gehört zum einen die Organisation, Planung und Ausgestaltung des Schienenpersonennahverkehrs (SPNV) in der Region. Zum anderen sollen Maßnahmen koordiniert werden, deren Ziel ein attraktiver, marktwirtschaft-

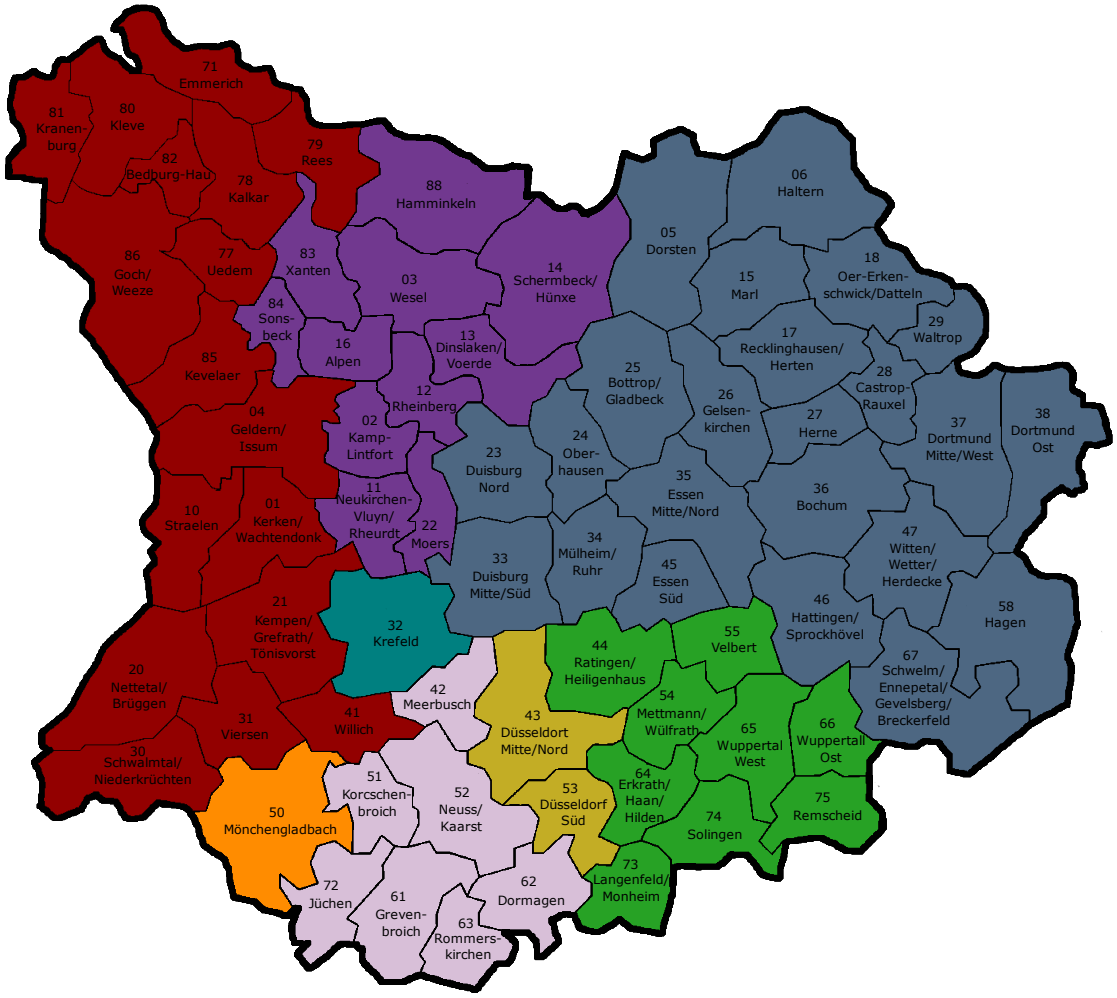


Abbildung 2.1: Der Verbundraum mit den einzelnen Tarifgebieten [12], eingefärbt für die jeweilige Region. Rot: Niederrhein [4]; Blau: Ruhrgebiet [5–7]; Lila: Niederrhein und Ruhrgebiet [4, 8]; Grün: Bergisches Land [2, 9]; Gelb: Stadt Düsseldorf; Orange: Stadt Mönchengladbach; Rosa: Rhein-Kreis Neuss [10].

licher ÖPNV ist. Dies wird beispielsweise durch einen einheitlichen Tarif erreicht [1, 11], welcher im folgenden Abschnitt vorgestellt wird.

2.1.1 Tarifsystem

Das Gebiet des VRRs ist in einzelne Waben unterteilt. Aus mehreren Waben setzt sich ein Tarifgebiet zusammen. Ein Tarifgebiet besteht meist aus einer Stadt, mehreren kleinen Städten oder Gemeinden. Ein Beispiel ist in Abbildung 2.2 zu sehen. In Rot sind die Wabengrenzen innerhalb der Tarifgebiete eingezeichnet. Die Grenzen zwischen den einzelnen Tarifgebieten sind in Schwarz eingezeichnet. Dabei gilt für die Waben (w_i) eines Tarifgebiets (t):

$$t = w_i : 10,$$

wie in Abbildung 2.2 zu sehen ist.
Die Informationen zu dem Tarifsystem stammen aus der Broschüre [13].

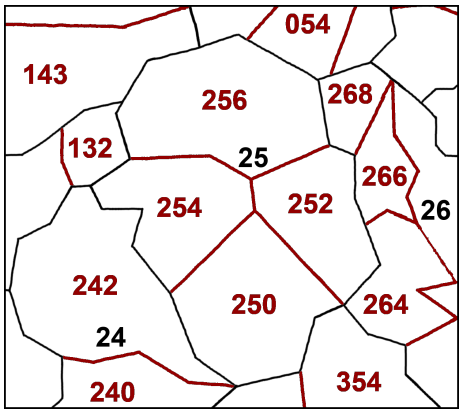


Abbildung 2.2: Ausschnitt des Wabenplans [14]. In Rot sind die Wabengrenzen und in Schwarz die Tarifgebietgrenzen eingezeichnet.

Im VRR Tarifsystem wird für jede Verbindung eine Preisstufe festgelegt. Es gibt die fünf Preisstufen Kurzstrecke (K), A, B, C und D. Für Kinder im Alter zwischen sechs und 15 Jahren gilt jeweils ein ermäßigter Fahrpreis. Kinder, die jünger als sechs Jahre sind, fahren unentgeltlich mit.
Mit einem Kurzstreckenticket können keine Angebote des SPNV, wie beispielsweise Regionalbahnen und S-Bahnen, genutzt werden. Auch ist ein Umstieg bei dieser Preisstufe nicht erlaubt. In der Regel hat das Ticket eine Gültigkeit von drei Haltestellen, was im Schnitt einer Strecke von 1,5 km entspricht.
Mit einem Ticket der Preisstufe A sind zum einen Fahrten innerhalb eines Tarifgebiets möglich. Zum anderen gilt diese Preisstufe für Fahrten zwischen zwei benachbarten Waben. In den fünf Städten Dortmund, Duisburg, Düsseldorf, Essen und Wuppertal, den Städten, die aus zwei Tarifgebieten bestehen, kann mit einem Fahrschein dieser Preisstufe der Nahverkehr im gesamten Stadtgebiet genutzt werden.

Bei den Einzelfahrscheinen, dem *7-TageTicket*, sowie dem *Ticket1000* und *Ticket2000* wird die Preisstufe A jeweils weiter unterteilt, in die drei Kategorien A1, A2 und A3. Die Preisstufe A1 gilt für kleine Städte oder Gemeinden. In Städten mit einem gut ausgebauten öffentlichen Nahverkehr gilt die nächsthöhere Stufe. Die Stufe A3 gilt in den fünf Städten Bochum, Dortmund, Düsseldorf, Essen und Wuppertal. In diesen Städten ist der Nahverkehr besonders gut ausgebaut und die Taktung der Linien sehr eng. Bei Fahrten zwischen zwei benachbarten Tarifgebieten, die jedoch unterschiedlichen Preisstufen zugeordnet sind, muss das Ticket in der höheren Stufe erworben werden.
Für die Zeitfahrscheine gilt, dass für die Preisstufe B beim Entwerten ein Zentralgebiet festgelegt wird. Für jedes dieser Gebiete ist ein Geltungsbereich festgelegt, in welchem der Fahrschein gültig ist. Die Einzelfahrscheine sind jeweils in dem Tarifgebiet gültig, in dem das Ticket entwertet wurde und meistens für einen Umkreis von zwei Tarifgebieten.
Für die Preisstufe C gibt es 19 festgelegte Regionen im Tarifgebiet des VRRs. Mit dem Entwerten des Tickets legt der Kunde das Gebiet fest.
Mit einem Ticket der Preisstufe D ist das Nutzen des Nahverkehrs im gesamten VRR

Gebiet möglich sowie in angrenzenden Bereichen von anderen Verkehrsverbünden. Für diese Gebiete gelten jedoch spezielle Regeln, die im Rahmen dieser Arbeit nicht betrachtet werden.

2.1.2 Fahrscheine

Der Verkehrsverbund Rhein-Ruhr bietet zum einen Tickets im Abonnement-System, zum anderen auch Tickets ohne monatliches Abonnement an. In dieser Arbeit werden nur die zuletzt genannten Fahrscheine betrachtet, denn Abonnements sind zum Teil nur für bestimmte Nutzerkreise, beispielsweise Schüler, verfügbar [15]. Außerdem wird die Mitnahme von Fahrrädern nicht berücksichtigt. Außerdem soll die Zielgruppe der Anwendung vor allem Nutzer sein, die selten den ÖPNV nutzen und somit kein Abonnement abgeschlossen haben. Die Informationen zu den einzelnen Fahrscheinen stammen, soweit nicht anders gekennzeichnet aus der Broschüre [13] des VRRs.

Einzelfahrscheine

Als Einzelfahrscheine werden Fahrscheine bezeichnet, die jeweils nur eine feste Anzahl von Fahrten erlauben. Im Verkehrsverbund gibt es drei verschiedene Einzelfahrscheine, welche in Tabelle 2.1 zu sehen sind. Mit diesen Fahrscheinen sind keine Rundfahrten erlaubt, sondern nur Fahrten in eine Richtung. Während einer Fahrt darf jedoch beliebig oft umgestiegen werden. Ausgenommen ist, wie im vorherigen Abschnitt erläutert, die Preisstufe Kurzstrecke. Ein Fahrschein kann für mehrere Personen genutzt werden, muss dann jedoch entsprechend häufig entwertet werden. Wie der Tabelle entnommen werden kann, wird kein 10erTicket für Kinder angeboten. Auch kann dieser Fahrschein im Gegensatz zu den anderen beiden Fahrscheinen nur online erworben werden.

Name	Anzahl Fahrten	Erwachsene	Kinder	Nur Online verfügbar
Einzelticket	1	✓	✓	✗
4erTicket	4	✓	✓	✗
10erTicket	10	✓	✗	✓

Tabelle 2.1: Einzelfahrscheine des VRRs [13].

Zeitfahrscheine

Zeitfahrscheine erlauben dem Nutzer beliebig viele Fahrten in einem festen Zeitintervall. Im Gegensatz zu Einzelfahrscheinen sind Rund- und Rückfahrten mit dem jeweiligen Fahrschein erlaubt [15, 16]. Die Fahrscheine sind nur für eine gleichbleibende Person gültig. Die sechs Zeitfahrscheine des VRRs, welche in Tabelle 2.2 aufgelistet sind, sind nur für Erwachsene zu kaufen, können jedoch auch von Kindern genutzt werden. Es gibt jedoch keine vergünstigten Fahrscheine für Kinder. Im Gegensatz zu den Einzelfahrscheinen sind nicht alle Fahrscheine für alle Preisstufen

verfügbar.
Das 4-StundenTicket ist werktags erst ab 9 Uhr gültig, am Wochenende sowie an

Name	Geltungsdauer	Preisstufen	Nur Online Verfügbar
4-StundenTicket	4 h	A1, A2	✗
HappyHourTicket	18^{00} bis 6^{00}	A	✓
24-StundenTicket	24 h	A bis D	✗
48-StundenTicket	48 h	A bis D	✗
7-TageTicket	7 d	A bis D	✗
30-TageTicket	30 d	A bis D	✓

Tabelle 2.2: Zeitfahrscheine des VRRs [13].

Feiertagen hingegen den ganzen Tag lang. Wird das Ticket erst nach 20 Uhr entwertet, so ist das Ticket zusätzlich am nächsten Tag noch gültig, jedoch maximal bis 3 Uhr. Dieser Fahrschein ist nur innerhalb eines Tarifgebiets gültig. Eine Ausnahme davon ist die Stadt Duisburg. Diese besteht aus zwei Tarifgebieten, in welchen das Ticket gültig ist.
In der Tabelle 2.2 ist für das HappyHourTicket die Geltungsdauer von 18 Uhr bis 6 Uhr angegeben, damit ist gemeint, dass die Entwertung erst ab 18 Uhr möglich ist, das Ticket jedoch unabhängig von der Startzeit bis zum Folgetag um 6 Uhr gültig ist. Im Gegensatz zu dem vorherigen Ticket kann der Nutzer für dieses festlegen, ob der Fahrschein in einem Tarifgebiet oder in zwei benachbarten Waben gültig ist. Dies trifft auch für die weiteren Fahrscheine in der Preisstufe A zu.
Im Gegensatz zu den anderen Tickets gibt es die Option, das 24- und 48-StundenTicket nicht nur für eine Person zu kaufen, sondern für bis zu fünf Personen.
Von dem 30-TageTicket gibt es zwei verschiedene Ausführungen, zum einen das Ticket1000, zum anderen das Ticket2000. Diese beiden Fahrscheine unterscheiden sich in den weiteren Leistungen, die diese Fahrscheine enthalten. Das Ticket1000 erlaubt es dem Nutzer, an Wochenenden, Feiertagen sowie werktags nach 19 Uhr bis zu drei Kinder unentgeltlich mitzunehmen. Bei dem Ticket2000 kann zusätzlich neben den drei Kindern noch eine weitere erwachsene Person mitfahren. Des Weiteren ist Letzteres zu den genannten Zeitpunkten im gesamten Tarifgebiet gültig und erlaubt es unentgeltlich das Fahrrad zu transportieren. Beide Fahrscheine lassen sich ebenso vergünstigt erwerben, sind dann jedoch werktags erst ab 9 Uhr gültig und im Abonnement erhältlich.
Mit Ausnahme des HappyHourTickets gilt für die Fahrscheine der Tabelle 2.2, dass die Geltungsdauer ab dem Zeitpunkt der Entwertung beginnt.

2.2 Vorhandene Anwendungen

Im Folgenden werden zunächst die Beratungsangebote des VRRs und anschließend eine externe Anwendung vorgestellt.

Im Rahmen dieser Arbeit wurden verschiedene Anwendungen betrachtet, jedoch wird vor allem auf zwei Anwendungen eingegangen. Bei den anderen Anwendungen werden hingegen nur Besonderheiten herausgestellt.
Die Anwendung *VRR App – Fahrplanauskunft* des VRRs wurde am 04.04.2011 veröffentlicht und bis heute über eine Million Mal heruntergeladen [17]. Die Funktionen sind die individuelle Fahrplanauskunft zwischen zwei Haltestellen zu einer gewünschten Abfahrts- oder Ankunftszeit, die Auskunft über die nächsten Abfahrten an einer gewählten Haltestelle, der Kauf von Tickets sowie eine Karte mit den nächsten eingezeichneten Haltestellen. Weitere Funktionen sind die Information über Verspätungen und die Netzpläne. Die Anwendung deckt die in Abschnitt 2.1 vorgestellte Region ab, des Weiteren werden Informationen für Fahrten in angrenzende Verkehrsverbünde zur Verfügung gestellt [17].
Eine Anwendung, die individuelle Fahrplanauskünfte liefert, einen integrierten Abfahrtsmonitor hat und über Verspätungen informiert, ist *Öffi– Farhplanauskunft* [18]. Weitere Funktionen sind die integrierten Netzpläne und eine integrierte Karte. Diese Funktionen sind in über 54 Regionen weltweit verfügbar, darunter auch die Region des hier betrachteten Verkehrsverbundes. Seit 2010 wurde die Anwendung über fünf Millionen Mal heruntergeladen. Entwickelt wurde Öffi von Andreas Schildbach [18].
Weitere betrachtete Anwendungen sind:

- *Mutti* – VRR – [19]
- *MyHannover* – Hannover – [20]
- *Jelbi* – Berlin – [21]
- *DB Navigator* – [22]
- *Moovit* – Über 3000 Städte weltweit – [23]
- *Citymapper* – 58 Städte und Regionen weltweit – [24, 25]

2.3 Vorhandene Ticketberater

2.3.1 Angebote des VRRs

Neben der klassischen Beratung in Ticketzentren bietet der Verkehrsverbund Rhein-Ruhr eine Telefonhotline an, welche der Kunde kontaktieren kann. Eine weitere Möglichkeit, sich beraten zu lassen, ist der *VRR-Ticketberater* [26]. Dieser bietet für den Nutzer die Beratungsmöglichkeiten *Ab-und-zu-Fahrer* sowie verschiedene Informationsangebote für *Vielfahrer* an.

Die Beratung für *Ab-und-zu-Fahrer* beinhaltet eine einfache Abfrage von Start und Ziel und die Anzahl der reisenden Personen. Auf Wunsch kann die Rückfahrt ebenfalls berücksichtigt werden.

Die Beratung für *Vielfahrer* unterteilt sich in die Beratung für Kunden, die regelmäßig die gleichen Strecken fahren oder häufiger in einer Region unterwegs sind, dabei jedoch nicht auf einzelne Verbindungen festgelegt sein wollen. Des Weiteren

werden die vier Abonnementtickets, welche an Bedingungen geknüpft sind, vorgestellt: das *Bärenticket* für Kunden ab 60, das *Schokoticket* für Schüler, das *YoungTicketPlus* für Auszubildende und das *SozialTicket* für Anspruchsberechtigte [26].

2.3.2 FAIRTIQ

FAIRTIQ ist eine App, welche automatisch die kostengünstigsten Fahrscheine ermittelt. Die Anwendung wird in der Schweiz entwickelt und ist in Deutschland nur in einzelnen Regionen verfügbar, der VRR gehört nicht dazu. In der Schweiz und in Liechtenstein ist hingegen der gesamte ÖPNV abgedeckt [27]. Nach eigenen Angaben wurden bereits mehr als 15 Millionen Fahrten mit der App zurückgelegt [27]. Seit der Veröffentlichung am 27.04.2016 wurde die Anwendung bereits mehr als 100 000 Mal heruntergeladen [28]. In dieser Anwendung muss der Nutzer nicht mehrere Fahrten planen, sondern die App ermittelt die gefahrene Strecke mittels Standortdaten und Reisedauer automatisch. Aus diesen Daten wird dann am Ende des Tages der Fahrpreis berechnet.

Kapitel 3

Analyse

Damit die Anwendung intuitiv zu bedienen ist, gleicherweise die Oberfläche übersichtlich und leicht verständlich, werden in diesem Kapitel zunächst verschiedene Anwendungen zur Fahrplanauskunft betrachtet. Neben dem Layout werden auch die Einstellungsmöglichkeiten und Funktionsweisen verglichen. Dabei werden Anforderungen an die Anwendung, in welche die Optimierung eingebettet wird, erarbeitet. Anschließend werden die Fahrscheinberatungsangebote untersucht. In diesem Schritt wird auf die Funktionsweise dieser eingegangen sowie ebenfalls Schlussfolgerungen für die eigene Beratung gezogen. Danach werden die Fahrscheine des VRRs betrachtet und es werden Regeln für die Optimierung aufgestellt. Dann werden die möglichen Optimierungsansätze vorgestellt. Zum Schluss werden die Anforderungen an die eigene Anwendung zusammengefasst.

3.1 Vergleich bestehender Fahrtenplaner für den ÖPNV

Zum Vergleich der einzelnen Anwendungen werden die vier Symbole ✓, ∼, ✗ und ❋ genutzt. Dabei bedeutet ✓, dass die jeweilige Anwendung das Kriterium oder Merkmal vollständig erfüllt. Erhält eine Anwendung eine ∼, so wird das Kriterium nur teilweise erfüllt. Wird ein Merkmal hingegen gar nicht erfüllt, erhält die Anwendung ein ✗. Mit ❋ sind Unklarheiten gekennzeichnet, sowie Bemerkungen, welche unter der jeweiligen Tabelle erläutert werden. Detailliert betrachtet werden nur die Anwendungen VRR App und Öffi, da es sich bei Ersterer um die App des exemplarisch betrachteten Verkehrsverbundes handelt. Letztere wurde als Basis für die Anwendung, welche im Rahmen dieser Arbeit entwickelt wird, gewählt, da es sich um eine Open Source Anwendung handelt [29]. Von den anderen betrachteten Anwendungen werden nur die Besonderheiten hervorgehoben. Die tabellarischen Vergleiche aller Anwendungen sind in Anhang A.1 zu finden.

3.1.1 Routenoptionen

In den ersten beiden Kategorien der Tabelle 3.1 (Tabelle A.1) werden die Anwendungen verglichen, inwieweit diese dem Nutzer erlauben, Einstellungen bezüglich der Barrierefreiheit von Routen oder des Gehtempos zu wählen. In der Kategorie

Anwendung	Kriterien		
	Barrierefreiheit	Gehtempo	Verkehrsmittel
VRR App	✓	✓	ÖPNV: ✓; Fahrrad: ✓; Sharing-Angebote: ✗
Öffi	∼	✓	ÖPNV: ✓; Fahrrad: ✗; Sharing-Angebote: ✗

Tabelle 3.1: Vergleich der vorgestellten Anwendungen bezüglich ihrer Optionen für die Routenoptimierung und berücksichtigte Verkehrsmittel.

Verkehrsmittel wird verglichen, ob die Anwendungen den klassischen ÖPNV, das eigene Fahrrad des Nutzers und verschiedene Sharing-Angebote bei der Verbindungsfindung berücksichtigen.
Bei den meisten Anwendungen kann der Nutzer keine Einstellungen bezüglich seiner eigenen Mobilität treffen. Besonders viele Möglichkeiten, das Reiseprofil anzupassen, hat der Nutzer in der VRR App. Öffi erhält die mittlere Wertung, da die App nur die drei Kriterien *keine Einschränkung, bedingt barrierefrei* sowie *barrierefrei* anbietet. Zu diesen Kategorien erhält der Nutzer jedoch keine Erläuterung.
Wie in Tabelle A.1 zu sehen ist, sind die beiden Anwendungen, neben Mutti, die einzigen, die dem Nutzer die Möglichkeit geben, das Gehtempo zu wählen. Für diese gilt, dass der Nutzer die drei Geschwindigkeiten *langsam, normal* und *schnell* zur Wahl hat.
Die Anwendung VRR App bietet dem Nutzer außerdem die Möglichkeit, für berücksichtigte Alternativhaltestellen eine maximale Fußwegzeit anzugeben. Der DB Navigator erlaubt es dem Nutzer, eine minimale Umstiegszeit einzustellen. In der Anwendung MyHannover kann die Anzahl der Umstiege begrenzt werden.
Für die Verkehrsmittel gilt, dass alle Anwendungen den öffentlichen Personennahverkehr berücksichtigen. In allen mobilen Anwendungen, außer in Jelbi und Citymapper, hat der Nutzer die Möglichkeit, die berücksichtigten Verkehrsmittel zu filtern. Das eigene Fahrrad als Transportmittel berücksichtigen vier Anwendungen, wobei nur Mutti dem Nutzer ermöglicht, die Route auf zwei verschiedene Arten zu optimieren. Sharing-Angebote können nur bei den beiden Drittanbieter Anwendungen Moovit und Citymapper und Jelbi verwendet werden. Mutti nutzt als Sharing-Angebote nur die Fahrräder von *Metropolrad* und erhält deshalb eine mittlere Wertung.

3.1.2 Übersichtlichkeit

Bei der Übersichtlichkeit von Anwendungen werden diese in drei verschiedenen Ansichten verglichen:

1. Angaben zur Fahrt machen

2. Übersicht über mögliche Verbindungen

(a) Ansicht

(b) Einzelne Fahrt

3. Detailansicht einer einzelnen Fahrt

Planung von Fahrten

Für alle hier betrachteten Anwendungen wurde festgestellt, dass die Bereiche zum Tippen und die Entfernung zwischen den einzelnen Buttons groß genug sind, um das *FatFinger Problem*[1] zu vermeiden. Die Anwendungen wurden anhand der Kriterien

Anwendung	Kriterien bei der Planung von Fahrten			
	Auswahl von Haltestellen	Wahl des Zeitpunkts	Zwischen-halte	Komfort-Funktionen
VRR App	✓	✓	✗	✓
Öffi	∼	✓	✓	✓

Tabelle 3.2: Vergleich der Anwendungen, wie übersichtlich die Planung einer Fahrt ist.

aus der Tabelle 3.2 verglichen. Die gleichen Kriterien werden für alle betrachteten Anwendungen in Tabelle A.2 in Anhang A.1 behandelt. In der Kategorie „Auswahl von Haltestellen“ werden die Anwendungen darauf hin untersucht, ob die jeweilige Anwendung alte Suchanfragen berücksichtigt, zusätzlich werden favorisierte Haltestellen anzeigt. Einige Anwendungen bieten noch weitere Funktionalitäten bei der Auswahl an, dies führt jedoch nicht zu einer besseren Wertung. Das Merkmal „Wahl des Zeitpunktes“ vergleicht für die Anwendungen, ob der Nutzer einen Zeitpunkt direkt auswählen kann. Bei der Kategorie „Zwischenhalte“ wird überprüft, ob der Nutzer Zwischenhalte hinzufügen kann. Beispiele für die Kategorie „Komfortfunktionen“ werden am Ende des Abschnitts vorgestellt.
Bei der Auswahl von Haltestellen lassen sich die Anwendungen in zwei Kategorien unterteilen: Einerseits diejenigen, welche in der Startansicht zwei Felder haben, andererseits Anwendungen, die in der Startansicht nur ein Feld haben. Beide hier betrachteten Anwendungen fallen in die erste Kategorie. Anwendungen, die in die zweite Kategorie fallen, nutzen als Startpunkt standardmäßig den aktuellen Standort des Nutzers.

Öffi ist die einzige hier betrachtete App, welche keine extra Ansicht für die Wahl eines Ortes oder die Eingabe der Zeit öffnet. Auch berücksichtigt Öffi keine Favoriten oder vorherigen Anfragen, weshalb diese Anwendung im Gegensatz zu den anderen Anwendungen nur eine mittlere Bewertung erhält, siehe Tabelle A.2. Besonders die Anwendungen Mutti und MyHannover fallen durch zusätzliche Funktionen zur Auswahl von Orten positiv auf.

[1] „Auf Touch-Geräten erschwert die große Kontaktfläche des Fingers es, kleine Bereiche zu berühren, weil für den Nutzer unklar ist, wo die genaue Courser-Position in diesem Bereich liegt“[30].

Anwendung	Kriterien bei der Auswahl einer Fahrt			
	grafische Ansicht	listen Ansicht	unterschiedliche Farben für einzelne Linien	früher / später
VRR App	✓	✓	∼	✓
Öffi	✓	✗	∼	✓

(a) Vergleich der Anwendungen bezüglich der Darstellung von möglichen Verbindungen.

Anwendung	Kriterien bei der Auswahl einer Fahrt					
	Anzahl Umstiege	Startzeit	Endzeit	Dauer	Preisstufe	Umstiegs-zeiten
VRR App	∼	✓	✓	✓	∼	∼
Öffi	∼	✓	✓	✗	∼	∼

(b) Vergleich der Anwendungen in der Darstellung einer möglichen Verbindung.

Tabelle 3.3: Vergleich der Anwendungen in der Übersicht von möglichen Verbindungen.

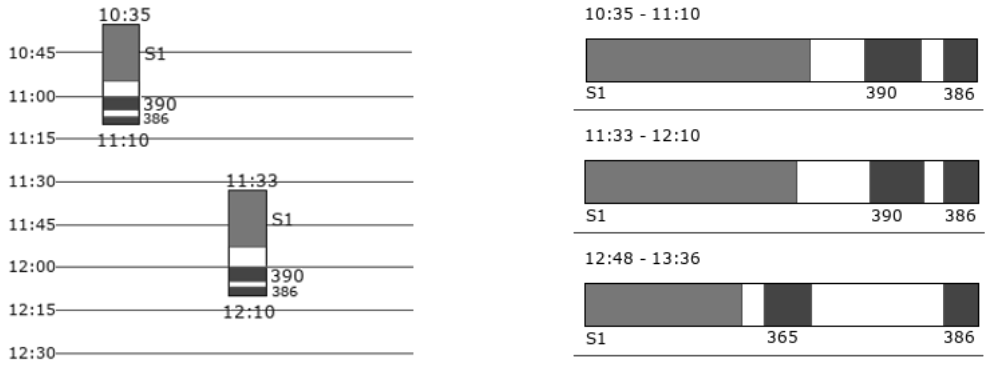
Für die Auswahl der Uhrzeit und des Datums erhalten alle Anwendungen die höchste Bewertung, bei denen dies im ersten Schritt einer Planung möglich ist. Das trifft auf die beiden hier genauer betrachteten Anwendungen zu. Ist die Wahl hingegen erst im nächsten Schritt oder nach der Ermittlung von Fahrten möglich, erhält die jeweilige Anwendung eine mittlere Wertung, vergleiche Tabelle A.2.
Die VRR App ermöglicht es dem Nutzer nicht, im Gegensatz zu Öffi, einen Zwischenhalt anzugeben. Die Anwendungen MyHannover und DB Navigator bieten dem Nutzer sogar an, zwei Zwischenhalte hinzuzufügen. In Mutti kann der Nutzer zwar nur einen Zwischenhalt angeben, für diesen kann er jedoch eine Aufenthaltszeit aus einer vorgegebenen Liste auswählen.
Die Komfortfunktionen der Anwendung VRR App sind besonders umfangreich: So kann der Nutzer beispielsweise mit einem Klicken die Anfrage löschen beziehungsweise abbrechen, Start und Ziel vertauschen und auf die Einstellungen zugreifen. Das Tauschen von Start und Ziel ist ebenfalls in Öffi und anderen Anwendungen möglich. Eine Komfortfunktion von Moovit ist, dass die letzte mögliche Verbindung von einem Tag mit nur einem Klick angezeigt werden kann. Jelbi besitzt diverse Komfortfunktionen für Sharing-Angebote, die hier jedoch nicht weiter vorgestellt werden. Der DB Navigator ist die einzige hier betrachtete Anwendung, bei welcher die Anzahl der Reisenden eingestellt werden kann, dabei handelt es sich um eine optionale Angabe.

Auswahl einer Fahrt

Für die Auswahl einer Fahrt wurden insgesamt zehn Kriterien verglichen.

Die ersten vier Kriterien in Tabelle 3.3a, beziehungsweise Tabelle A.3 im Anhang, beschäftigen sich vor allem mit der Übersicht über die möglichen Fahrten.

Dafür wurde untersucht, welche Anwendungen eine grafische Ansicht und welche eine Ansicht in Listenform besitzen. Beispiele für diese Darstellungen sind in Abbildung 3.1 zu sehen. In der grafischen Ansicht ist der Verlauf einer einzelnen Fahrt vertikal. Die einzelnen Fahrten sind horizontal nebeneinander angeordnet. In der Listenansicht sind hingegen die einzelnen Fahrten untereinander angeordnet. In dieser ist der Verlauf einer einzelnen Fahrt horizontal angeordnet.
Ein Merkmal war die Einfärbung von einzelnen Linien in der Übersicht einer Fahrt



(a) Grafische Ansicht **(b)** Listenansicht

Abbildung 3.1: Beispiel für listen Ansicht und grafische Ansicht, basierend auf den Ansichten der Anwendung Mutti [19].

zur einfacheren Unterscheidung. Das letzte geprüfte Kriterium war, ob der Nutzer leicht auf frühere oder spätere Verbindungen zugreifen kann.
Die Anwendungen VRR App und Mutti sind die einzigen hier betrachteten Anwendungen, in denen der Nutzer zwischen grafischer Ansicht und Listenansicht wechseln kann. Bis auf Öffi nutzen alle anderen Anwendungen nur die Listenansicht, vergleiche Tabelle A.3.
Die beiden hier betrachteten Anwendungen erhalten für die Färbung der Linien die mittlere Bewertung, da unterschiedliche Fahrzeugarten verschiedene Farben besitzen.
In Öffi kann der Nutzer mittels links- beziehungsweise rechts-Swipe frühere oder spätere Verbindungen abfragen. Während dies in der VRR App über zwei Buttons umgesetzt ist.
Die restlichen sechs Kriterien in Tabelle 3.3b, beschäftigen sich damit, inwieweit für eine einzelne Fahrt in der Übersicht die Merkmale: Anzahl der Umstiege, Startzeit, Endzeit, Dauer, Preisstufe und Umstiegszeiten (oder Wartezeiten) für jede einzelne mögliche Verbindung erkennbar sind. Die Übersicht über alle Anwendungen ist in Tabelle A.4 auf Seite 93 zu sehen.
Bei allen Anwendungen kann der Nutzer die Anzahl der Umstiege durch Abzählen selbst ermitteln, dies führt zu einer mittleren Bewertung. Die Anwendungen MyHannover und DB Navigator geben die Anzahl an Umstiegen direkt an und erhalten somit eine bessere Bewertung.
Sowohl in Öffi als auch in der VRR App kann der Nutzer die Startzeit direkt ablesen. Dies ist nicht in allen Anwendungen so umgesetzt. In Jelbi beispielsweise wird die Startzeit darüber angegeben, in wie viel Minuten die Fahrt beginnt.

Anwendung	Kriterien für die Detailansicht einer Fahrt				
	Übersicht	Zwischen-halte	Verspätungen / Probleme	Navi-gation	Zeiten einer Linie
VRR App	∼	✓	✓	✓	✓
Öffi	✓	✓	✓	✗	✓

Tabelle 3.4: Vergleich der Anwendungen bezüglich der Detailansicht einer Fahrt.

In den beiden hier betrachteten Anwendungen ist die Endzeit jeder Verbindung, wie bereits die Startzeit, direkt erkennbar.
Die Dauer der einzelnen Verbindung ist nur in Öffi nicht angegeben, sondern muss in dieser Anwendung vom Nutzer selbst ermittelt werden.
Keine der hier betrachteten Anwendungen teilt dem Kunden direkt die Preisstufe für die Verbindung mit. Je nach Tarifsystem ist dies aber auch nicht nötig. Bei mehreren Anwendungen, unter anderen die beiden detailliert betrachteten, werden dem Nutzer die Kosten für ein Einzelticket angezeigt, aus diesen lässt sich die Preisstufe ableiten.
Die Umstiegszeiten für eine Verbindung ließen sich am besten in der Anwendung Jelbi entnehmen. In dieser werden Umstiege und Wartezeiten mit dem entsprechenden Piktogramm und der Minutenanzahl gekennzeichnet. In den Anwendungen VRR App, Mutti, MyHannover und Öffi ließen sich diese Zeiten durch freie Bereiche erkennen, eine genaue Zeit lässt sich aus diesen Bereichen jedoch nicht ablesen. Die restlichen Anwendungen informieren den Nutzer in dieser Ansicht hingegen gar nicht über diese Zeiten, vergleiche Tabelle A.4.

Detailansicht einer einzelnen Fahrt

Die Auswertung der Kriterien, die eine Detailansicht erfüllen soll, sind in Tabelle 3.4, beziehungsweise Tabelle A.5 für alle betrachteten Anwendungen, zu sehen. In der Übersicht der Ansicht sollten die folgenden Punkte direkt ablesbar sein:

- Start- und Zielpunkt der Fahrt
- Abfahrts- und Ankunftszeit
- Dauer
- Preisstufe
- Anzahl der Umstiege

Auch soll die Abfolge der einzelnen Verbindungen mit den jeweiligen Zwischenhalten in der gesamten Ansicht ersichtlich sein. Des Weiteren sollte der Nutzer in dieser Ansicht über Verspätungen und Störungen der jeweiligen Linien informiert werden. Ein weiteres Kriterium ist die Navigation zwischen zwei Punkten, bei der die Strecke nicht mit dem ÖPNV zurückgelegt wird. Das letzte Kriterium, welches

geprüft wird, ist, ob für die einzelnen Verbindungsabschnitte die Abfahrts- und Ankunftszeit angezeigt wird.
Wie in beiden Tabellen zu sehen ist, erhält die VRR App, im Gegensatz zu Öffi, für die Übersicht über eine einzelne Fahrt nur die mittlere Bewertung, da dem Nutzer nicht der Start- und Zielpunkt angezeigt werden. In beiden Anwendungen kann sich der Nutzer die Zwischenhalte einzelner Abschnitte anzeigen lassen. Beide informieren den Nutzer über Verspätungen und Probleme, wobei die Auskunft der VRR App ausführlicher ist.
Für die Navigation zwischen zwei Punkten öffnet Öffi *GoogleMaps*, während in der VRR App die einzelnen Navigationsschritte angezeigt werden. Ebenso öffnen andere Anwendungen GoogleMaps, weitere haben hingegen eine Kartenansicht eingebunden.
In allen Anwendungen lässt sich die Strecke auf der Karte anzeigen, jedoch ist dies unterschiedlich realisiert. Im DB Navigator wird zum Beispiel eine externe App geöffnet.
Das Teilen und Speichern von Verbindungen ist im Großteil der Anwendungen möglich, auch in den beiden hier betrachteten Anwendungen.
Eine Besonderheit der VRR App ist das Anpassen von Fahrten, indem der Nutzer einzelne Abschnitte der Fahrt anpasst. Dies ist auch in den Anwendungen Moovit und Citymapper möglich und in diesen für den Nutzer einfacher umgesetzt. In den Anwendungen Mutti und MyHannover kann der Nutzer zu der vorherigen oder nachfolgenden Fahrt mittels swipe nach links, beziehungsweise rechts, wechseln. In mehreren der betrachteten Anwendungen, unter anderem in der VRR App, hat der Nutzer die Möglichkeit, sich eine detaillierte Schritt-für-Schritt-Anweisung anzeigen zu lassen. Dabei wird jeweils nur ein Schritt auf dem Display angezeigt. Mittels swipe lassen sich die weiteren Schritte anzeigen. Meist ist der entsprechende Abschnitt im oberen Bereich auf einer Karte eingezeichnet.

3.1.3 Konsequenzen für die Anwendung

In der Anwendung, die im Rahmen dieser Arbeit entwickelt wird, soll der Nutzer für die Barrierefreiheit, wenn möglich, die gleichen Einstellungsmöglichkeiten wie in der VRR App haben. Falls umsetzbar, wird der Nutzer das Gehtempo wählen können. An Verkehrsmitteln ist zunächst geplant, nur den ÖPNV in der Anwendung zu berücksichtigen. Dabei soll der Nutzer die berücksichtigten Verkehrsmittel wählen können. Die Sharing-Angebote werden ausgeschlossen, da hier nur der ÖPNV betrachtet wird.

Für das Füllen des Formulars mit den Angaben zu einer Fahrt wird für die Auswahl der Haltestellen eine Extraansicht angestrebt. Falls realisierbar, werden in dieser Favoriten und alte Suchanfragen direkt angezeigt. Des Weiteren muss der Zeitpunkt direkt zu Beginn der Planung festgelegt werden. Eine weitere geplante Funktion ist das Hinzufügen eines Zwischenhaltes. Jedoch soll dabei keine Aufenthaltszeit eingeplant werden können. Während in der Anwendung DB Navigator die Angabe der Anzahl an Reisenden optional ist, vergleiche Abschnitt 3.1.2 Absatz „Planung von Fahrten“, wird diese Angabe in der App verpflichtend sein. Da die Anzahl der Personen eine wesentliche Komponente bei der Optimierung einnimmt.

Geplant ist, von den untersuchten Anwendungen verschiedene Komfortfunktionen zu übernehmen:

- Möglichkeit zum einfachen Vertauschen von Start und Ziel
- Der Zugriff auf die Routenoptionen während der Planung, wie in der VRR App
- Das direkte Abbrechen einer Anfrage
- Die letzte Verbindung eines Tages anzeigen lassen, analog zu Moovit

Über der Anzeige der möglichen Verbindungen ist vorgesehen, die vorherigen Eingaben des Nutzers (Start, Ziel, Datum, Uhrzeit) anzuzeigen. Die beabsichtigte Darstellung der möglichen Fahrten ist die Listendarstellung, wobei jede Linie anders eingefärbt wird. Mithilfe von Buttons soll der Nutzer sich frühere beziehungsweise spätere Verbindungen anzeigen lassen. Des Weiteren ist beabsichtigt, dass die Anwendung alle Kriterien der Tabelle 3.3b erfüllt. Der Nutzer wird also die folgenden Merkmale einer Verbindung direkt ablesen können:

- Abfahrtszeit
- Ankunftszeit
- Dauer
- Anzahl der Umstiege
- Preisstufe

In der Darstellung der Abfolge der Verkehrsmittel einer Fahrt sollen die jeweiligen Warte- beziehungsweise Umstiegszeiten direkt ablesbar sein.
Die gleichen fünf Merkmale einer Fahrt werden auch in der Detailansicht einer einzelnen Fahrt angestrebt. Zusätzlich ist geplant, Start-, Zielort und die Zwischenhalte von den einzelnen Abschnitten anzuzeigen. Für jeden einzelnen Abschnitt werden dabei die Abfahrtzeit sowie der Abfahrtsort angegeben. Außerdem hat der Nutzer die Möglichkeit, sich über Verspätungen zu informieren. Eine Navigationsbeschreibung wird, wenn möglich, wie in der VRR App mittels Navigationsanweisungen erfolgen. Die gesamte Fahrt soll sich der Nutzer auf einer Karte anzeigen lassen können. Des Weiteren ist beabsichtigt, dass der Nutzer die Möglichkeit hat, die Fahrten im Kalender zu speichern und zu teilen. Außerdem ist geplant, eine Schritt-für-Schritt Anleitung zu realisieren.

3.2 Vorhandene Ticketberater

3.2.1 VRR online Ticketberater

Von den drei möglichen Beratungsangeboten des VRRs, die in Abschnitt 2.3.1 vorgestellt wurden, wurde nur der online Ticketberater getestet. Die Ticketzentren konnten aufgrund der Entfernung nicht betrachtet werden. Die Hotline wurde nicht betrachtet, da ursprünglich geplant war, mehrere Fahrten zu betrachten und dies zu

umständlich gewesen wäre. Der Ticketberater des VRRs war beim Testen sehr unzuverlässig. Dabei zeigte der Berater beispielsweise eine leere Seite ohne Fahrscheine an, oder die Abfrage wurde mit einem `HTTP ERROR 503` abgewiesen.

Eine mögliche Ursache für die fehlenden Ergebnisse ist, dass der Kunde, um die Angaben zu überarbeiten, nicht „zurück“ im Browserfenster, sondern nur die explizit dafür vorgesehen Buttons *zurück* und *bearbeiten* auf der Website drücken darf. Dies funktionierte bei gleichem Browser jedoch nur auf einem von sieben getesteten Computern. Auf die Ergebnisse des Ticketberaters kann deswegen in dieser Arbeit nur bedingt eingegangen werden.
Bei einem Test der Beratung für *Ab-und-zu-Fahrer* zeigte der Berater alte Ticketpreise an. Die Empfehlung für zwei Erwachsene, die an einem Tag die Strecke *Herten Mitte – Recklinghausen HBF* hin- und zurückfahren wollen, ist ein 24-StundenTicket oder ein 4erTicket der Preisstufe A zu kaufen. Beide Empfehlungen sind korrekt, jedoch hätten bei der Berücksichtigung der Abfahrtszeiten eventuell kostengünstigere Fahrscheine gewählt werden können.

In der Beratung für Ab-und-zu-Fahrer kann der Nutzer, neben Start- und Zielhaltestelle, auch die Anzahl der reisenden Erwachsenen festlegen sowie die Anzahl der Kinder. Er hat jedoch nicht die Möglichkeit, diese Fahrt mit weiteren Fahrten zu kombinieren.
Für *Vielfahrer* wird, im Rahmen dieser Arbeit, nur die Beratung für festgelegte Strecken und für festgelegte Regionen betrachtet.
Um den Kunden zu beraten, muss der Nutzer die Strecken angeben, welche er regelmäßig fahren will. Für diese Fahrten muss er Start, Ziel und wie häufig er die Strecke voraussichtlich zurücklegen will, angeben. Er kann dabei angeben, wie oft er die jeweilige Verbindung pro Woche oder Monat nutzen und ob er die Strecke am gleichen Tag zurückfahren will. Diese Fahrten kann der Nutzer ändern, löschen oder weitere Fahrten hinzufügen. Bei der Beratung kann jedoch nicht die Anzahl der Personen festgelegt werden.
Für Kunden, die mehr Flexibilität wollen, können diese in der Suche Haltestellen oder Orte angeben, welche im Gültigkeitsbereich liegen sollen. Alternativ kann der Kunde durch Klicken auf der Wabenkarte die gewünschten Waben auswählen.

3.2.2 FAIRTIQ

In der App FAIRTIQ wählt der Nutzer einen Verkehrsverbund aus, in welchem er reisen will und hinterlegt eine Zahlungsmethode. Damit eine Fahrt einen Fahrschein erhält, muss der Nutzer seinen Standort bestimmen lassen. Passend zu diesem werden dem Nutzer mögliche Haltestellen vorgeschlagen, unter diesen muss die Starthaltestelle ausgewählt werden. Steigt der Nutzer ein, muss er dies der App mitteilen. Bei einem Umstieg muss der Nutzer die Fahrt nicht stoppen, sondern erst wenn er seine Fahrt beendet. Bei einer Kontrolle zeigt der Nutzer einen QR-Code vor. Bei einem Test im Verkehrsverbund Mittelthüringen (VMT) funktionierte das Erkennen der Starthaltestelle sehr gut. Beim Aussteigen hingegen hatte die Anwendung zunächst Probleme, die Endhaltestelle zu ermitteln, weshalb kurz an dieser gewartet werden musste. Nachdem dies erfolgt war, wurde weiterhin angezeigt: *Ermittlung der Ankunftsposition läuft.* In der Statusleiste hingegen konnte trotzdem weiterhin die

Fahrt beendet werden. Bis die Fahrt ermittelt wurde, dauerte es etwa zehn Minuten. Die Fahrt erhielt einen vorläufigen Preis. Am nächsten Tag wurde eine E-Mail mit dem endgültigen Preis der Fahrt neben der Ankündigung der Abbuchung zugestellt. Eine Besonderheit des Ticketpreises ist jedoch, dass der Preis eines Einzelfahrscheins am Automaten oder anderen Vorverkaufstellen höher ist als in der App. Bis zum 31.12.2020 wird ein Rabatt von 10 % auf die Einzelfahrt gewährt [31]. Jedoch konnte nicht getestet werden, ob und wie diese Fahrt eventuell bei einem Wochenticket berücksichtigt werden würde.
Ein Nachteil der Anwendung ist, dass der Nutzer eine weitere Anwendung zur Fahrplanauskunft benötigt, da diese nicht in FAIRTIQ enthalten ist. Dafür ist der Nutzer mit dieser Anwendung sehr flexibel und muss keine Fahrscheine mehr im Voraus erwerben. Daraus resultiert jedoch auch, dass der Nutzer die Kosten erst im Nachhinein erfährt.

3.2.3 Schlussfolgerungen für die Ticketberatung

Im Gegensatz zur Beratung für Ab-und-zu Fahrer des VRR-Ticketberaters muss der Nutzer neben Start, Ziel, Datum und der Anzahl reisender Personen ebenso die Uhrzeit der Fahrten festlegen. Im Gegensatz zu diesem Berater wird der Nutzer in der hier entwickelten Anwendung auch in der Lage sein, mehrere einzelne Fahrten zu kombinieren, ohne jedoch direkt anzugeben, dass er diese Fahrt mindestens einmal pro Monat zurücklegen will. Dabei soll der Nutzer ebenfalls nicht auf nur eine reisende Person beschränkt sein, sondern mehrere angeben können. Hier soll der Nutzer jedoch nur die Möglichkeit haben, die gültigen Tarifgebiete für Zeitfahrscheine zu wählen, falls diese nicht eindeutig bestimmt sind. Dadurch kann er den Gültigkeitsbereich an seinen Aufenthaltsort anpassen und den Fahrschein für weitere Fahrten nutzen.
In der Anwendung, die im Rahmen dieser Arbeit entwickelt wird, soll der Nutzer seine Fahrten im Voraus planen und die angegebenen Fahrscheine selbstständig kaufen. Dadurch sind die Kosten für die Fahrscheine dem Nutzer vorher bekannt. Damit ist, anders als bei FAIRTIQ, keine weitere Anwendung zur Fahrplanauskunft nötig. Die Anwendung wird keinen Zugriff auf den Nutzerstandort benötigen, da die Daten nur auf den Eingaben des Nutzers basieren. Da geplant ist, dass die Fahrscheine im Voraus bekannt sind, ist für den Nutzer ersichtlich, welche Fahrscheine für welche Fahrten genutzt werden müssen.

Bis jetzt wurden nur das geplante Aussehen und die Funktionsweise der App analysiert. Eine Betrachtung der Fahrscheine des Verkehrsverbundes erfolgt im Anschluss.

3.3 Analyse der Fahrscheine

In diesem Abschnitt werden die Preise für die einzelnen Fahrscheine verglichen, um daraus Regeln herzuleiten, die dann bei der Optimierung genutzt werden können. Dafür wird zuerst die Rabattberechnung erläutert.

3.3.1 Rabattberechnung

Sei G der Grundwert und p der Prozentsatz der Ersparnis, dann ergibt sich der rabattierte Preis W aus folgender Formel:

$$\begin{aligned} W &= G \cdot \left(1 - \frac{p}{100}\right) \\ \Leftrightarrow 1 - \frac{p}{100} &= \frac{W}{G} \\ \Leftrightarrow \frac{p}{100} &= 1 - \frac{W}{G} \end{aligned} \tag{3.1}$$

[32]. Die Ersparnis wird im Folgenden ermittelt werden, um zu vergleichen, wie viel der Kunde mit dem Kauf eines Tickets spart.

3.3.2 Einzelfahrscheine

Das Einzelticket bildet die kleinste Fahrscheineinheit und damit die Preisreferenz für alle weiteren Fahrscheine. Für das 4erTicket gilt, dass das Ticket in allen Preisstufen erst ab der vierten Fahrt eine Ersparnis gegenüber dem Kauf von entsprechend vielen Einzeltickets liefert. Diese Ersparnis liegt zwischen 6,25 % und 11,46 %. Der VRR gibt eine Ersparnis von bis zu 12 % für den Kunden an [33]. Für das *10erTicket* hingegen ist die Anzahl an Fahrten, ab welcher der Nutzer mit diesem Ticket spart, abhängig von der Preisstufe. Die Ersparnis des Fahrscheins für den Kunden gibt der VRR mit bis zu 33 % an [33]. Dabei wird jedoch nicht genannt, ob die Ersparnis gegenüber dem Kauf von zehn Einzeltickets gemeint ist oder gegenüber zwei Einzeltickets und zwei 4erTickets. Nach Formel 3.1 liegt die Ersparnis für den Kunden im zweiten Fall bei bis zu 28 % und im ersten Fall bei bis zu 33,22 %. In beiden Fällen ist die Ersparnis abhängig von der Preisstufe.
Die Kosten von zwei oder mehr Fahrscheinen kleinerer Preisstufen sind zum Teil niedriger als ein einzelner Fahrschein einer höheren Preisstufe. Somit können Fahrten in zwei oder mehr Fahrten unterteilt werden. Das ist vor allem bei Fahrten mit Umstieg gut umsetzbar.

3.3.3 Zeitfahrscheine

4-StundenTicket

Das 4-StundenTicket ist nur in den Preisstufen A1 und A2 verfügbar. Ein Vergleich mit dem Preis für das Einzelticket in der jeweiligen Preisstufe zeigt, dass für die Preisstufen A1 und A2 das Ticket sich bereits ab zwei Fahrten innerhalb von vier Stunden anbietet. Ein Vergleich mit den anderen Einzelfahrscheinen ist somit nicht sinnvoll. Die Ersparnis für diesen Fahrschein gegenüber der minimalen Anzahl an Einzelfahrscheinen beträgt 25 % für die beiden Preisstufen.

HappyHourTicket

Das HappyHourTicket kann nur für die Preisstufen A1, A2 und A3 erworben werden. Wie für das 4-StundenTicket gilt, dass der Nutzer mit dem Ticket ab zwei Fahrten

gegenüber dem Kauf von Einzeltickets spart. Auch in diesem Fall ist kein Vergleich mit den anderen Einzelfahrscheinen nötig.

Da das HappyHourTicket länger als vier Stunden gültig ist, lohnt es sich, ab 18 Uhr, dieses Ticket anstelle des *4-StundenTickets* zu kaufen. Neben der längeren Gültigkeit spart der Nutzer im Vergleich 24,05 % beim Kauf dieses Tickets.

Tagesticket

Für eine Person gilt, dass sich nach den Angaben des VRRs ein Tagesticket ab der zweiten Fahrt in den Preisstufen C und D und ab der dritten Fahrt für die verbleibenden Preisstufen rentiert [15]. Für die Preisstufe A gilt, dass sich das Ticket im Vergleich zum HappyHourTicket nur lohnt, wenn das Ticket vor 18 Uhr benötigt wird. Des Weiteren gilt für diese Preisstufe, dass wenn die Fahrten durch ein 4-StundenTicket abgedeckt werden können, dies kostengünstiger ist.

Auch für diesen Fahrschein gilt, dass ein Preisvergleich nur mit den Einzelfahrscheinen sinnvoll ist. Dabei stellt sich heraus, dass für die Preisstufen A und B bei drei Fahrten die Ersparnis über 10 % beträgt, für die Preisstufen C und D bei zwei Fahrten hingegen unter 5 % liegt.
Die Fahrscheine für mehrere Personen sind günstiger als der Kauf von mehreren Fahrscheinen für eine Person. Dabei gilt für die Preisstufe A, dass jede weitere Person mindestens zwei Fahrten absolvieren muss. Für die höheren Preisstufen tritt die Vergünstigung bereits nach einer Fahrt ein.
Für zwei Regionen, die jeweils durch ein Tagesticket abgedeckt sind und deren Fahrten innerhalb von 24 h liegen, ist ein gemeinsames Ticket, welches beide Regionen abdeckt, der höheren Preisstufe günstiger. Eine Ausnahme bildet nur das 24-StundenTicket für eine Person der Preisstufe A.

Zwei-Tagesticket

Der Kauf eines 48-StundenTickets ist kostengünstiger als der Kauf von zwei 24-StundenTickets, jedoch teurer als der Kauf eines 24-StundenTickets und eines 4-StundenTickets oder eines HappyHourTickets. Auch für diesen Fahrschein gilt, dass die Fahrscheine für mehrere Personen günstiger sind als die entsprechende Anzahl für eine Person.

Für mehrere Personen ist die minimale Anzahl an benötigten Fahrten für die jeweilige Person abhängig von der Preisstufe und liegt zwischen einer und drei Fahrten.
Für dieses Ticket gilt, genauso wie für das 24-StundenTicket, dass es günstiger ist, zwei benachbarte Regionen mit einem gemeinsamen Ticket einer höheren Preisstufe abzudecken, sofern dies möglich ist.

Verbleibende Fahrscheine

Für das 7-TageTicket, Ticket1000 und Ticket2000 lassen sich keine einfachen Regeln aufstellen, ab wann sich diese Fahrscheine rentieren. In allen drei Fällen müssen nicht nur die Einzelfahrscheine verglichen werden, sondern auch die bisher betrachteten

kleineren Zeitfahrscheine. Die Fahrten können dabei in dem Zeitraum eines Fahrscheins auf verschiedene Arten verteilt sein, sodass ein systematisches Vergleichen der Fahrscheine sehr umfangreich wäre. In Tabelle A.6 in Anhang A.2 werden für jede Preisstufe nur die minimale Anzahl an Einzelfahrscheinen ermittelt sowie die mögliche Ersparnis durch die jeweiligen Zeitfahrscheine. Bei den Einzelfahrscheinen wird dabei immer der Größtmögliche genutzt. Da die Werte je nach Preisstufe variieren, wird in Tabelle 3.5 nur die Preisstufe A1 betrachtet. Für das 7-TageTicket gilt für die Preisstufe A1, dass ab elf Fahrten in der Woche, ohne das diese sich zu einem anderen Zeitfahrschein kombinieren lassen, der Nutzer fast 11 % spart. Für die anderen beiden Fahrscheine in der gleichen Preisstufe gilt, dass der Nutzer zwar mit einem Ticket2000 gegenüber den Einzelfahrten mehr als bei einem Ticket1000 spart, jedoch liegt die minimale Anzahl an Fahrten für diesen Fahrschein höher. Wie in der Übersicht über alle Preisstufen zu sehen ist (Tabelle A.6 in Anhang A.2) gilt jedoch nicht für alle Preisstufen, dass die Ersparnis mit einem Ticket2000 höher ist. Wie der Tabelle A.6 zu entnehmen ist, ist die minimale Anzahl an Einzelfahrten für alle Preisstufen für das Ticket1000 kleiner als für das Ticket2000.

Aufgrund der geringeren Fahrtenanzahl ergibt sich, dass sich das Ticket2000 gegenüber dem Ticket1000 nur anbietet, wenn die Vorteile dieses Fahrscheins genutzt werden, da das Ticket1000 gegenüber dem Ticket2000 7 % bis 12 %, je nach Preisstufe, günstiger ist.

Fahrschein	Anzahl Fahrten	Ersparnis in %
7-TageTicket	11	10,70
Ticket1000	31	0,42
Ticket2000	35	1,70

Tabelle 3.5: Minimale Anzahl an Fahrten für die Fahrscheine und die jeweilige Ersparnis in % gegenüber Einzelfahrscheinen für die Preisstufe A1.

3.3.4 Konsequenzen aus den Fahrscheinen

In der Anwendung ist geplant, alle Einzelfahrscheine zu betrachten. Dabei müssen die unterschiedlichen Anzahlen an Fahrten, damit sich ein Fahrschein rentiert, berücksichtigt werden.
Da die Zeitfahrscheine eine komplexere Struktur besitzen, werden folgende Vereinfachungen angenommen: Bei der Optimierung werden die Preisstufen getrennt betrachtet. Dabei wird jedoch die kleinste Preisstufe (K) ausgeschlossen, da für diese keine Fahrscheine existieren sowie nur kurze Strecken abgedeckt werden. Auch soll nicht versucht werden, Fahrscheine mit einer geringeren Preisstufe durch Fahrscheine mit einer höheren Preisstufe abzudecken. Des Weiteren wird für jeden Fahrschein die minimale Anzahl an Fahrten aus Tabelle A.2, damit sich dieser gegenüber den Einzelfahrscheinen rentiert, verwendet. Nur wenn entsprechend viele Fahrten durch das Ticket abgedeckt werden können, wird das Ticket genutzt. Die erweiterten Nutzungsmöglichkeiten von Zeitfahrscheinen sind an Wochenende und Feiertage gebunden, werden jedoch zunächst nur an Wochenenden berücksichtigt.

Von den 30-TageTickets soll nur das Ticket1000 betrachtet werden, welches den ganzen Tag gültig ist. Dabei wird jedoch die Möglichkeit, bis zu drei Kinder unentgeltlich mitzunehmen, nicht berücksichtigt.
Hinzu kommt, dass die Nutzergruppe als fest angenommen wird. Dadurch ist zum einen sichergestellt, dass die gleichen Personen die Fahrscheine nutzen, zum anderen entsteht so insgesamt ein geringerer Preis, auch wenn dies für den einzelnen in einem höheren Preis resultieren kann. Es werden nur die reisenden Personen betrachtet, die Mitnahme von Fahrrädern wird nicht berücksichtigt.
Aus der Analyse ergibt sich folgende Reihenfolge der Fahrscheine für die Zeitfahrscheine für die Optimierung:

1. HappyHourTicket
2. 4-StundenTicket
3. 24-StundenTicket
4. 48-StundenTicket
5. 7-TageTicket
6. 30-TageTicket

Wenn möglich, sollten Zeitfahrscheine genutzt werden, da diese günstiger sind als entsprechend viele Einzelfahrscheine. Deshalb ist geplant, die Fahrten zuerst mit Zeitfahrscheinen und die verbleibenden Fahrten mit Einzelfahrscheinen zu optimieren.

3.4 Optimierungsansätze

Mögliche Optimierungsansätze sind auf der einen Seite die Fahrten mit Einzelfahrscheinen abzudecken oder auf der anderen Seite mit Zeitfahrscheinen. Diese beiden Ansätze lassen sich auch kombinieren. Ein weiterer Ansatz wäre, die Fahrten an den Umstiegen aufzuteilen. Dafür müssten alle Abschnitte einzeln betrachtet werden, sowie alle aufeinander folgenden Abschnitte. Bei einer Fahrt mit zwei Umstiegen und folglich den drei Abschnitten a_1, a_2 und a_3 ergäben sich folgende Fahrten, die mit den beiden anderen Ansätzen optimiert werden müssten:

$$\{a_1, a_2, a_3\} - \{a_1a_2, a_3\} - \{a_1, a_2a_3\} \text{ und } \{a_1a_2a_3\}$$

Aus diesen vier Möglichkeiten würde dann die kostengünstigste Aufteilung gewählt werden.

3.4.1 Konsequenzen für die Optimierung

Von den drei vorgestellten Optimierungsansätzen werden nur zwei umgesetzt, einerseits die Optimierung mit Einzelfahrscheinen, andererseits die Optimierung mit Zeitfahrscheinen. Wenn möglich, werden diese beiden Ansätze kombiniert. Der Ansatz der Aufteilung der Fahrten wird nicht betrachtet werden, weil dieser mit einem

hohen Rechenaufwand einhergeht. Bei der Optimierung ist geplant, wie in der Analyse der Fahrscheine ermittelt, zuerst mit Zeitfahrscheinen möglichst viele Fahrten abzudecken und anschließend Einzelfahrscheine für die verbleibenden Fahrten zu verwenden.

3.5 Schlussfolgerungen

Da die Optimierung im Fokus dieser Arbeit liegt, wird die umgebene App nur auf Handys mit dem Betriebssystem Android 8.0 oder höher laufen. Die Wahl fiel auf Android, da aktuell die Anzahl an Smartphones mit diesem Betriebssystem in Deutschland am höchsten ist [34]. Weltweit ist die Anzahl an Nutzern mit der Version 8.0 am höchsten [35]. Außerdem stand zur Entwicklung nur ein Android Gerät mit dieser Version zur Verfügung. Auch kann die Anwendung von Smartphones mit neueren Versionen genutzt werden, da neue Android Versionen abwärtskompatibel sind [36]. Da das zur Verfügung stehende Handy eine Displaygröße von etwa 12 cm besitzt, wird nur die Darstellung auf dieser Größe betrachtet. Bei anderen Displaygrößen kann es somit zu verzerrten Darstellungen kommen.
Als Basis für die eigene Anwendung wird die Anwendung Öffi gewählt, da diese die einzige hier betrachtete Open-Source Anwendung ist. Zusätzlich ist Öffi abwärtskompatibel bis zu Android Version 4.0 und ist somit kompatibel mit Version 8.0 [29, 36]. Aufgrund von Technologien, die benutzt werden, wird nur eine Abwärtskompatibilität bis zu Version 8.0 erreicht. Die Verbindungsdaten werden dabei vom VRR angefordert. Dieser bietet ein OpenData-Portal an, welches über eine öffentliche Schnittstelle verfügt. Mithilfe dieser Schnittstelle kann auf das Testsystem zugegriffen werden. Das System ermöglicht den Zugriff auf Echtzeitinformationen, Betriebsstörungen und gibt weitere Auskünfte [37].
Die Anwendung, welche im folgenden Kapitel entworfen wird, soll dabei die einzelnen Konsequenzen für die Routenoptionen und Übersichtlichkeit (Abschnitt 3.1.3), die Fahrscheinberatung (Abschnitt 3.2.3) sowie die Fahrscheine (Abschnitt 3.3.4) erfüllen. Des Weiteren werden nur die zwei Optimierungsansätze Einzelfahrscheine und Zeitfahrscheine kombiniert betrachtet werden, wie in Abschnitt 3.4.1 erläutert. Da in die Anwendung kein Kauf von Fahrscheinen eingebunden sein wird, muss die feste Nutzergruppe den Kauf der Fahrscheine selber übernehmen. Aufgrund dessen hat die Anwendung keinen Zugriff auf die Fahrscheine, weshalb das Entwerten der Fahrscheine in den Verantwortungsbereich des Nutzers fällt. Eine Abweichung des Nutzers von den angegebenen Fahrscheinen soll in dieser Arbeit nicht berücksichtigt werden. Es wird davon ausgegangen, dass der Nutzer die Fahrt so antritt, wie geplant. Bei Änderungen muss die Fahrt angepasst oder gelöscht werden. Falls der Nutzer eine Fahrt doch nicht antritt, so muss er die Fahrt löschen, damit diese bei der Optimierung nicht weiter berücksichtigt wird.

Kapitel 4

Entwurf

In diesem Kapitel wird der Entwurf der App vorgestellt. Dafür werden zunächst die einzelnen Anwendungsfälle erläutert. Anschließend werden die geplanten Abläufe genauer erklärt, indem die jeweiligen Aktivitätsdiagramme von einzelnen Aktivitäten vorgestellt werden.

Danach wird das Klassendiagramm, dann das geplante User Interface und zum Schluss das geplante Vorgehen der Implementierung präsentiert.

4.1 Anwendungsfalldiagramm

In Abbildung 4.1 sind die einzelnen Anwendungsfälle der App zu sehen. In der App soll sich der Nutzer über Fahrscheine sowie zukünftige Fahrten informieren können. Dabei sollen die Fahrten, die einem Fahrschein zugeordnet sind, jeweils in der Fahrscheinübersicht angezeigt werden.

Bei der Planung mehrerer Fahrten fügt der Nutzer mehrere einzelne Fahrten hinzu. Für jede einzelne Fahrt kann der Nutzer die Einstellungen bearbeiten. Nachdem der Nutzer die Informationen für eine einzelne Fahrt angegeben hat, werden ihm mögliche Verbindungsvorschläge angezeigt, die vom VRR-Testserver ermittelt werden. Unter diesen Vorschlägen wählt der Nutzer eine Verbindung aus. Die vom Nutzer gewählten Fahrten werden anschließend optimiert und die entsprechenden Tickets zugewiesen.
Nach der Optimierung der Fahrten muss der Nutzer gegebenenfalls noch den Gültigkeitsbereich einzelner Fahrscheine festlegen.
Eine detaillierte Erläuterung der einzelnen Anwendungsfälle erfolgt in Tabelle 4.1.

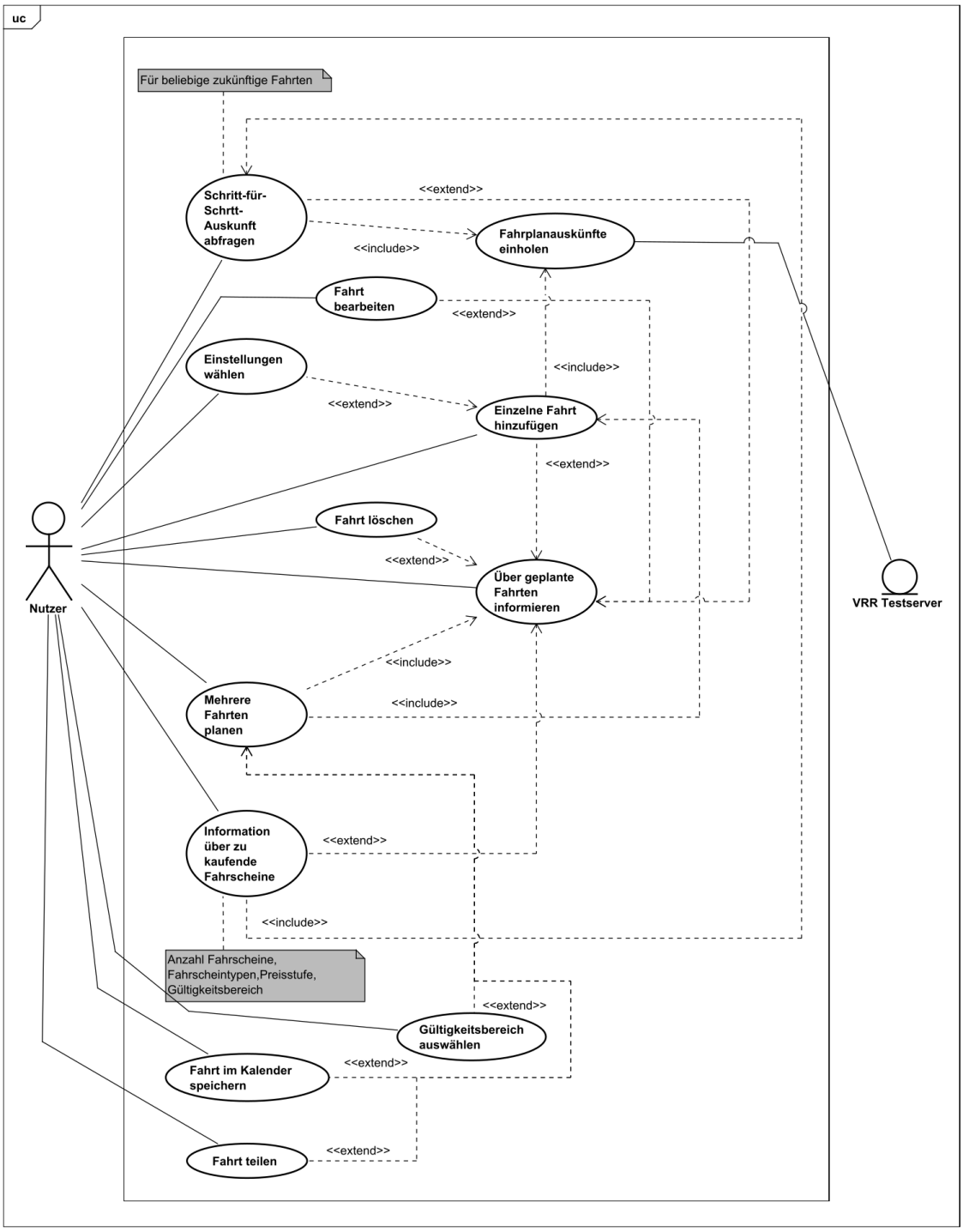


Abbildung 4.1: Anwendungsfalldiagramm.

Anwendungsfall	Erläuterung
Einstellungen wählen	Einstellung der Verkehrsmittel, welche bei der Routenfindung berücksichtigt werden sollen, sowie der Barrierefreiheit, der Route und des Gehtempos.
Mehrere Fahrten planen	Hauptfunktion der Anwendung: Eingabe der geplanten Fahrt, Anzeige der Fahrt mit der Option, weitere Fahrten zu ergänzen, zu editieren oder zu löschen. Im Anschluss Ermittlung und Anzeige der notwendigen Fahrscheine.
Einzelne Fahrt hinzufügen	Für eine einzelne Fahrt kann der Nutzer den Start- und Zielpunkt sowie die Anzahl der reisenden Personen und die Ankunfts- oder Abfahrtszeit festlegen. Für diese Angaben ermittelt der VRR-Testserver passende Verbindungen. Aus diesen Verbindungen wählt der Nutzer eine Geeignete aus.
Fahrt bearbeiten	Der Nutzer kann die einzelnen Punkte (Start-, Zielpunkt, An- beziehungsweise Abfahrtszeit, Personenanzahl) für eine Fahrt bearbeiten oder eine andere Verbindung wählen.
Fahrt löschen	Die vom Nutzer gewählte Fahrt wird aus der Planung entfernt.
Über geplante Fahrten informieren	Dem Nutzer werden alle zukünftigen Fahrten angezeigt. Diese Fahrten können bearbeitet und gelöscht werden, der Nutzer kann außerdem neue Fahrten hinzufügen.
Information über zu kaufende Fahrscheine	Dem Nutzer werden die zu kaufenden Fahrscheine (Fahrscheintyp und Preisstufe) angezeigt. Für die einzelnen Fahrscheine lassen sich die jeweiligen Fahrten anzeigen, für die dieser Fahrschein genutzt werden soll.
Schritt-für-Schritt Auskunft abfragen	Für eine einzelne Fahrt wird dem Nutzer jeder einzelne Schritt angezeigt. Im ersten Schritt wird dem Nutzer mitgeteilt, welches Ticket für die jeweilige Fahrt genutzt werden soll. Durch einen Swipe nach links oder rechts wird der nächste beziehungsweise der vorherige Schritt angezeigt. Für Abschnitte, die zu Fuß zurückgelegt werden, wird zum einen ein Kartenausschnitt angezeigt, zum anderen werden die einzelnen Navigationsanweisungen angezeigt.
Fahrplanauskünfte einholen	Der VRR-Testserver ermittelt passend zu den Nutzereingaben mögliche Verbindungen.

Anwendungsfall	Erläuterung
Region auswählen	Bei Zeitfahrscheinen gibt es verschiedene Bereiche, in denen ein Ticket der Preisstufe A, B oder C gültig ist. Wenn dieser Bereich nicht eindeutig durch die geplanten Fahrten festgelegt ist, muss der Nutzer den Bereich selbst wählen.
Fahrt im Kalender speichern	Der Nutzer hat die Möglichkeit, jede Fahrt im Kalender zu speichern.
Fahrt teilen	Wenn der Nutzer die Fahrt teilt, werden die wesentlichen Schritte der Fahrt als Text formuliert, und in den Messenger, den der Nutzer wählt, eingefügt.

Tabelle 4.1: Erläuterung der Anwendungsfälle.

4.2 Aktivitätsdiagramme

In diesem Abschnitt werden die geplanten Abläufe für die Anwendungsfälle: *mehrere Fahrten planen, neue Fahrt hinzufügen* sowie *Fahrt bearbeiten* vorgestellt.

4.2.1 Mehrere Fahrten planen

Kurzbeschreibung:
Der Nutzer plant mindestens eine Fahrt. Danach wird die kostengünstigste Fahrscheinkombination ermittelt und gespeichert.

Akteure:
Benutzer des Systems.

Auslöser:
Der Benutzer wählt die Aktion *mehrere Fahrten planen* aus.

Vorbedingung:
Keine.

Nachbedingung:
Die benötigten Fahrscheine werden an diese Fahrten angepasst, die neuen Fahrten sind ebenfalls in der Liste zukünftiger Fahrten gespeichert.

Standardablauf:

1. Planen einer einzelnen Fahrt
2. Dem Nutzer werden die geplanten Fahrten angezeigt
3. Der Nutzer kann weitere Fahrten hinzufügen
4. Berechnung der kostengünstigsten Fahrscheinkombination

5. Speichern dieser Kombination

Mögliche Fehler:
Die möglichen Fehler aus den Aktivitäten *neue Fahrt hinzufügen* und *über geplante Fahrten informieren*.

Alternativablauf:
Der Nutzer klickt *zurück*, es folgt eine Sicherheitsabfrage, ob er wirklich abbrechen will.
Wenn durch die Fahrten nicht eindeutig festgelegt ist, in welchem Bereich Zeitfahrscheine gelten, so muss der Nutzer diese manuell aus den möglichen Regionen wählen.

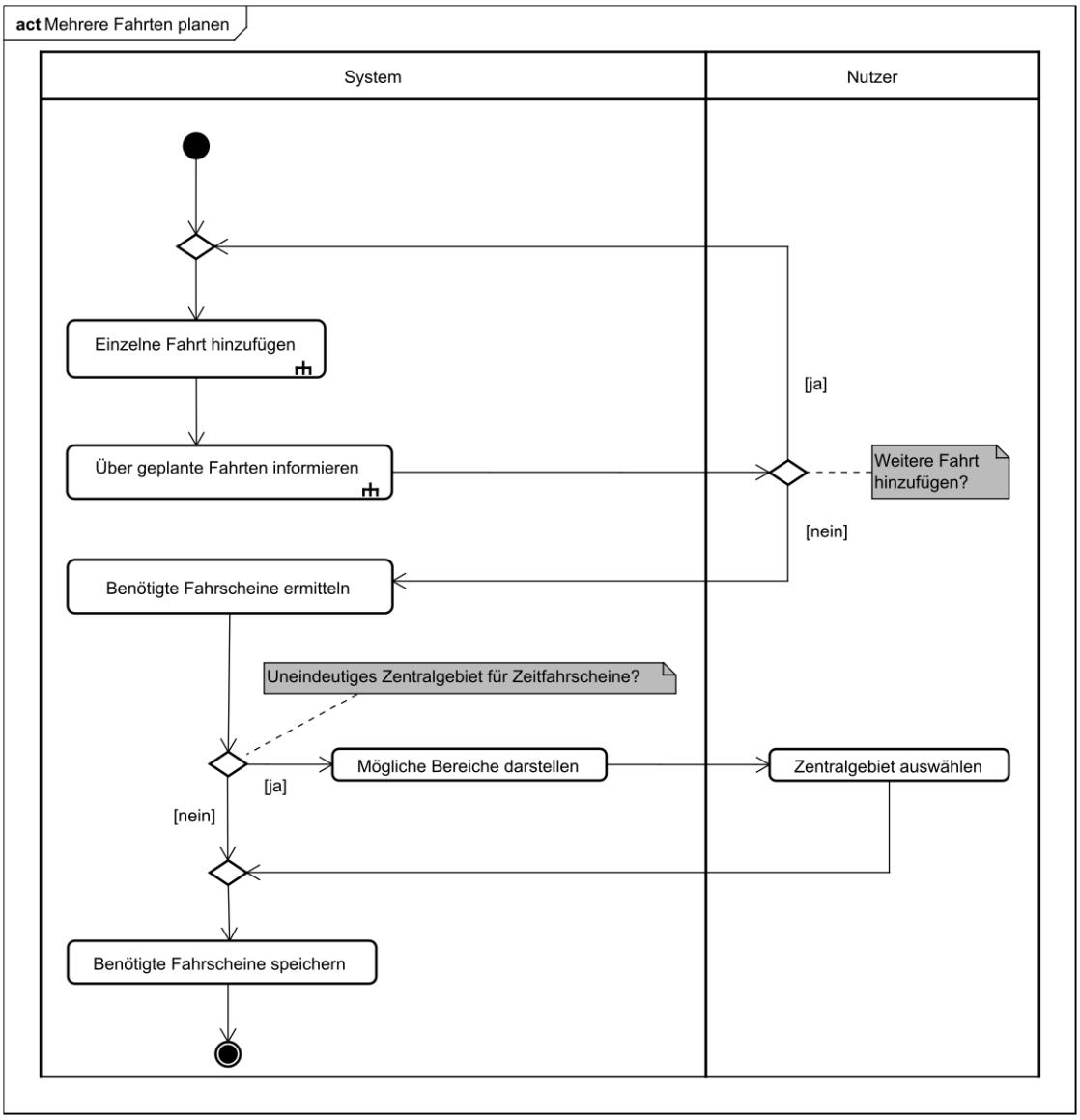


Abbildung 4.2: Aktivitätsdiagramm für *mehrere Fahrten planen*.

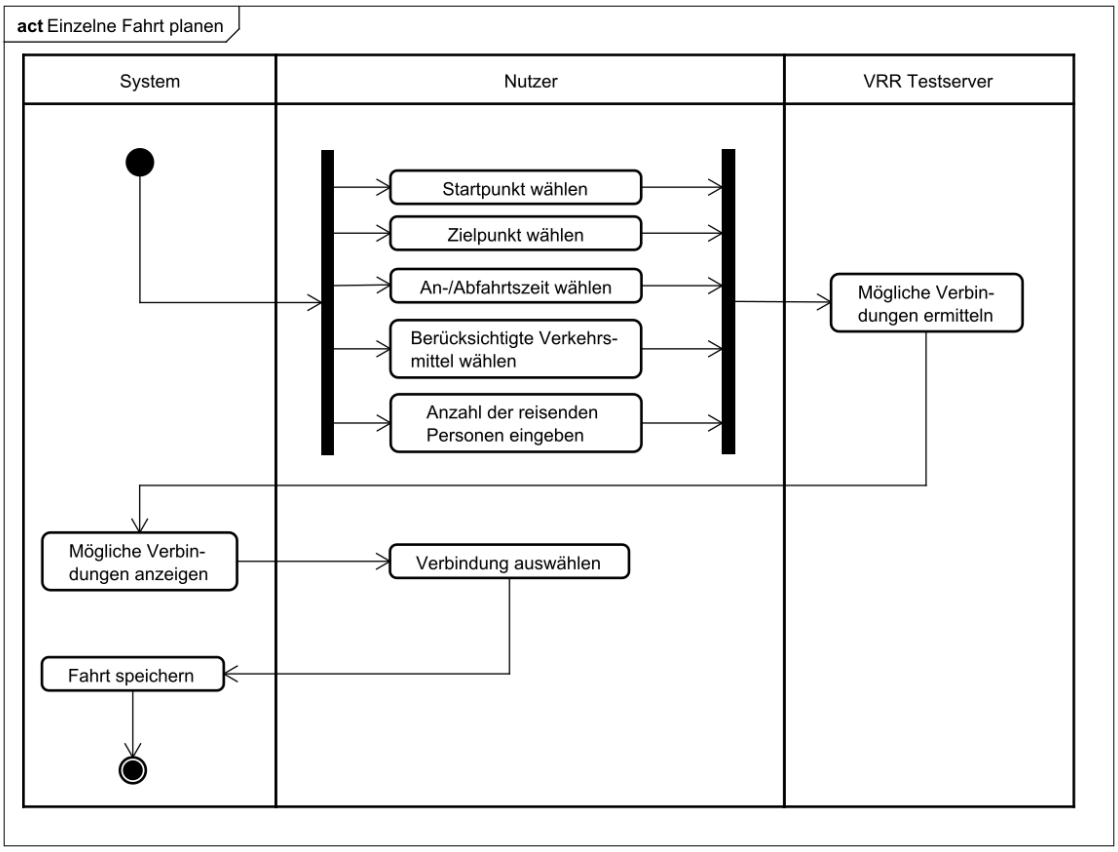


Abbildung 4.3: Geplanter Ablauf der Aktivität *einzelne Fahrt planen*.

4.2.2 Einzelne Fahrt planen

Kurzbeschreibung:
Der Nutzer plant eine einzelne Fahrt, dazu füllt er das Formular aus und wählt anschließend unter den vorgeschlagenen Verbindungen eine für ihn passende aus.

Akteure:
Benutzer des Systems.

Auslöser:
Der Nutzer will mehrere Fahrten planen, er plant eine einzelne Fahrt oder bearbeitet eine bestehende Fahrt.

Vorbedingung:
Keine.

Nachbedingung:
Die vom Nutzer gewählte Fahrt wird in der Liste zukünftiger Fahrten gespeichert.

Standardablauf:
1. Der Nutzer gibt für die Fahrt die wesentlichen Angaben Start-, Zielpunkt, An- oder Abfahrtszeitpunkt, berücksichtigte Verkehrsmittel, Anzahl der reisenden Personen an.

2. Für diese Angaben ermittelt der VRR-Testserver passende Verbindungen.
3. Aus diesen wählt der Nutzer eine für ihn passende Fahrt aus.
4. Die Fahrt mit allen Angaben wird dem Nutzer detailliert angezeigt. Sollte dem Nutzer die Verbindung doch nicht zusagen, so kann er sich mittels *zurück* die möglichen Verbindungen erneut anzeigen lassen.
5. Anschließend wird die Fahrt hinzugefügt und gespeichert, damit die Details offline verfügbar sind.

Mögliche Fehler:
Der Nutzer gibt einen ungültigen Start- oder Zielpunkt an oder einen Zeitpunkt, der bereits in der Vergangenheit liegt. Über den jeweiligen Fehler wird der Nutzer informiert und muss diesen beheben.

Alternativablauf:
In der Ansicht möglicher Verbindungen hat der Nutzer die Möglichkeit, spätere und frühere Verbindungen anzufordern.
Klickt der Nutzer auf *zurück*, so muss er bestätigen, dass er wirklich beenden will. In diesem Fall wird keine neue Fahrt hinzugefügt.

4.2.3 Fahrt bearbeiten

Kurzbeschreibung:
Der Nutzer kann die einzelnen Angaben zu einer Fahrt ändern und eine neue Verbindung auswählen.

Akteure:
Der Benutzer des Systems.

Auslöser:
Der Nutzer wählt die Funktion *Bearbeiten* aus.

Vorbedingung:
Der Nutzer hat eine Fahrt ausgewählt.

Nachbedingung:
Wenn die Veränderung durchgeführt werden soll, ist die Änderung gespeichert und wird in der Übersicht angezeigt.

Standardablauf:
In der Planungsphase wird die Fahrt gelöscht und der Nutzer plant eine neue Fahrt. Dabei ist das Formular schon ausgefüllt. Anschließend wird die gewählte Fahrt vom Nutzer gespeichert.
Für bereits optimierte Fahrten hingegen, wird nachdem der Nutzer die Fahrt gewählt hat, verglichen, ob sich dadurch eine Änderung in den Fahrscheinen ergibt. Falls dies der Fall ist, wird der Nutzer darüber informiert. Der Nutzer entscheidet dann, ob die Änderung trotzdem durchgeführt werden soll oder nicht. Falls nicht, wird die alte Fahrt gespeichert und keine Änderung an den Fahrscheinen vorgenommen.

Mögliche Fehler:
Fehler aus den Aktivitäten *Fahrt löschen* und *einzelne Fahrt planen*.

Alternativablauf:
Sollte der Gültigkeitsbereich von Zeitfahrscheinen nicht eindeutig sein, so muss der Nutzer diesen manuell wählen.
Klickt der Nutzer auf *zurück*, so wird ebenfalls die vorherige Fahrt in die Fahrtenliste eingefügt.

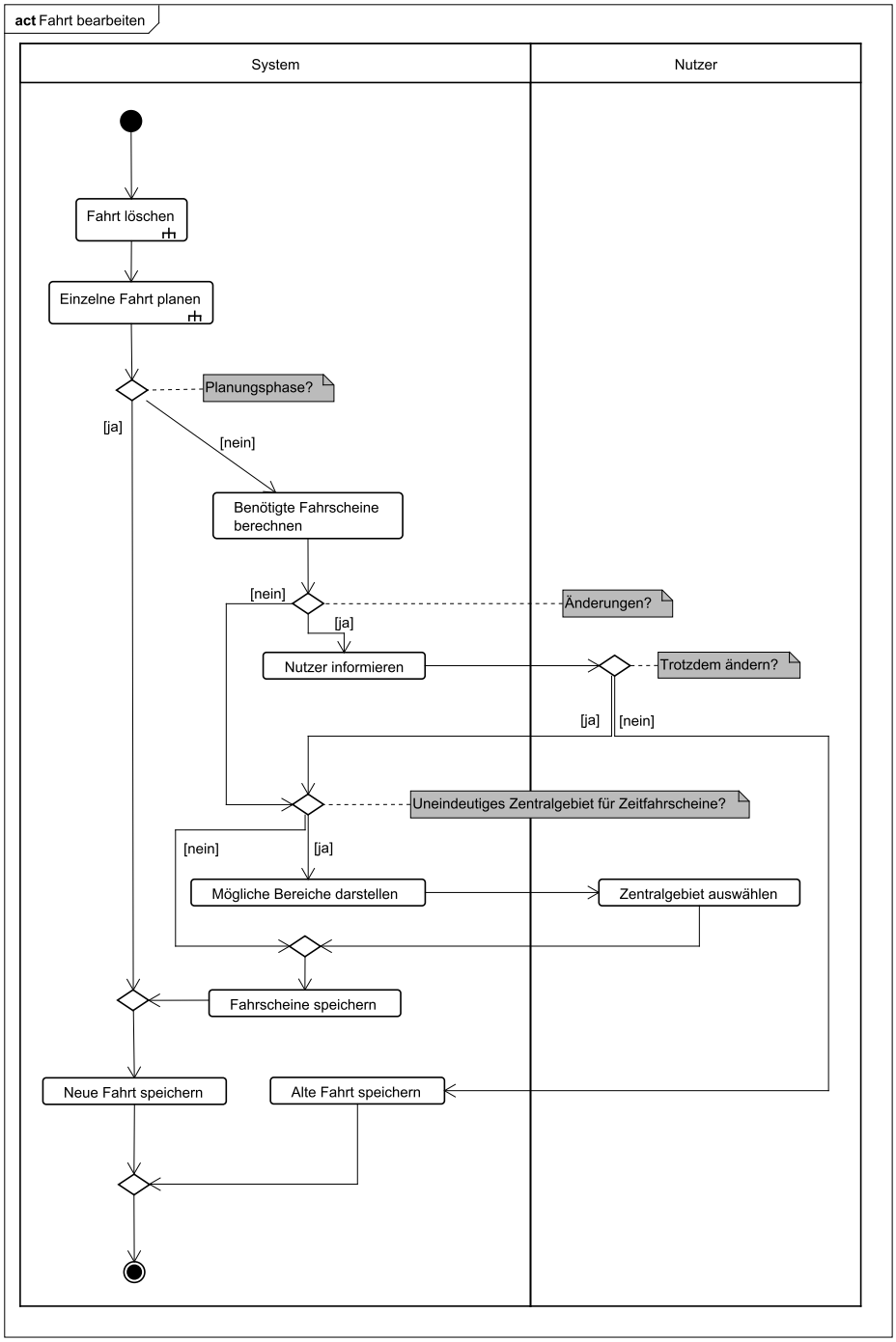


Abbildung 4.4: Aktivitätsdiagramm für die Aktivität *Fahrt bearbeiten*

4.3 Klassendiagramm

In diesem Abschnitt wird das Klassendiagramm vorgestellt. Dieses wird im Lauf der Implementierung erweitert. Da Öffi als Basis für die eigene Anwendung genutzt wird, werden nur die hier wesentlichen Klassen, welche im Verlauf der Implementierung vermutlich genutzt werden, vorgestellt. Des Weiteren wird die geplante Realisierung von Fahrscheinen vorgestellt.

4.3.1 Klassen aus Öffi

Die wesentlichen Klassen aus Öffi lassen sich in zwei Gruppen unterteilen: Eine Gruppe enthält die Informationen für eine Fahrt. Die Klassen der anderen Gruppe ermöglichen den Zugriff auf die Schnittstelle des VRRs.

Zunächst wird auf die zuletzt genannten Klassen eingegangen. Diese sind im oberen Bereich von Abbildung 4.5 zu sehen. Damit der Nutzer den Start- und Zielpunkt einer Fahrt wählen kann, müssen passend zu seinen Eingaben Vorschläge bestimmt werden. Zwischen diesen Punkten müssen anschließend passende Verbindungen ermittelt werden. In dem Interface `NetworkProvider` gibt es die Funktionen `suggestLocations` und `queryTrips`, welche diese Aufgaben erfüllen (vergleiche Abbildung 4.5). Realisiert werden diese zwei Funktionen in der Klasse `AbstractEfaProvider`. In dieser Klasse wird jeweils eine entsprechende Anfrage an den Server geschickt und dabei dessen Antwort *geparst*. In der Klasse `VrrProvider`, welche die Klasse `AbstractEfaProvider` erweitert, wird die URL, an welche die Anfragen gestellt werden, festgelegt sowie einige Besonderheiten der Linien des Providers abgefangen. Neben diesen zwei Funktionen besitzt das Interface `NetworkProvider` die Funktion `queryMoreTrips`. Mithilfe dieser Funktion können Fahrten, die früher oder später als die bisherigen Fahrten starten, der Liste möglicher Verbindungen hinzugefügt werden.
Im unteren Bereich der Abbildung 4.5 ist die Darstellung einer einzelnen Fahrt zu sehen. Die wesentliche Klasse ist `Trip`, welche alle Informationen zu einer Fahrt enthält. Neben einer ID zur Identifikation verfügt die Klasse über die Start- und Zielpunkte einer Verbindung, aber über die Kosten der Verbindung. Die verfügbaren Nutzerklassen stehen in dem Enum `Type`. Für jede Nutzerklasse werden die Kosten der Verbindung angegeben. Die Klasse `Fare` enthält dabei den Preis für das Einzelticket, die Währung, den Anbieter der Verbindung inklusive der Preisstufe für die betrachtete Verbindung. Zusätzlich ist jeder Fahrt eine Liste an Verbindungen (`Legs`) zugeordnet. Jedes `Leg`-Objekt steht für einen einzelnen Abschnitt zwischen zwei Punkten. Die Segmente unterscheiden sich dadurch, ob sie mit dem öffentlichen Personennahverkehr zurückgelegt werden (`Public`) oder nicht (`Individual`).

Die Klasse `Individual` besitzt als weitere Attribute die Dauer und Entfernung des betrachteten Abschnitts sowie eine Start- und Endzeit. Über das innere Enum `Type` der Klasse `Individual` wird festgelegt, wie der Abschnitt bewältigt werden soll.

Die Klasse `Public` erweitert die Oberklasse um die Einstiegs- und Ausstiegs- und die Endhaltestelle der Linie. Die von der Klasse `Public` verwendeten Klassen sind in Abbildung 4.6 zu sehen. Neben den einzelnen Haltestellen verfügt *Public* auch

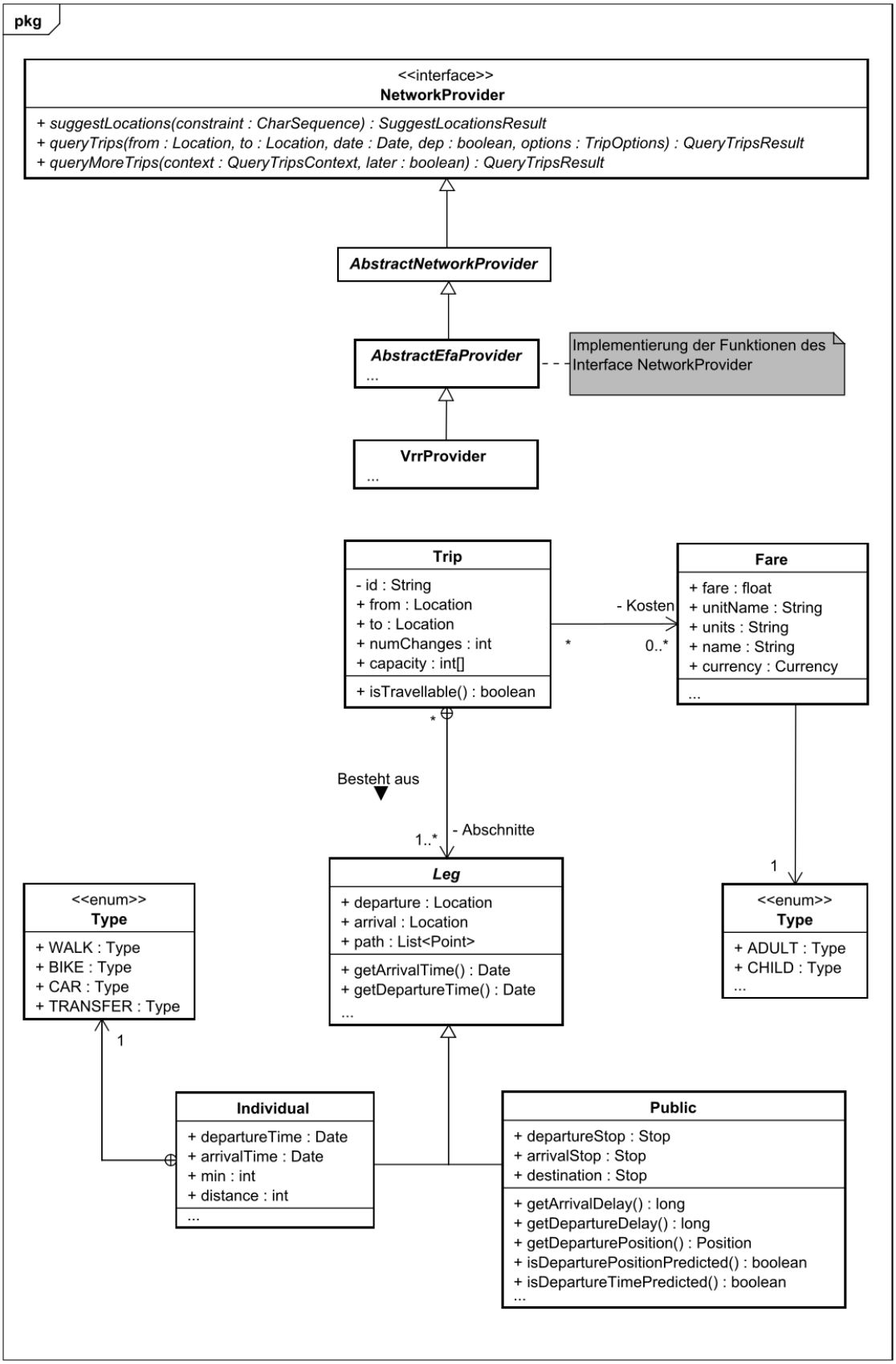


Abbildung 4.5: Klassen aus Öffi [29].

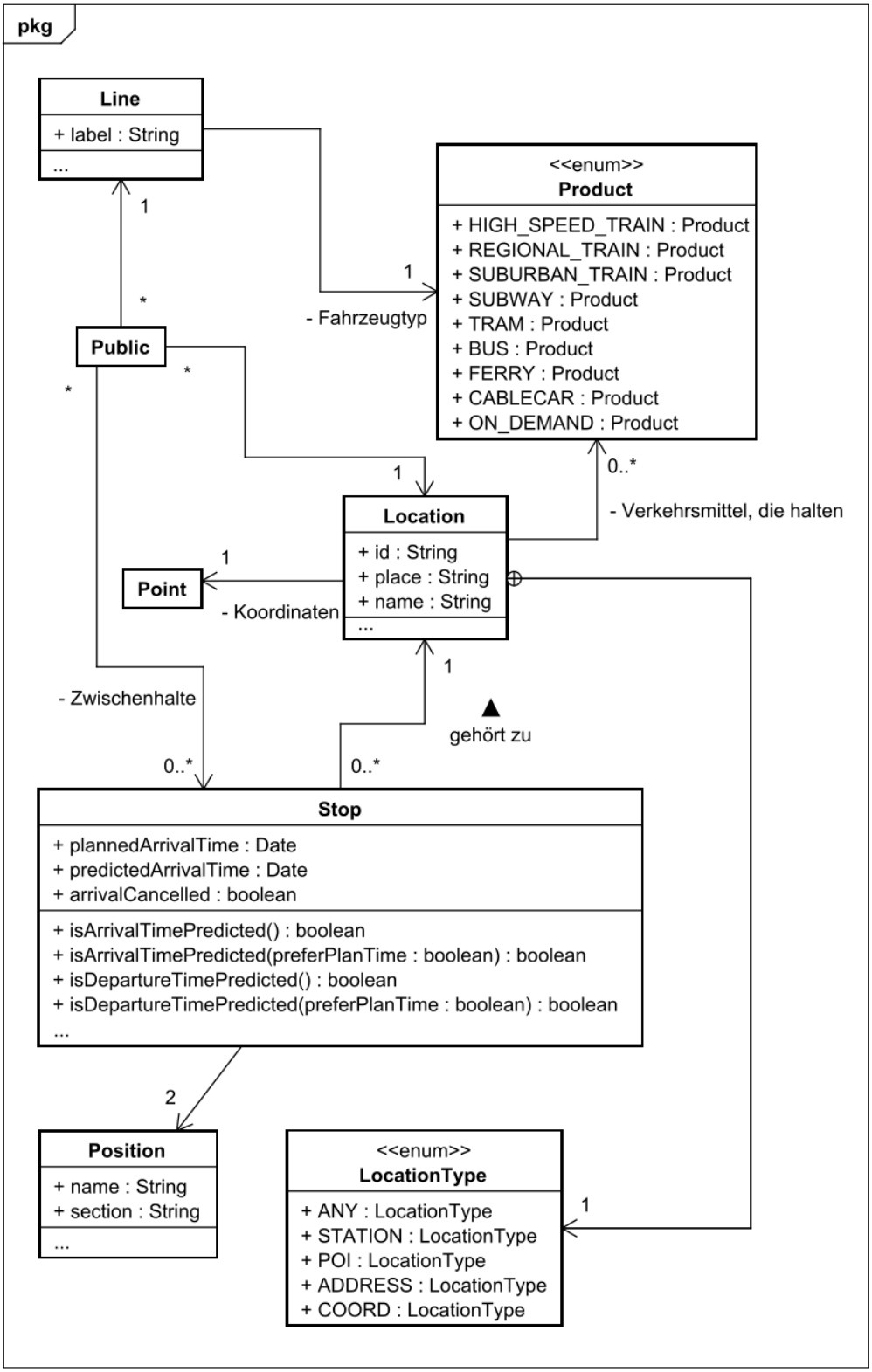


Abbildung 4.6: Klassen für die Darstellung eines ÖPNV-Abschnitts in Öffi [29].

über eine Liste an Zwischenhalten. In `Line` ist der Betreiber, Linienname und das Verkehrsmittel der Verbindung enthalten. Dabei nimmt das Verkehrsmittel einen Wert des Enums `Product` an.

Bei einem öffentlichen Abschnitt ist kein direkter Zugriff auf die Start- beziehungsweise Endzeit möglich, dies ist nur über die jeweiligen Methoden möglich, welche auf die entsprechenden Haltestellen zugreifen können. Jede Haltestelle (`Stop`) hat als Attribute die geplanten und aktuellen An- und Abfahrtsorte sowie -zeiten und einen zugeordneten Ort.
Ein Objekt der Klasse `Location` hat eine ID, einen Namen, zum Beispiel Hauptbahnhof und einen Ort. Jedes Objekt hat einen anderen `LocationType`. Wenn der `LocationType` den Wert `STATION` besitzt, so gibt die Liste an `Product` Objekten an, welche Verkehrsmittel an diesem Ort halten. Ein Ort hat dabei eindeutig festgelegte GPS-Koordinaten (`Point`).
Die Klasse `Position` gibt an, von welchem Gleis oder Steig das Verkehrsmittel abfährt. Bei längeren Bahnsteigen kann auch ein Abschnitt angegeben sein.

4.3.2 Fahrscheine

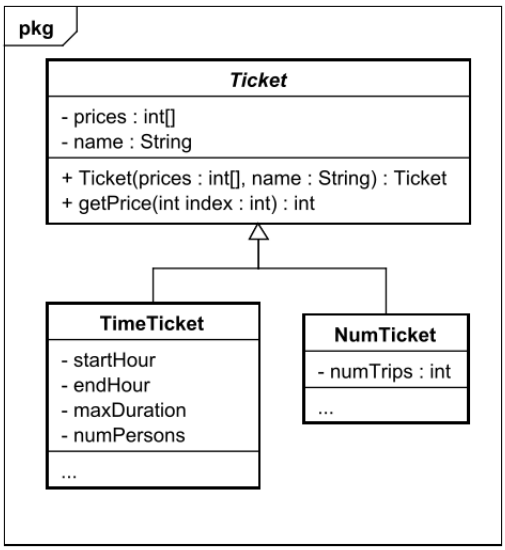


Abbildung 4.7: Klassen für die Fahrscheine.

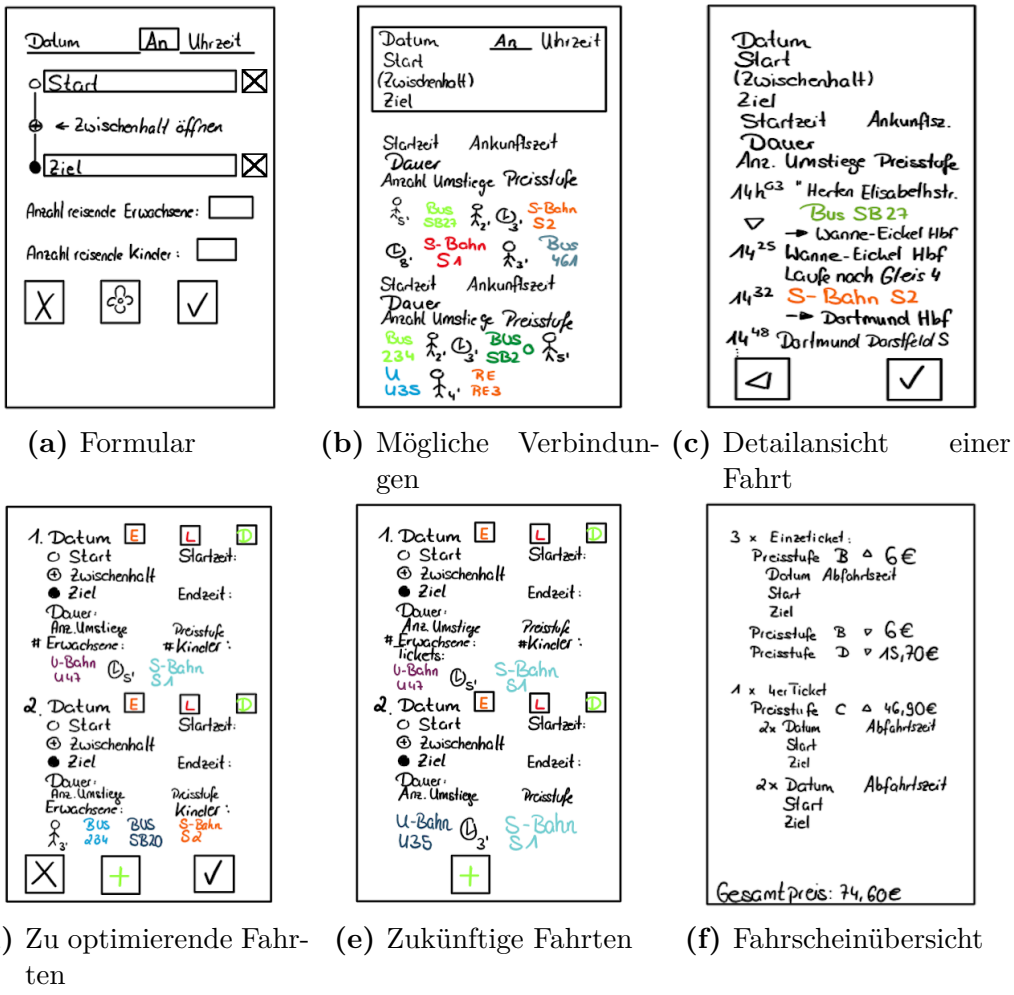
Für die Optimierung werden Klassen benötigt, welche den Fahrscheinen entsprechen, siehe Abbildung 4.7. Diese lassen sich in die Gruppen Einzelfahrscheine und Zeitfahrscheine unterteilen. Diese Unterteilung soll in der Anwendung beibehalten werden. Die Klasse `NumTicket` symbolisiert die Einzelfahrscheine mit einer festen Anzahl an Fahrten und die Klasse `TimeTicket` die Zeitfahrscheine mit beliebig vielen Fahrten in einem Zeitfenster. Beide Fahrscheingruppen haben jeweils einen Namen für den Fahrschein sowie ein Array für die Preise der einzelnen Preisstufen. Wenn ein Fahrschein für eine Preisstufe nicht verfügbar ist, so wird der Preis auf unendlich gesetzt.

Die Klasse `NumTicket` erhält zusätzlich das Attribut `numTrips`, welches angibt, wie viele Fahrten mit diesem Ticket zurückgelegt werden können. Die Klasse

`TimeTicket` hingegen benötigt außerdem die Attribute `startHour` und `endHour`. Diese geben an ab beziehungsweise bis wie viel Uhr ein Ticket gültig ist. Dies ist beispielsweise für das HappyHourTicket nötig, da dies wie in Tabelle 2.2 angegeben, nur zwischen 18 Uhr und 6 Uhr gültig ist. Das Attribut `maxDuration` gibt an, wie viele Millisekunden ein Ticket maximal gültig ist. Für den Fall, dass Fahrscheine von mehreren Personen genutzt werden können, wie zum Beispiel das 24-StundenTicket, muss die entsprechende Personenanzahl angegeben werden.

4.4 Layout Entwurf

Das Layout der App ist in verschiedene einzelne Ansichten unterteilt. Bei dem Entwurf wurden zum einen die Anforderungen an die App aus Abschnitt 3.5 berücksichtigt, zum anderen ist die Abfolge der einzelnen Ansichten durch den Ablauf der Aktivitäten *eine Fahrt hinzufügen* (Abschnitt 4.2.2) und *mehrere Fahrten planen* (Abschnitt 4.2.1) beeinflusst. In Abbildung 4.8 sind die einzelnen skizzierten Ansichten zu sehen. Abbildung 4.8a zeigt dabei die Ansicht, in der der Nutzer alle Angaben zur Fahrt macht. Diese Ansicht wird sowohl für Fahrten, die optimiert werden sollen, als auch für die reine Fahrplanauskunft genutzt. Dabei werden die Felder zur Anzahl der Personen im zweiten Fall ausgeblendet. Um zwischen Ankunftszeit und Abfahrtszeit zu wechseln, klickt der Nutzer auf einen Button. Über einen Klick auf den mittleren Button in der unteren Leiste öffnen sich die Einstellungen. Über den Knopf links davon kann die Abfrage abgebrochen werden. Damit sind zwei der drei Komfortfunktionen von der VRR-App übernommen worden. Das Vertauschen von Start und Ziel wurde aus Platzgründen ausgelassen.
Klickt der Nutzer auf den Haken in der Menüleiste, soll die Ansicht in die Anzeige möglicher Verbindungen (Abbildung 4.8b) wechseln, wie in der Aktivität *einzelne Fahrt hinzufügen* geplant. Diese Anzeige ist in zwei Teile geteilt. Im oberen Bereich werden die Nutzerangaben und im unteren Bereich werden die möglichen Verbindungen präsentiert. Dabei wird, wie in Abschnitt 3.1.3 geplant, der Nutzer für jede Verbindung über die Abfahrts- und Ankunftszeit, Dauer, Anzahl an Umstiegen und die Preisstufe der jeweiligen Verbindung informiert. Unter diesen Angaben ist die Abfolge der Verkehrsmittel erkennbar. Tippt der Nutzer auf eine Verbindung, dann öffnet sich die Detailansicht der jeweiligen Fahrt.
Wie in Abbildung 4.8c zu sehen ist, werden im oberen Bereich wieder dieselben Informationen angezeigt, sowie Start- und Zielort. Im unteren Bereich werden für jeden Abschnitt die detaillierten Informationen dargestellt. Dabei werden, wenn der Nutzer auf den jeweiligen Pfeil nach unten klickt, die einzelnen Zwischenhalte präsentiert. Mit *zurück* kommt der Nutzer wieder in die Übersicht aller möglichen Verbindungen. Klickt der Nutzer hingegen auf den *Haken*, so ist die folgende Ansicht abhängig davon, ob der Nutzer eine einfache Fahrplanauskunft anfordert, oder eine Fahrscheinberatung, wobei im oberen Bereich jeweils die Anzahl reisender Personen dargestellt werden. In diesem Fall sieht der Nutzer, wie in Abbildung 4.8d zu erkennen, alle zu optimierenden und aktuell geplanten Fahrten. Dabei kann jede Fahrt über die Buttons editiert (e), gelöscht (l) oder dupliziert (d) werden. Über das ✚ in der Mitte der unteren Symbolleiste können weitere zu optimierende Fahrten hinzugefügt werden. Wählt der Nutzer hingegen ✗, so gelangt er zurück in das



(a) Formular **(b)** Mögliche Verbindungen **(c)** Detailansicht einer Fahrt

(d) Zu optimierende Fahrten **(e)** Zukünftige Fahrten **(f)** Fahrscheinübersicht

Abbildung 4.8: Entwurf des Layouts

Hauptmenü. Bestätigt der Nutzer die Fahrten, so werden die Fahrten optimiert und anschließend die Übersicht aller Fahrten angezeigt, vergleiche Abbildung 4.8e. In der gleichen Ansicht erhält der Nutzer für jede Fahrt die wesentlichen Informationen. Dabei werden jedoch für die optimierten Fahrten die benötigten Fahrscheine angezeigt. Klickt der Nutzer auf eine Fahrt, wird erneut die Detailansicht geöffnet. Jedoch werden bei optimierten Fahrten die Fahrscheine detailliert erläutert. In die Ansicht 4.8e wechselt der Nutzer ebenfalls, wenn er eine nicht zu optimierende Fahrt auswählt.
In Abbildung 4.8f ist die Fahrscheinübersicht zu sehen, der die benötigten Fahrscheine visualisiert werden. Diese werden dabei zunächst gruppiert. Für jede Fahrscheingruppe lassen sich die einzelnen Fahrscheine anzeigen. Für die einzelnen Fahrscheine lassen sich die zugeordneten Fahrten anzeigen. Tippt der Nutzer auf eine dieser Fahrten, öffnet sich erneut die Detailansicht. In dieser Ansicht lassen sich die Fahrten jedoch nicht löschen, editieren oder duplizieren.

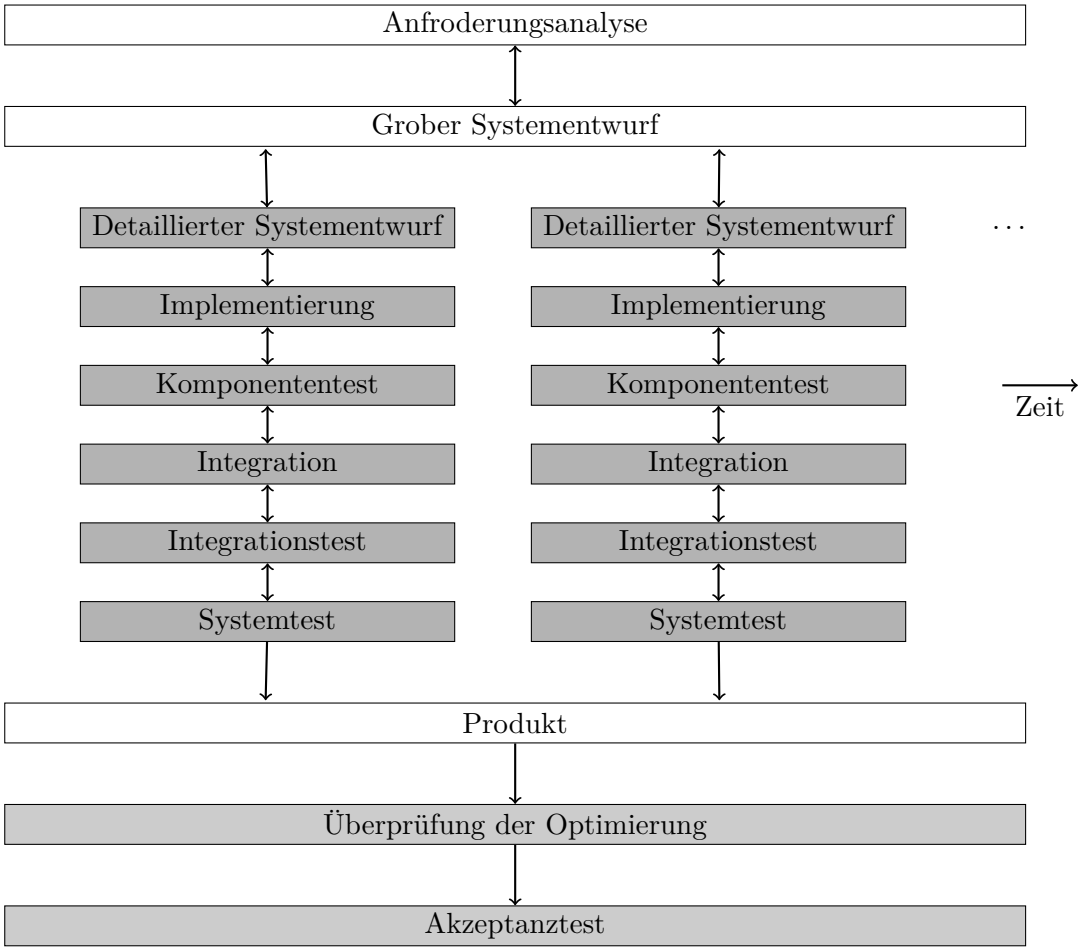


Abbildung 4.9: Entwicklungsmodell.

4.5 Geplantes Implementierungsvorgehen

Die Entwicklung der Anwendung soll nach einem angepassten inkrementellen Entwicklungsmodell, welches in Abbildung 4.9 zu sehen ist, erfolgen. Wie in dem inkrementellen Entwicklungsmodell werden zunächst die Anforderungen an das System festgelegt. Anschließend wird ein Prototyp entworfen, welcher schrittweise erweitert wird. In jeder Iteration werden die gleichen Schritte durchlaufen. Da sich bei jeder Erweiterung die Anforderungen und der Entwurf ändern, besteht zwischen den einzelnen Komponenten, dem Entwurf und den Anforderungen eine wechselseitige Beziehung [38].
Bis jetzt wurden in Kapitel 3.5 die Anforderungen, welche die Anwendung erfüllen soll, erarbeitet sowie zu Beginn des aktuellen Kapitels das System grob entworfen. Im Folgenden werden die einzelnen Komponenten, welche realisiert werden sollen, vorgestellt.
Die Entwicklung ist in folgender Reihenfolge geplant:

1. Zu Beginn soll das Layout für die einzelnen Ansichten umgesetzt werden.

2. Darauf folgt die Umsetzung der Aktivität *einzelne Fahrt hinzufügen*, aus Abschnitt 4.2.2. Dabei soll zunächst die Optimierung wie auch die Anzahl der

reisenden Personen entfallen und nur eine Fahrplanauskunft erfolgen.

3. Der vorherige Schritt kann nun erweitert werden, indem die Anzahl der reisenden Personen mit angegeben wird.

4. Wenn mehrere Fahrten geplant werden können, soll die Optimierung erfolgen. Diese soll in drei Schritte unterteilt werden:

   (a) Zuerst wird die Optimierung nur mit Einzelfahrscheinen realisiert.

   (b) Danach sollen die Fahrten nur mit Zeitfahrscheinen optimiert werden. Sollte sich kein passender Fahrschein für eine Fahrt finden lassen, so bleibt diese Fahrt zunächst ohne Tickets.

   (c) Zum Schluss ist die Kombination beider Ansätze geplant, damit keine Fahrt ohne Ticket verbleibt.

Wie in dem inkrementellen Entwicklungsmodell soll zunächst für jede Komponente ein detaillierter Entwurf erarbeitet werden. Dieser soll anschließend implementiert werden. Darauf folgend soll jeweils die einzelne Komponente getestet werden, bevor diese in das bestehende Gerüst integriert wird. Die Integration wird im Anschluss ebenfalls überprüft, bevor ein abschließender Systemtest durchgeführt wird. Eine Erläuterung der einzelnen Teststufen erfolgt in den folgenden Abschnitten.
Nach der erfolgreichen Realisierung aller Komponenten erfolgt ein Vergleich zwischen einer manuellen Optimierung und der Optimierung in der App. Dadurch soll überprüft werden, ob durch eine händische Optimierung der Kosten eine Ersparnis erreicht werden kann und wie hoch diese ausfällt.
Zum Schluss wird die gesamte Anwendung durch eine kleine Nutzergruppe überprüft. Die Gruppe erhält vorgegebene Testszenarien und dokumentiert dabei ihre Erfahrungen und Fehlermeldungen mit der App. Ziel ist es, zum einen die Anwendung auf verschiedenen Geräten zu testen. Zum anderen soll getestet werden, inwieweit die Nutzer die Anwendung akzeptieren, insbesondere wie sie die Bedienung der App finden und wie verständlich die einzelnen Schritte sind [39].
Am Ende der Realisierung einer einzelnen Komponente soll die Anwendung funktionsfähig sein. Ergeben sich aus der Realisierung späterer Komponenten Änderungen an einer früheren Komponente, so wird in die jeweilige Spalte gewechselt. Nach den Anpassungen müssen erneut die Teststufen durchlaufen werden. Sollte aus einem Test eine Änderung resultieren, so wird in der jeweiligen Spalte nach oben gewechselt.

Im Folgenden werden die drei Teststufen, die für jede Komponente durchlaufen werden, erläutert. Bei den Tests werden systematisch verschiedene Uhrzeiten und Tarifgebiete gewählt, um verschiedene Nutzereingaben abzudecken. Jedoch ist es nicht möglich, alle Spezialfälle zu überprüfen.

4.5.1 Komponententest

Mithilfe des Komponententests soll die Korrektheit kleinstmöglicher, sinnvoller Testobjekte gezeigt werden. Bei diesen Testobjekten kann es sich sowohl um Klassen als

auch um Methoden handeln. Bei diesem Testverfahren wird ein Testrahmen benötigt, welcher einzelne Komponenten simuliert [40]. In allen Komponenten mussten Fahrten simuliert werden. Eine Fahrt wird durch ein Objekt der Klasse `Trip` aus Öffi dargestellt. Da die Klasse zu komplex ist und Attribute von weiteren Klassen besitzt, vergleiche Abbildung 5.13, wurde die Klasse `TestTripItem` als Ersatz genutzt. Dabei benötigt jede Fahrt mindestens die folgenden Attribute: ID, Preisstufe und Anzahl reisender Personen.
Zusätzlich wird die Laufzeit einzelner Funktionen gemessen. Dazu wird die Systemzeit vor dem Aufruf und direkt danach genommen. Aus der Differenz dieser zwei Werte ergibt sich die Laufzeit in Millisekunden.
Wenn die Tests erfolgreich waren und keine weiteren Anpassungen mehr nötig sind, wird die einzelne Komponente in die bestehenden Komponenten eingebunden.

4.5.2 Integrationstest

Die Integration soll nach dem *Backbone* Ansatz erfolgen, da schon frühzeitig ein funktionsfähiges Produkt erwartet wird. Das Gerüst wird schrittweise durch die einzelnen Komponenten ergänzt und erhält dadurch weitere Funktionalitäten [40].
Wie in Abbildung 4.9 zu sehen ist, soll nach jeder Integration überprüft werden, ob die vorherigen Komponenten ihre volle Funktion beibehalten haben. Des Weiteren wird die Kombination der verschiedenen Komponenten überprüft. Als Platzhalter wird erneut die Klasse `TestTripItem` genutzt.

4.5.3 Systemtest

Durch die Systemtests wird überprüft, ob das Zusammenspiel der einzelnen Komponenten mit realen Daten funktioniert [40]. Des Weiteren wird bei diesen Tests die Laufzeit der Optimierung betrachtet. Dazu wird wie bei den Komponententests die Zeit in ms ausgegeben. Zusätzlich wird mit dieser Testart die Anzeige der Fahrscheine für den Nutzer überprüft. Dies ist notwendig, da Fahrscheine, um Platz zu sparen, wenn möglich gruppiert für den Nutzer angezeigt werden sollen, vergleiche Abbildung 4.8f. Die Anzeige muss zum einen korrekt sein, zum anderen aber auch für den Nutzer verständlich sein.
Für die Tests sind die einzelnen Fahrten nicht relevant. Dabei sind nur die Attribute Preisstufe und Anzahl reisender Personen wichtig. Bei Tests, die die Zeitfahrscheine betreffen, sind das Datum sowie die ungefähre Uhrzeit der Fahrt relevant.

Da der Server des VRRs nur Informationen für einen Zeitraum zur Verfügung stellt, müssen die Testdokumentationen so notiert sein, dass die Fälle leicht an das aktuelle Datum angepasst werden können.
Im Rahmen der Tests werden nur die realen Daten des VRRs betrachtet sowie die Optimierung nach den Regeln dieses Verkehrsverbundes. Für weitere Regionen müssen diese Tests unter Umständen angepasst und erneut durchgeführt werden.
Ein Abgleich zwischen der realisierten Anwendung und den Anforderungen aus der Analyse und dem Entwurf kann erst nach der Umsetzung aller Komponenten erfolgen. Der Vergleich ist in Kapitel 6 zu finden.

Kapitel 5

Organisatorische Umsetzung und Implementierung

In diesem Kapitel wird die Umsetzung der App dokumentiert. Dabei werden Schwierigkeiten und Probleme bei der Umsetzung der in Abschnitt 4.5 vorgestellten Schritte erläutert. Des Weiteren wird die Realisierung der Optimierung präsentiert. Diese einzelnen Schritte wurden, wie im vorhergegangenen Abschnitt beschrieben, erst detailliert entworfen, danach implementiert und abschließend getestet.

5.1 Planung von Fahrten

Weil die Optimierung in eine App eingebunden werden soll, muss in dieser zunächst die Möglichkeit realisiert werden, Fahrten zu planen. Dafür wird zunächst nur die Planung einer einzelnen Fahrt betrachtet.

5.1.1 Planung einer Fahrt

In diesem Schritt wurden zunächst die Layout-Skizzen aus Abschnitt 4.4 in `XML` realisiert. In der Detailansicht einer einzelnen Fahrt wurde darauf verzichtet, erneut Start und Ziel der Fahrt anzugeben, da die Informationen aus dem ersten und letzten Abschnitt entnehmbar sind. Anschließend wurde die Planung einer einzelnen Fahrt realisiert. Hierbei wurde auf den Entwurf aus Abbildung 4.8 zurückgegriffen. Dabei wurde zunächst die Speicherung von Fahrten ausgeschlossen. Des Weiteren konnte in diesem Schritt noch keine Anzahl reisender Nutzer festgelegt werden. Dieser Anwendungsfall wurde zuerst realisiert, da er als Basis für die Planung mehrerer Fahrten dient. In dem Formular, welches der Nutzer ausfüllt, werden das Datum der Fahrt mittels Date-Picker und die Ankunfts- beziehungsweise Abfahrtzeit mithilfe eines Time-Pickers gewählt. Dann wurde Öffi eingebunden, um auf die Funktion `suggestLocations` zugreifen zu können. Diese wird in der Auto-Vervollständigung der jeweiligen Orte genutzt.
Der Nutzer ist gezwungen, einen Vorschlag aus der Liste auszuwählen, um sicherzustellen, dass die Orte eindeutig sind. Damit die Verbindungssuche gestartet werden kann, muss der Nutzer Start-, Zielpunkt, Datum und Uhrzeit festlegen. Erst danach können in der nächsten Ansicht die möglichen Verbindungen dargestellt werden.

Anders als in den Anforderungen in Kapitel 3 geplant, macht der Nutzer alle Angaben in der gleichen Ansicht. Auch werden alte Suchanfragen nicht berücksichtigt. Des Weiteren kann der Nutzer keine Orte als Favoriten markieren. Da es sich dabei um Komfort-Funktionen handelt, verliert die Anwendung nicht an Funktionsumfang.
Für die Suche nach möglichen Verbindungen wird ein `AsyncTask`[1] gestartet. In diesem wird mit den Nutzereingaben eine Serveranfrage gestartet, welche passende Verbindungen ermittelt. Dabei wird auf die Klasse `QueryTripsResult` von Öffi zurückgegriffen. In jener ist eine Liste möglicher Verbindungen (`Trips`) enthalten. Über entsprechende Buttons hat der Nutzer die Möglichkeit, frühere oder spätere Verbindungen anzufordern oder seine Angaben zu ändern. In der Anzeige aller möglichen Verbindungen wurde, anders als geplant, für jede Linie und jedes Verkehrsmittel die gleiche Farbe gewählt. Um die Verkehrsmittel dennoch direkt zu unterscheiden, wurde vor jede Linie ein entsprechender Icon gewählt, siehe Abbildung B.4d. Anschließend wurde die Berücksichtigung von Einstellungen bei Anfragen umgesetzt. Der Nutzer kann an dieser Stelle wählen, welche Verkehrsmittel berücksichtigt werden sollen sowie ob alle Streckenabschnitte barrierefrei sein sollen. Dabei konnten nur die gleichen Einstellungsmöglichkeiten wie in Öffi umgesetzt werden und nicht wie geplant die weiteren Möglichkeiten aus der VRR App. Dies liegt an der Klasse `NetworkProvider`, welche in Öffi nur diese Optionen zur Verfügung stellt und nicht erweitert wurde. Wie geplant kann der Nutzer unter den drei Laufgeschwindigkeiten langsam, normal und schnell wählen.
Bis zu diesem Zeitpunkt wurden nur primitive Datentypen an die nächste Aktivität übergeben, dies wurde jedoch überarbeitet, sodass mehr Informationen erhalten bleiben. Durch diese Änderung ergab sich in der Detailansicht einer Fahrt die Möglichkeit, Zwischenhalte anzuzeigen. Um die Übersichtlichkeit der Ansicht zu erhalten, kann der Nutzer für jeden Streckenabschnitt einzeln entscheiden, ob die Zwischenhalte angezeigt werden sollen oder nicht.
Für Strecken, die individuell zurückgelegt werden, ist eine Navigation geplant worden. Der erste Ansatz war, das Attribut `List<Point> path` der Klasse `Individual` zu nutzen, jedoch war dieses stets `Null`, weshalb kein Kartenausschnitt mit der jeweiligen Route angezeigt werden kann. Stattdessen wird versucht, eine Navigationsanwendung auf dem Handy zu öffnen. Zwar ist es möglich, die Navigations API in die App einzubinden, dies ist aber kostenpflichtig[2].
Da an dieser Stelle der Umfang der Optimierung unbekannt war, wurde zunächst auf die Schritt-für-Schritt Übersicht verzichtet, weil es sich dabei nur um eine übersichtlichere Ansicht der Detailansicht handelt und sie durch die komponentenweise Entwicklung später hinzugefügt werden kann. Das Anzeigen von Umstiegen und Wartezeiten in der Fahrplanauskunft konnte nicht so realisiert werden wie geplant, weil die benötigten Informationen dazu nicht aus den `Leg`-Objekten ermittelt werden können. Diese Einschränkungen betreffen ebenso die Planung mehrerer Fahrten.

An dieser Stelle lassen sich die drei Testklassen nicht trennscharf unterscheiden. So wurden zu Beginn mithilfe von Dummy-Objekten die Ansichten gefüllt, um die

[1] Berechnung, die in einem Hintergrund Thread läuft und deren Ergebnis im User Interface Thread angezeigt werden soll, vergleiche [41].
[2] Mehr zur Preisgestaltung siehe [42].

Anzeige zu testen. Diese wurden anschließend gegen reale Daten ausgetauscht. Außer Verspätungen und Gleiswechseln werden keine weiteren Informationen zu Störungen angezeigt. Im Rahmen der Systemtests zeigte sich, dass Haltestellen, die außerhalb des VRR-Gebiets liegen, vorgeschlagen werden. Eine manuelle Filterung war im Zuge dieser Arbeit nicht möglich. Des Weiteren waren die Verbindungsdaten zum Teil fehlerhaft und unvollständig, so fehlten beispielsweise Angaben zum Gleis. Besonders auffällig war die uneinheitliche Benennung von Haltestellen. Dabei ist nicht einheitlich, ob der Name der Stadt ein Bestandteil des Haltestellennamens ist oder nicht. Als Lösung wird eine eigene Hilfsfunktion verwendet, die dies überprüft. Die Überprüfung führt jedoch zu Problemen in Sonderfällen wie dem Hauptbahnhof der Stadt Recklinghausen. Jener wird in der App jetzt mit dem Namen *Recklinghausen RE Hbf* angezeigt, obwohl RE die Abkürzung für Recklinghausen ist. Die Probleme sind im Rahmen der Arbeit nicht zu beheben, da sie auf die Datenbank des VRRs zurückzuführen sind.

5.1.2 Planung mehrerer Fahrten

Nachdem die Implementierung der Planung einer einzelnen Fahrt erfolgt ist, folgt die Planung mehrerer Fahrten. Dabei wurde die vorherige Komponente um die Anzahl an reisenden Personen erweitert. Außerdem können die Fahrten gespeichert werden. Die Speicherung erfolgt mittels `SharedPreferences`[3] und `GSON`[4], bei dem Serialisieren und Deserialisieren müssen jedoch die einzelnen Verbindungsabschnitte manuell betrachtet werden, da beim Laden nicht erkenntlich ist, ob es sich um ein Objekt der Klasse `Public` oder `Individual` handelt, vergleiche Abbildung 4.5. Die Verwaltung der gespeicherten Fahrten erfolgt in der neuen Klasse `MyTripList`. Die Klasse erweitert das Klassendiagramm in Abbildung 5.1 und besitzt Funktionen zum Laden und Speichern von Fahrten.
Die Unterscheidung, welche Fahrten aktuell betrachtet werden sollen, erfolgt über zwei verschiedene Datenpfade. Die Fahrten, die der Nutzer aktuell plant und optimiert werden sollen, werden unter einem anderen Datenpfad gespeichert und später geladen, als die Liste aller Fahrten.
Gespeichert werden die Objekte der Klasse `TripItem` (vergleiche Abbildung 5.1). Diese Klasse hat als Attribut zum einen die Verbindungsinformationen, zum anderen die Anzahl der reisenden Personen. Das Attribut `isComplete` gibt an, ob die Fahrt später optimiert werden soll oder nicht. Fahrten, bei denen das Attribut `true` ist, werden nicht optimiert und dienen nur zur Fahrplanauskunft.
Auch für die aktuelle Komponente lässt sich zwischen den einzelnen Testklassen nicht genau unterscheiden. Es wurde direkt mit realen Daten innerhalb der Anwendung gearbeitet. Einerseits wurden mehrere Fahrten, welche eine Anzahl an Nutzern erhalten haben, erstellt und anschließend überprüft, ob diese korrekt in den jeweiligen Listen angezeigt werden. Andererseits wurden Fahrten ohne reisende Nutzer erstellt und überprüft, ob diese korrekt angezeigt werden. Im Anschluss wurden

[3] Für mehr Informationen zur Speicherung mittels `SharedPreferences` siehe [43, 44].

[4] Eine Bibliothek zur Umwandlung von Java Objekten zu JSON[5]-Zeichenketten und umgekehrt, siehe [45].

[5] „JSON ist ein schlankes Datenaustauschformat, das [...] für Maschinen einfach zu parsen [...] und zu generieren ist “[46].

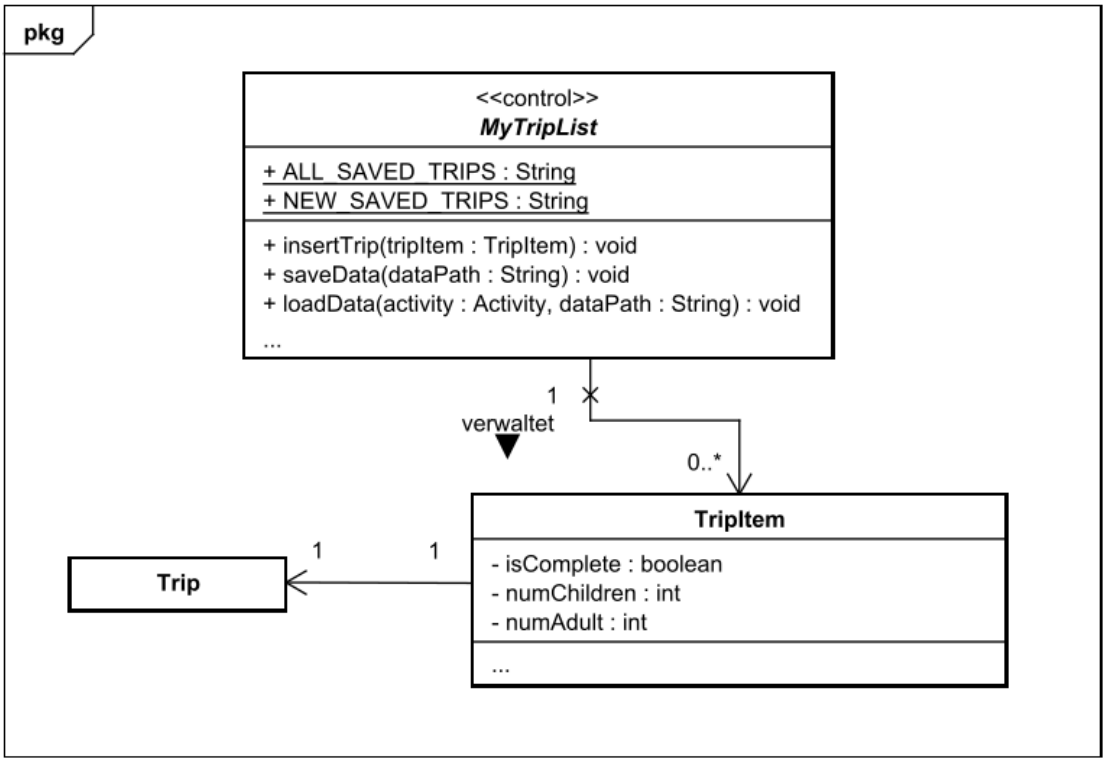


Abbildung 5.1: Die Klasse `TripItem` enthält die Informationen zur Verbindung und zu den reisenden Personen. Die Klasse `MyTripList` verwaltet die gespeicherten Fahrten.

sowohl Fahrten, die später optimiert werden sollen, aber auch einfache Fahrplanauskünfte durchgeführt und überprüft, ob die Listen an Fahrten jeweils die korrekten Fahrten anzeigen. Dabei traten keine Fehler auf.

5.2 Optimierung mit Einzelfahrscheinen

Im aktuellen Abschnitt wird die Implementierung der Optimierung mit Einzelfahrscheinen erläutert. Dafür wird, wie in dem Entwicklungsansatz geplant, zuerst auf den Entwurf dieser eingegangen und danach die Umsetzung und Testung erläutert.

5.2.1 Entwurf

Für die betrachteten Fahrscheine gilt, dass die Zeitabstände zwischen den einzelnen Fahrten irrelevant sind. Außerdem kann ein Fahrschein mit einer höheren Preisstufe für eine Fahrt mit einer geringen Preisstufe genutzt werden. Die Optimierung soll für die zwei Nutzerklassen *Erwachsene* und *Kinder* getrennt erfolgen.
Um die Optimierung besser nachvollziehen zu können, ist das Vorgehen in Abbildung 5.2 an Hand eines Beispiels skizziert.

Zu Beginn sind die Fahrten, wie in Abbildung 5.2a, nach der Abfahrtszeit aufsteigend sortiert. Wie im Beispiel zu sehen ist, werden die Fahrten zuerst aufsteigend

D	B	C	D	D

(a) Fahrten nach ihren Abfahrtszeiten sortiert.

B	C	D	D	D

(b) Fahrten nach Preisstufen aufsteigend sortieren.

B	C	D	D	D
6,00	18,80	34,10	49,80	?

(c) Optimierung der vierten Fahrt abgeschlossen.

B	C	D	D	D
6,00	18,80	34,10	49,80	63,10

(d) Optimierung der fünften Fahrt abgeschlossen.

Abbildung 5.2: Beispiel für die rekursive Optimierung mit Einzelfahrscheinen.

nach der Preisstufe sortiert (Abbildung 5.2b). Dies muss vor der Optimierung geschehen.
In der Optimierung wird bis zur jeweils betrachteten Fahrt an Position i der minimale Fahrschein ermittelt. Um für alle Fahrten die minimale Fahrscheinkombination zu ermitteln, muss über alle Fahrten iteriert werden. Für die aktuell betrachtete Fahrt wird nun das Ticket mit den minimalen Kosten gesucht, indem die Rekursionsformel (5.1) angewendet wird:

$$p(i) := \begin{cases} \min\{p(i-1) + E(f_i), p(i-4) + V(f_i), p(i-10) + Z(f_i)\} & \text{für } i \geq 1 \\ 0 & \text{für } i < 1 \end{cases} \tag{5.1}$$

Dabei bezeichnet f_i die Preisstufe der i-ten Fahrt und E, V und Z den Preis des jeweiligen Tickets. Diese Formel lässt sich auf beliebige Fahrscheine erweitern.

Für das Beispiel ergibt sich, dass zwischen Abbildung 5.2c und 5.2d die folgenden Betrachtungen erfolgen:

$$\begin{aligned} Einzelticket &: 49,80€ + 15,70€ = 65,50€ \\ 4erTicket &: 6,00€ + 57,10€ = 63,10€ \\ 10erTicket &: 0,00€ + 104,85€ = 104,85€ \end{aligned}$$

Aus dem letzten ermittelten Fahrschein lassen sich alle benötigten Fahrscheine rekursiv ermitteln.

Der Algorithmus findet immer die preisgünstigste Fahrscheinkombination für die angegebenen Fahrten, da es sich um einen rekursiven Ansatz handelt, der alle Möglichkeiten probiert und anschließend das Minimum wählt.

5.2.2 Realisierung

Implementierung

Zunächst wurde der Entwurf außerhalb der bereits realisierten Anwendung implementiert. Um eine bessere Laufzeit zu erzielen, wurde der rekursive Ansatz dynamisch implementiert. Da am Ende nicht nur der minimale Preis, sondern zusätzlich das entsprechende Ticket benötigt wird, wird für jede Fahrt neben dem Preis auch

B	C	D	D	D
6,00	18,80	34,10	49,80	?
←	←	←	←	

(a) Optimierung der vierten Fahrt abgeschlossen.

B	C	D	D	D
6,00	18,80	34,10	49,80	63,10
←	← ←	←	←	

(b) Optimierung der fünften Fahrt abgeschlossen.

VT D	D D D C
ET B	B

(c) Fahrscheine und ihre zugewiesenen Fahrten.

B	C	D	D	D
ET	VT	VT	VT	VT
B	D	D	D	D

(d) Fahrten und ihre zugewiesenen Fahrscheine.

Abbildung 5.3: Beispiel für die dynamische Optimierung mit Einzelfahrscheinen.

der Fahrschein und dessen Preisstufe gemerkt. So müssen diese später nicht erneut bestimmt werden.

Für die Umsetzung des dynamischen Ansatzes wurde die Klasse `TicketInformationHolder` hinzugefügt, welche die Informationen über die Kosten und den Fahrschein enthält. Zusätzlich besitzt die Klasse das Attribut `previous`, über diese Eigenschaft kann abschließend die Liste der benötigten Fahrscheine ermittelt werden (Abbildung 5.4).
Wie in Abschnitt 4.5.1 erläutert, werden für die Erstellung von Objekten der Klasse `Trip` viele weitere Objekte benötigt, weshalb stattdessen zunächst auf Platzhalter der Klasse `TestTripItem` zurückgegriffen wurde. Zuerst wurde für jede Fahrt nur ein Erwachsener angenommen.
Um die Fahrten nach ihren jeweiligen Preisstufen zu sortieren, wurde ein Comparator für die Klasse `TripItem` genutzt.

Zur Visualisierung des dynamischen Ansatzes wird das obige Beispiel erneut betrachtet. Um keine Fehler zu erhalten, wenn auf einen Index kleiner null zugegriffen wird, wurde die Länge des `TicketOptimisationHolder`-Arrays m gewählt, wobei

$$m := \texttt{length(trips)} + n$$

ist. n ist dabei die maximale Anzahl an Fahrten, die mit den Einzelfahrscheinen möglich ist. Dabei werden die ersten n Einträge ohne Vorgänger, Ticket, Preisstufe und einem Preis von 0 € initialisiert. Dies ist in Abbildung 5.3 nicht zu sehen, damit die Grafik übersichtlich ist. In der untersten Zeile der Tabelle ist jeweils der Zeiger auf das vorherige Element zu sehen. Betrachtet man nun den Schritt von Abbildung 5.3a nach Abbildung 5.3b, ist aus dem vorherigen Abschnitt bekannt, dass ein 4erTicket das optimale Ticket ist. Deshalb wird in Abbildung 5.3b als Vorgänger die erste Fahrt gewählt.

Nachdem für jede Fahrt das beste Ticket ermittelt wurde, werden den Fahrscheinen die Fahrten zugewiesen, ebenso andersrum. Dafür wird die Liste an Fahrten und das Array an Tickets von hinten durchlaufen. Im Beispiel heißt das, dass zuerst das 4erTicket betrachtet wird. Diesem werden nun die letzten vier Fahrten zugewiesen. Anschließend wird sein Vorgänger, das Einzelticket der Preisstufe B betrachtet. Dem Fahrschein wird nun die nächste Fahrt, in diesem Fall auch die letzte Fahrt,

zugewiesen. Dies ist in Abbildung 5.3c zu sehen. In Abbildung 5.3d hingegen ist die entsprechende Zuordnung der Fahrscheine an die Fahrten zu sehen.
Bei der Ermittlung der benötigten Fahrscheine wird zuerst das letzte Ticket in eine Liste eingefügt und dann auf dessen Vorgänger zugegriffen. Dieser wird ebenfalls in die Liste eingefügt, dies wird solange wiederholt, wie das Ticket nicht `Null` ist. Danach wird die Liste der benötigten Fahrscheine gespeichert. Die Serialisierung der Fahrscheine erfolgt manuell, da die Klasse `Ticket` abstrakt ist. Die Speicherung erfolgt wieder mittels `GSON` und `SharedPreferences`. Das Laden und Speichern der Fahrscheine erfolgt in der Controller-Klasse `AllTickets`. Diese benötigt nur einen String zum Speichern, da es nur eine Fahrscheinliste gibt (Abbildung 5.4).
Als nächste Erweiterung wurde die Anzahl der reisenden Erwachsenen erhöht. Dabei zeigte sich, dass der Entwurf überarbeitet werden musste, da die Fahrten jetzt mehrere Fahrscheine benötigen. Statt eines einzelnen Tickets wurde nun eine Liste von Fahrscheinen genommen. Diese Anpassung gilt ebenso für die Preisstufen der Fahrscheine. Wenn mehr als eine Person reist, wird die jeweilige Fahrt entsprechend oft in die Liste der zu optimierenden Fahrten eingefügt. Abschließend wurden neben Erwachsenen auch Kinder berücksichtigt. Dadurch ergab sich, dass statt den Listen an Fahrscheinen und deren Preisstufe jeweils eine Liste von Listen benötigt wird, um die entsprechenden Fahrscheine zu unterscheiden. Diese Anpassungen trafen gleichermaßen auf die Klasse `AllTickets` zu. In dieser werden nun ebenfalls Listen von Fahrscheinlisten sowie den jeweiligen Preisstufenlisten genutzt. Die Änderungen und Korrekturen sind in Abbildung 5.4 zu sehen.

Testen

Bereits während der Implementierung wurde mittels Komponententests überprüft, ob die jeweiligen Ergebnisse der Optimierung korrekt sind. Dabei bedeutet korrekt, dass jede Person ein gültiges Ticket besitzt und einem Fahrschein nicht mehr Fahrten zugewiesen werden als erlaubt. Dafür wurden Fahrten, die sowohl verschiedenen Preisstufen als auch unterschiedlich viele Reisende besitzen, erstellt und anschließend das Ergebnis überprüft.
Dabei zeigte sich, dass für jeden Fahrschein und jede Preisstufe die korrekte minimale Anzahl an Fahrten, ab dem sich das Ticket lohnt, berücksichtigt wird. Gegebenenfalls werden dabei Fahrten mit einer niedrigeren Preisstufe durch einen Fahrschein mit höherer Preisstufe abgedeckt.
Dann wurde die Optimierung in die App eingebunden. Mittels Integrationstest wurde überprüft, ob die Planung von Fahrten weiterhin funktioniert und die Übertragung der Funktionen ohne Fehler verlaufen war. Abschließend wurde mittels Systemtest überprüft, ob die Optimierung für reale Daten funktioniert. Dafür wurden zunächst in der App mehrere Fahrten geplant und optimiert. So wurde beispielsweise eine Fahrt mit zwei Erwachsenen und einem Kind von *Dortmund Hauptbahnhof* nach *Bochum Ehrenfeld S* geplant sowie eine Rückfahrt zwei Tage später mit nur einem Erwachsenen. Als Ergebnis wurden drei Einzeltickets der Preisstufe B für die Erwachsenen und zwei Einzeltickets der Preisstufe D für das Kind erwartet. Überprüft wurde das Ergebnis in der Detailansicht der Fahrten, ob jeweils die richtigen Fahrscheine zugewiesen wurden. In gleichem Zuge wurde in der Fahrscheinliste die Zuordnung der richtigen Fahrten kontrolliert. Dabei traten keine Fehler auf. Es

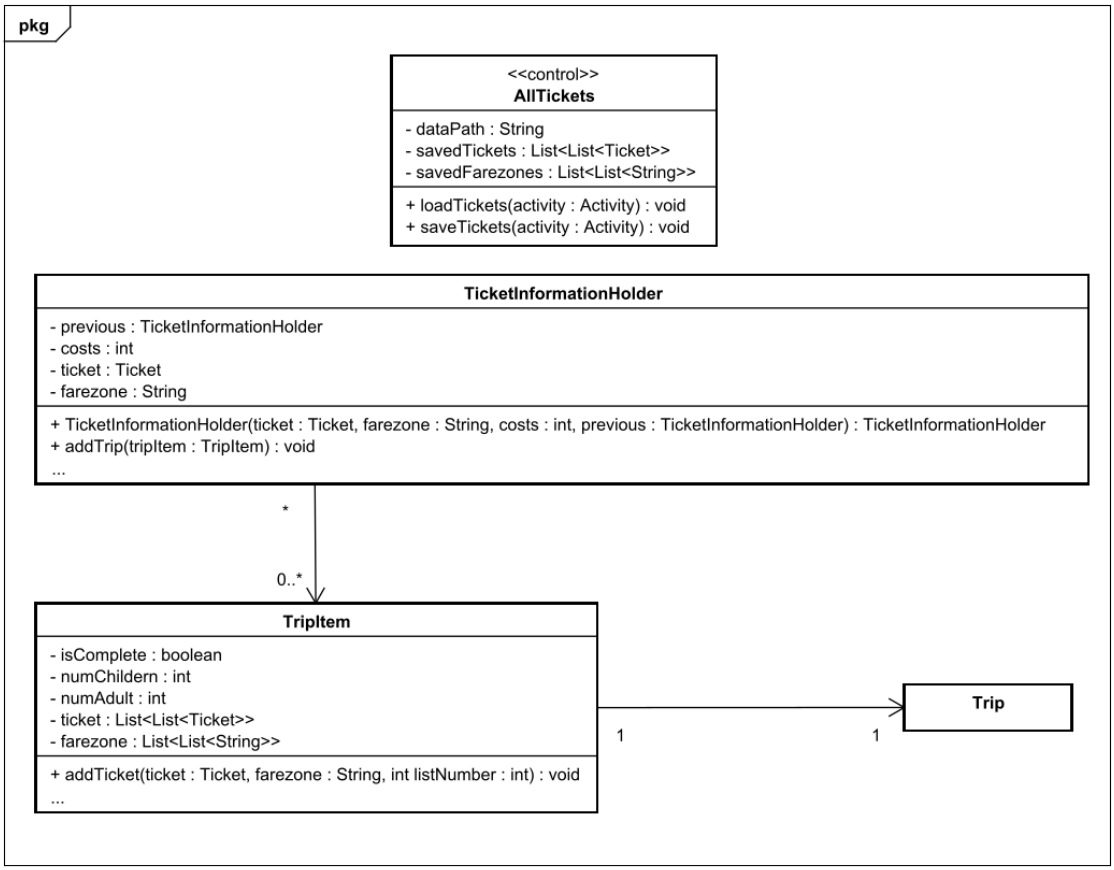


Abbildung 5.4: Erweiterung des Klassendiagramms um die Klassen `TicketOptimisationHolder` und `AllTickets` und Anpassung der Klasse `TripItem`.

zeigte sich jedoch, dass bei Fahrten, die optimiert werden sollen, der Nutzer keinen Zwischenhalt angeben darf, weil die Preisstufe der Fahrt in diesen Fällen nicht angegeben wird. Die Preisstufe ist jedoch für die Optimierung eine wichtige Komponente, die zwingend benötigt wird.

5.3 Generalisierung

Anders als in Abschnitt 4.5 geplant, wurde als dritter Schritt nicht die Optimierung mit Zeitfahrscheinen durchgeführt, sondern der Ansatz insgesamt verallgemeinert, sodass dieser auf andere Verkehrsverbünde mit einer ähnlichen Struktur übertragen werden kann. Des Weiteren wurde die Datenstruktur, in welcher die Fahrscheine gespeichert werden, geändert.
Dafür wurde zunächst das Klassendiagramm erweitert und überarbeitet. Damit die App leicht an andere Verkehrsverbünde angepasst werden kann, wird die Klasse `MyProvider` genutzt. Wie in Abbildung 5.5 zu sehen ist, gibt es Funktionen, die abhängig vom Verkehrsverbund sind und in der jeweiligen Klasse implementiert werden müssen.
Die Klasse `TicketToBuy` enthält hingegen wie die Klasse

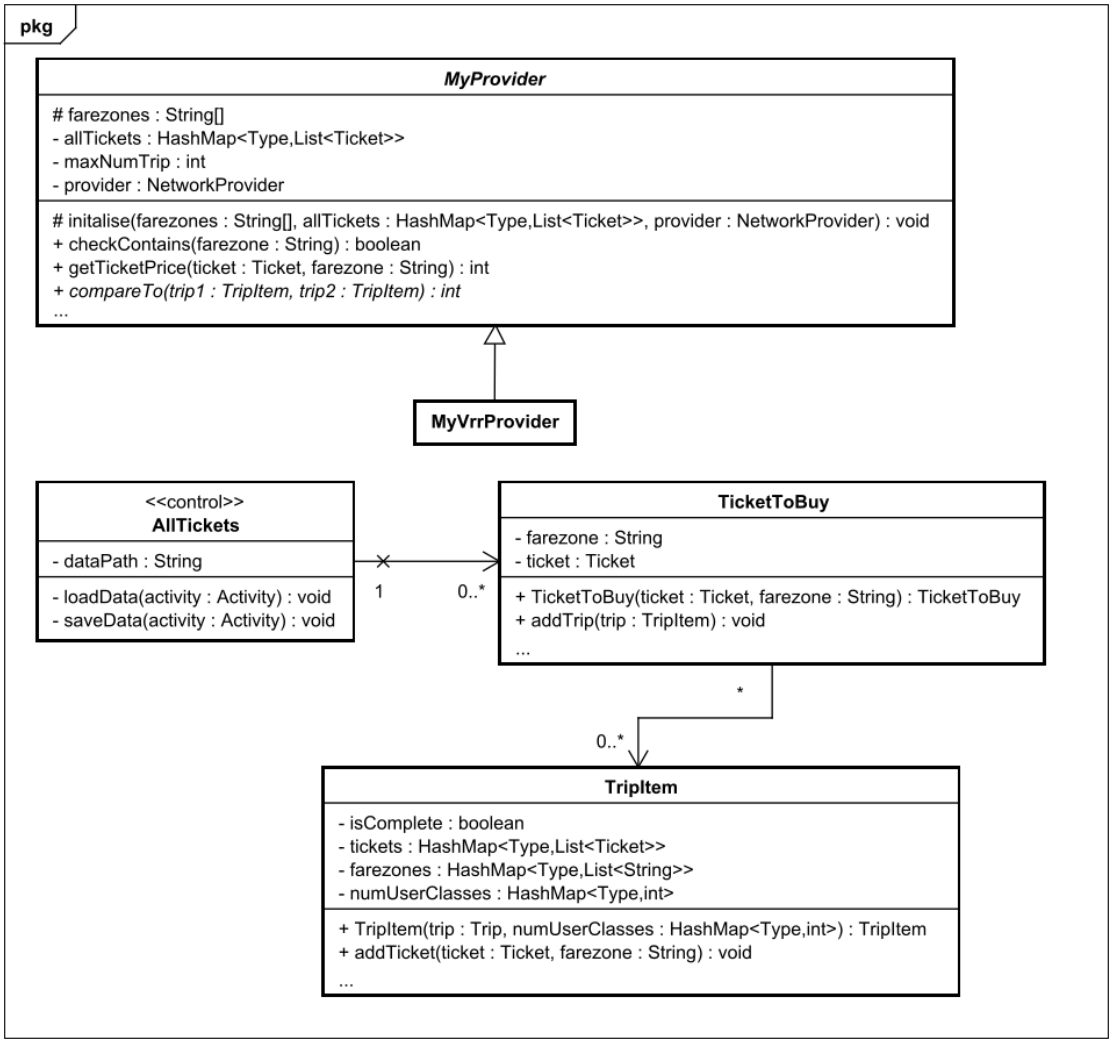


Abbildung 5.5: Erweiterung des Klassendiagramms um die abstrakte Klasse `MyProvider` und die Klasse `TicketToBuy` und Anpassung der Klasse `TripItem`.

`TicketOptimisationHolder` das Ticket, die Preisstufe des Tickets und die Liste an zugeordneten Fahrten. Dadurch ergab sich, dass in der Controller-Klasse `AllTickets` nicht mehr die Objekte der Klasse `Ticket` gespeichert werden, sondern die der Klasse `TicketToBuy`. Um die Fahrscheine der jeweils richtigen Nutzerklasse zuzuweisen, wurde, wie in Abbildung 5.5 zu sehen, auf eine HashMap gewechselt.
Zusätzlich werden die reisenden Personen nicht mehr als `Integer` mit entsprechendem Namen gespeichert, sondern sind nun in einer HashMap gespeichert, mit dem jeweiligen Typ als Schlüssel. Als Typ wird das innere Enum `Type` der Klasse `Fare` aus Öffi genutzt. Dadurch lässt sich die Optimierung auf weitere Nutzerklassen übertragen und so auch für weitere Fahrscheinklassen verwenden. Für weitere Nutzerklassen muss nur das Layout angepasst werden. Mittels `Type` wird für jeden Fahrschein festgelegt, für welche Nutzerklasse er gilt.
Die im Klassendiagramm (Abbildung 5.5) erkennbaren Änderungen wurden anschließend implementiert. Danach wurden die Tests der zwei bestehenden Kom-

ponenten erneut durchgeführt, wodurch sichergestellt wurde, dass die Änderungen erfolgreich in die Anwendung eingebunden werden konnten.

5.4 Verwendung alter Fahrscheine

Ein weiterer Zwischenschritt, bevor die Optimierung mit Zeitfahrscheinen umgesetzt wird, ist die Verwendung von freien Fahrten einzelner Fahrscheine. Diese Fahrten können zwei Ursachen haben:

- Durch die Optimierung: Es ist günstiger, einen größeren Einzelfahrschein zu kaufen als mehrere kleine Fahrscheine. Im Falle des VRRs ist es beispielsweise günstiger, ein 10erTicket Preisstufe D zu kaufen, als zwei 4erTickets der gleichen Preisstufe.
- Durch den Nutzer verursacht: Löschen oder Editieren von Fahrten.

  Ein Beispiel dafür ist die Planung von vier Fahrten der gleichen Preisstufe mit jeweils einer Person an vier aufeinander folgenden Tagen. Diese vier Fahrten sind einem 4erTicket zugewiesen. Löscht der Nutzer an Tag 2 jedoch die vierte Fahrt, hat dieses Ticket eine ungenutzte Fahrt. Oder aber der Nutzer editiert eine der zukünftigen Fahrten, wodurch die Fahrt eine höhere Preisstufe erhält. Dann benötigt diese Fahrt einerseits einen neuen Fahrschein, andererseits besitzt der vorherige Fahrschein eine freie Fahrt.

Um Kosten einzusparen, sollen die bereits bezahlten Fahrten dieser Fahrscheine ebenfalls genutzt werden. Da die Verwendung alter Fahrscheine somit sehr eng mit der Entstehung dieser zusammenhängt, werden in diesem Schritt beide Komponenten zusammen betrachtet.

5.4.1 Entwurf

Im Zuge des Entwurfs der Komfortfunktionen Löschen und Editieren soll die Funktion kopieren entworfen werden. Klickt der Nutzer auf Kopieren, so soll sich das Formular öffnen, mit dem gleichen Start und Zielpunkt, gegebenenfalls dem Zwischenhalt oder der Anzahl reisender Personen. Dabei sollen weder das Datum noch der Zeitpunkt bereits ausgefüllt sein. Beim Editieren hingegen sollen sowohl die Uhrzeit als auch das Datum ebenfalls ausgefüllt sein. Bricht der Nutzer das Editieren ab, soll die ursprüngliche Fahrt wieder gespeichert werden. Der geplante Ablauf aus dem Aktivitätsdiagramm in Abbildung 4.4 soll dabei umgesetzt werden. Beim Löschen soll die Fahrt nicht mehr gespeichert sein und nicht mehr angezeigt werden. Ebenso soll die Fahrt keinem Fahrschein mehr zugewiesen sein.
Sollte aus einem der oben genannten Gründe ein Fahrschein noch freie Fahrten besitzen, so ist bei der Verwendung dieser wichtig, dass die Preisstufe der neuen Fahrt kleiner oder gleich der Preisstufe des Fahrscheins ist. Um jedoch keine Kapazitäten zu verschwenden, wird versucht, die kleinstmögliche Ticketpreisstufe zu verwenden. Deshalb werden die Fahrscheine von unten durchlaufen, die Fahrten jedoch von oben. Ein illustriertes Beispiel des Vorgehens ist in Abbildung 5.6 zu sehen. Zuerst wird der zweiten Fahrt das erste Ticket zugewiesen, da dies die erste Fahrt mit

einer Preisstufe kleiner der Fahrscheinpreisstufe ist. Anschließend wird das zweite Ticket für die erste Fahrt genutzt, da die Preisstufe C höher als die Preisstufe B ist. Abschließend wird das verbleibende Ticket für die letzte Fahrt verwendet.

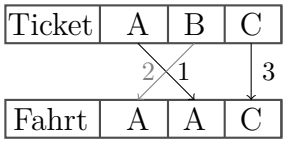


Abbildung 5.6: Beispiel für die Verwendung alter Einzelfahrscheine.

5.4.2 Realisierung

Implementierung

Die Implementierung der drei Funktionen sowie der Optimierung mit alten Fahrscheinen erfolgte jeweils direkt in der Anwendung.
Falls eine Fahrt optimiert wurde, muss, falls der Nutzer die Fahrt editieren oder löschen will, eine Anpassung der Fahrscheine erfolgen. In beiden Fällen erscheint zunächst eine Sicherheitsabfrage. Wenn der Nutzer eine Fahrt löscht, so wird diese aus der Liste entfernt und sollte es sich um eine optimierte Fahrt handeln und im Anschluss erneut optimiert. Ist die Fahrt dabei einem Fahrschein zugewiesen, von dem mindestens eine Fahrt in der Vergangenheit liegt, so wird der Nutzer darüber informiert, dass die Kosten dadurch nicht mehr minimal sein könnten. An dieser Stelle hat der Nutzer erneut die Möglichkeit abzubrechen. Löscht der Nutzer dennoch die Fahrt, wird die Anzahl freier Fahrten des Fahrscheins erhöht und die Fahrt aus der Liste der zugeordneten Fahrten des Fahrscheins entfernt. Liegen hingegen alle Fahrten eines Fahrscheins in der Zukunft, werden alle Fahrten von zukünftigen Fahrscheinen erneut optimiert. Daraus resultiert, dass der Nutzer immer nur die Fahrscheine für die nächste Fahrt kaufen sollte.
Anders als ursprünglich geplant, editiert der Nutzer nicht erst die Fahrt und wird anschließend über Veränderungen der Fahrscheine informiert, sondern er erhält vorher eine Warnung, dass die Kosten unter Umständen nicht mehr minimal sind. Beim Editieren wird die Aktivität Löschen ausgeführt und danach wie geplant das ausgefüllte Formular geöffnet. Um die Kosten zu minimieren, sollen bei der Optimierung dann die freien Fahrten von alten Fahrscheinen verwendet werden.

Bricht der Nutzer hingegen das Editieren ab, so wird die alte Verbindung wieder in die Liste der zukünftigen Fahrten eingefügt und die Fahrten erneut optimiert. Durch die Verwendung von alten Fahrscheinen soll die Fahrt die gleichen Fahrscheine erneut zugewiesen bekommen.
Bei der Integration wurde darauf geachtet, dass zuerst die alten Fahrscheine und danach die neuen Fahrscheine genutzt werden. Die Fahrten eines Fahrscheins, von dem mindestens eine Fahrt in der Vergangenheit liegt, werden nicht erneut optimiert. Die anderen Fahrten hingegen werden neu optimiert und die noch nicht benutzten Fahrscheine aus der Liste der gespeicherten Fahrscheine entfernt. Auch aus diesem Grund sollte der Nutzer jeweils nur die Fahrscheine für die nächste Fahrt kaufen und dies erst kurz vor Antritt der Fahrt. Sollten die angefangenen Fahrscheine noch freie

Fahrten haben, so werden diese nicht nur in der Liste der gespeicherten Fahrten erhalten, sondern für die Optimierung verwendet. Für die Optimierung mit alten Fahrscheinen konnte der Entwurf so realisiert werden, wie geplant.

Testen

Da die Implementierung jeweils direkt in der Anwendung erfolgte, fielen die drei Teststufen erneut zusammen.
Um eine der drei Aktionen löschen, editieren oder kopieren auszuführen, müssen die Fahrten zunächst geplant werden. Dadurch wurde sichergestellt, dass die vorher realisierten Komponenten ihre Funktionalität beibehielten.
Bei den Tests zeigte sich, dass beim Löschen von Fahrten eine ID für die Fahrscheine benötigt wird, um sicher zu stellen, dass die richtigen Fahrscheine Fahrten freigegeben bekommen, beziehungsweise gelöscht werden. Um diese Informationen in der jeweiligen Fahrt speichern zu können, wurde die Klasse `TripTicketInformationHolder` erstellt. Damit nicht bei jedem Anzeigen der Verbindung die Fahrscheine geladen werden müssen, enthält diese neben der ID des Tickets auch den Fahrschein sowie dessen Preisstufe.
Weitere Tests zeigten, dass beim Laden der Fahrscheine die Fahrten nicht mehr so oft in der Liste der zugeordneten Verbindungen enthalten waren, wie sie ursprünglich zugeordnet wurden. Zur Lösung dieses Problems wurde die Klasse `TripQuantity` hinzugefügt. Als Attribut besitzt die Klasse eine Fahrt und die Anzahl, wie häufig diese dem Ticket zugeordnet wurde. Um die Verwendung von alten Fahrscheinen

Systemzeit: Datum: x Uhrzeit: 13^{00}						
1	Tag x	Ab	10^{00}	Duisburg Hbf	Bochum Hbf	2 Erwachsener
2	Tag x	Ab	16^{00}	Bochum Riemke	Duisburg Hbf	2 Erwachsene
3	Tag x + 1	Ab	11^{00}	Dortmund Hbf	Hattingen Mitte	1 Erwachsener
4	Tag x + 1	Ab	19^{00}	Dorsten Bf	Dortmund Hbf	2 Erwachsene
5	Tag x + 4	Ab	14^{00}	Dorsten Bf	Hattingen Mitte	1 Erwachsener
Systemzeit: Datum: $x + 1$ Uhrzeit: 13^{00}						
Löschen von Fahrt 4						
6	Tag x + 2	Ab	15^{00}	Düsseldorf Hbf	Dortmund Hbf	1 Erwachsener
7	Tag x + 3	An	23^{00}	Marl Mitte	Bochum Hbf	3 Erwachsene
Lösung:						
1x 10erTicket Preisstufe C – Fahrt 1, 2, 3, 5, 7 1x Einzelticket Preisstufe D – Fahrt 6						

Tabelle 5.1: Beispiel für einen Systemtest, um die Verwendung von alten Fahrscheinen zu überprüfen.

zu überprüfen, wurden vor allem Systemtests durchgeführt. Dazu wurden jeweils Fahrten geplant und optimiert. Dadurch wurde sichergestellt, dass die vorherigen

Komponenten durch die Änderungen unverändert funktionieren. Um die Simulation nicht über mehrere Stunden zu testen, wurde einfachheitshalber die Systemzeit verändert. Die Zeit wurde so gewählt, dass diese zwischen zwei Fahrten fällt.

Anschließend wurden die zukünftigen Fahrten gelöscht oder editiert. Ein Beispiel dafür ist in Tabelle 5.1 zu sehen: Warum in der Tabelle keine konkreten Fahrten angegeben sind, wurde bereits in Abschnitt 4.5.3 erklärt. In diesem Fall würden die fünf zuerst geplanten Fahrten ein 10erTicket der Preisstufe C zugewiesen bekommen, da die Strecken jeweils diese Preisstufe besitzen. Außerdem ist dieses Ticket günstiger als zwei 4erTickets der Preisstufe C, das Ticket hat also zwei freie Fahrten. Wenn die Uhrzeit auf den nächsten Tag auf 13 Uhr gestellt wird, so liegen die ersten drei Fahrten in der Vergangenheit. Durch das Löschen der vierten Fahrt werden auf dem Ticket noch zwei weitere Fahrten frei. Insgesamt sind also vier Fahrten unbenutzt. Bei dem Hinzufügen der neuen Fahrten sollen diese berücksichtigt werden. Dabei muss jedoch beachtet werden, dass die sechste Fahrt die höhere Preisstufe D und die letzte Fahrt die niedrigere Preisstufe B besitzt. Da die sechste Fahrt eine höhere Preisstufe besitzt als der bereits gekaufte Fahrschein, wird für diese Fahrt ein neuer Fahrschein benötigt. Für die siebte Fahrt mit der niedrigeren Preisstufe kann hingegen der alte Fahrschein verwendet werden. Da diese Fahrt jedoch drei reisende Erwachsene zugewiesen hat, verbleibt auf dem Fahrschein nur noch eine freie Fahrt. Bei der Überprüfung traten keine Fehler auf.
Die neuen Klassen sowie wesentliche Änderungen in den bereits vorhanden Klassen sind in Abbildung 5.7 zu sehen.

5.5 Optimierung mit Zeitfahrscheinen

In diesem Abschnitt erfolgt die Optimierung mit Zeitfahrscheinen, welche ursprünglich als sechster Schritt geplant war. Dafür wird zunächst auf den Entwurf eingegangen. Dann wird die Implementierung vorgestellt. Abschließend wird die Generalisierbarkeit des Ansatzes diskutiert sowie Ergebnisse der Tests.

5.5.1 Entwurf

Da wie in Abschnitt 2.1.1 erläutert, jede Preisstufe einen anderen Umfang und Struktur hat, wird jede Stufe separat betrachtet. Die Optimierung für die jeweiligen Preisstufen werden im Folgenden erläutert.

Preisstufe D

Zuerst wurde die Optimierung für die Preisstufe D entworfen. Mit einem Ticket dieser Preisstufe kann, wie in dem oben genannten Abschnitt erläutert, das gesamte Gebiet befahren werden, deshalb müssen keine einzelnen Geltungsbereiche betrachtet werden, und die Optimierung gestaltet sich am einfachsten. Das Vorgehen der Optimierung als Pseudo-Code ist in Algorithmus 1 zu sehen.

Bei der Optimierung werden nur die Fahrten der Preisstufe D betrachtet, diese werden in der Funktion `collectTrips` in eine eigene Liste eingefügt. Wenn es keine

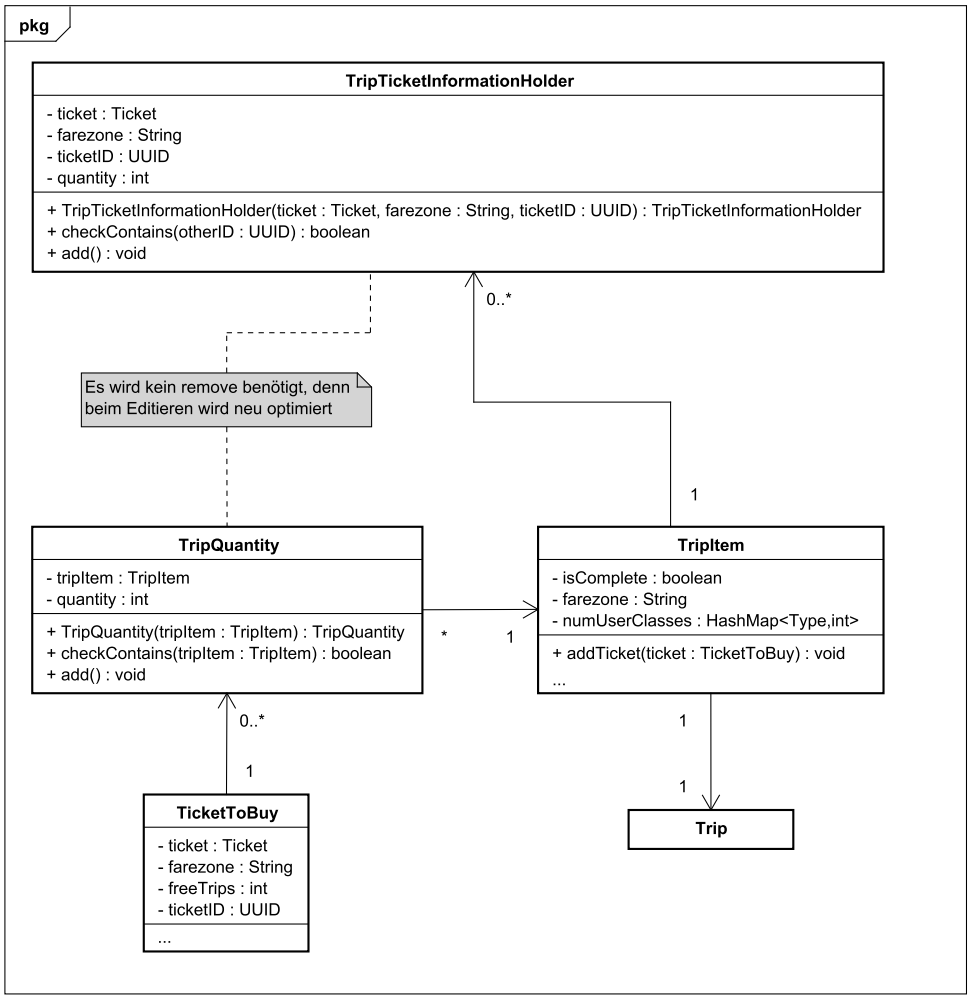


Abbildung 5.7: Anpassung des Klassendiagramms für die Verwendung alter Fahrscheine.

Fahrten der Preisstufe gibt, wird abgebrochen, da es keine zu optimierenden Fahrten gibt.

Bei diesem Ansatz werden alle übergebenen Fahrscheine ausprobiert. Im Falle des VRRs müssen, wie in Tabelle 2.2 zu sehen, zwei Tickets ausgeschlossen werden. Wie in Abschnitt 3.3.4 beschrieben, besitzt jeder Zeitfahrschein eine minimale Anzahl an Einzelfahrten, gegenüber der sich der Kauf lohnt. Dieser wird jeweils für das aktuelle Ticket in Zeile 11 verglichen. Dieser Schwellenwert muss für jede Preisstufe berechnet werden und erweitert die Klasse `TimeTicket`. Dabei werden die Zeiten der Fahrten zunächst ignoriert, um die Laufzeit zu optimieren.
Wenn insgesamt mehr Fahrten vorhanden sind, als der Schwellenwert des Tickets vorgibt, wird anschließend das beste Intervall für das aktuelle Ticket gesucht. Dies geschieht in der Funktion `bestTicketIntervall`. Die Funktion gibt zum einen den Startindex des besten Ticketintervalls zurück (`maxIndex`) und zum anderen die Anzahl der Fahrten, die ab diesem Index mit dem aktuell betrachteten Ticket unternommen werden können (`maxNumTrips`), vergleiche Zeile 12.
Eine Erläuterung der Funktionsweise dieser Funktion erfolgt in Abschnitt 5.5.2.

Wenn ein Intervall an Fahrten gefunden wurde, dass mehr Fahrten als benötigt enthält, wird ein neues Objekt der Klasse `TicketToBuy` erzeugt, welches das aktuelle Ticket der Preisstufe D enthält. Im Anschluss werden die Fahrten aus dem Intervall $[maxIndex, maxIndex + maxNumTrips]$ dem Ticket zugewiesen (Zeile 16). Danach müssen diese Fahrten aus den beiden Listen mit den zu optimierenden Fahrten entfernt werden (Zeile 17). Dann werden erneut der maximale Startindex sowie die maximale Anzahl an Fahrten ermittelt.
Wenn $maxNumTrips < minNumTrips$ gilt, wird überprüft, ob bereits mehr als ein Ticket gesammelt wurde, falls ja, wird mit der Funktion `sumUpTickets` in Zeile 23 überprüft, ob sich die gesammelten Fahrscheine preisgünstiger zusammenfassen lassen. Auch diese Funktion wird in Abschnitt 5.5.2 erläutert.
Anschließend wird versucht, noch weitere Fahrten mit den bereits in `result` enthaltenen Fahrscheinen abzudecken. In der Funktion `checkTicketsForOtherTrips`, welche in Zeile 25 aufgerufen wird, werden nicht nur Fahrten der Preisstufe D berücksichtigt. Da eventuell Fahrten der Preisstufe D durch diese Fahrscheine bereits abgedeckt werden, dürfen diese nicht mehr in `tripsD` enthalten sein, da kein Fahrschein mehr für diese benötigt wird. Eine einfache Lösung ist, erneut alle Fahrten der Preisstufe D aus der Liste der zu optimierenden Fahrten zu sammeln.

Algorithmus 1 Optimierung für die Preisstufe D - Teil 1

Parameter: Fahrten die zu optimieren sind, mögliche Fahrscheine
Rückgabewert: Liste an optimalen Fahrscheinen der Preisstufe D

```
1: Funktion FAREZONE_D(tripItems, timeTickets)
2:     tripsD ← COLLECTTRIPS(tripItems, D)
3:     tripsLen ← LENGTH(tripsD)
4:     if tripsLen = 0 then
5:         return
6:     end if
7:     result                                  ▷ Enthält die benötigten Fahrscheine
8:     for ticketIndex ← 1 to LENGTH(timeTickets) do
9:         currentTicket ← timeTickets[ticketIndex]
10:        minNumTrips ← currentTicket.minNumTrips[D]
11:        if tripsLen ≥ minNumTrips then
12:            maxIndex, maxNumTrips ← BESTTICKETINTERVALL(tripsD, cur-
    rentTicket)
13:            while maxNumTrips ≥ minNumTrips do
14:                newTicket ← newTICKETTOBUY(currentTicket, D)
15:                result.ADD(newTicket)
16:                newTicket.ADDTRIPITEMS(tripsD, maxIndex, maxNumTrips)
17:                REMOVETRIPS(tripItems, tripsD, maxIndex, maxNumTrips)
18:                maxIndex, maxNumTrips ← BESTTICKETINTERVALL(tripsD,
    currentTicket)
19:            end while
20:                                         ▷ Weiter in Algorithmus 2
```

Algorithmus 2 Optimierung für die Preisstufe D - Teil 2

```
21:             if LENGTH(result) > 1 then
22:                 SORT(result)   ▷ Nach der Abfahrtszeit der ersten Fahrt sortieren
23:                 SUMUPTICKETS(result, timeTickets, ticketIndex + 1, D)
24:             end if
25:             CHECKTICKETSFOROTHERTRIPS(allTrips, result)
26:             tripsD ← COLLECTTRIPS(tripItems, D)
27:             tripsLen ← LENGTH(tripsD)
28:         end if
29:     end for
30:     return result
31: end Funktion
```

Preisstufe B

Zunächst wurde nicht die Preisstufe C betrachtet, sondern die Preisstufe B. Bei dieser Preisstufe gibt es verschiedene Geltungsbereiche, welche durch ein Zentralgebiet identifiziert werden. Für jede Fahrt müssen deshalb die durchquerten Tarifgebiete ermittelt werden. Anhand dieser Gebiete lässt sich feststellen, in welchen Geltungsbereich eine Fahrt fällt. Um die verschiedenen Geltungsbereiche darzustellen, wurde als interne Datenstruktur ein Graph genutzt. Als Knoten werden die einzelnen Tarifgebiete genutzt. Die Kanten sind gerichtet und gehen von dem Zentralgebiet aus zu den Tarifgebieten, die zu dem jeweiligen Gültigkeitsgebiet gehören, ein Beispiel ist in Abbildung 5.8 zu sehen. Dabei ist in Grau das zentrale Tarifgebiet, hier Marl, eingefärbt. Neben diesem Tarifgebiet ist das Ticket auch in den Tarifregionen, auf die ein Pfeil zeigt, gültig. Daraus ergibt sich, dass die Klassen `TripItem` und `TicketToBuy`

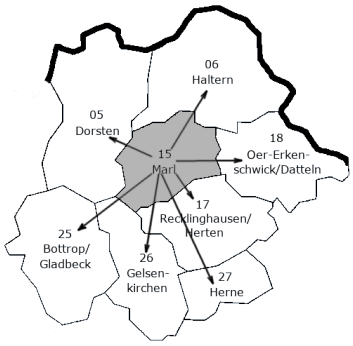


Abbildung 5.8: Gültigkeitsbereich 15 für die Preisstufe B [13].

erweitert werden müssen. Wie oben erläutert, müssen einer Fahrt die durchkreuzten Tarifgebiete zugeordnet werden. Den Fahrscheinen müssen die Tarifgebiete, in denen das Ticket gültig ist, zugewiesen werden, da diese nicht mehr im gesamten Tarifraum gültig sind. Des Weiteren wird der Klasse `TicketToBuy` als Attribut die zentrale Tarifregion hinzugefügt. Diese Information wird für den Nutzer zum Entwerten benötigt. Zusätzlich wurden die Klassen `Farezone` und `FarezoneTrips` hinzugefügt. Die Klasse `Farezone` enthält den Namen eines Tarifgebiets sowie dessen ID. Ein Objekt der Klasse `FarezoneTrips` enthält neben dem Tarifgebiet außerdem die Fahrten,

deren Starttarifgebiet mit dem Tarifgebiet übereinstimmt. Um die durchkreuzten Tarifgebiete einer Fahrt herauszufinden, wird erneut ein Blick auf die Klassen aus Öffi geworfen:

Jeder öffentlich zurückgelegte Abschnitt wird durch ein Objekt der Klasse `Public` repräsentiert. Diese hat eine Liste an `Stop`s, welche jeden einzelnen Zwischenhalt repräsentieren. Jeder dieser Halte gehört zu einer `Location`, welche eine ID besitzt. Der VRR bietet auf seiner OpenData Plattform einen Datensatz zu allen Haltestellen im Verbundraum an [47]. In dem Datensatz sind neben den Namen und IDs für jede Haltestelle auch die Wabe, in der die jeweilige Haltestelle liegt, enthalten. Ein Vergleich der beiden IDs zeigt, dass die Öffi-ID eine führende zwei hat, sonst aber mit den IDs des Datensatzes übereinstimmt. Für die zu optimierenden Fahrten soll, wenn der Nutzer auf Speichern klickt, für alle angefahrenen Haltestellen die ID in eine Liste eingefügt werden. Anschließend soll für jede dieser Haltestellen eine Datenbankabfrage gestartet werden, um die zugehörigen Waben zu erhalten. Die Waben werden dann der Fahrt zugeordnet. Da sich aus den Wabennummern die Tarifgebietnummer herleiten lässt, wird stattdessen die ID der Wabe gespeichert. Im Folgenden soll nun die Optimierung erläutert werden:

Wie in Algorithmus 3 zu sehen ist, werden bei dieser Optimierung nur die Fahrten einer Preisstufe betrachtet. Wie bei der Preisstufe D wird die Optimierung an dieser Stelle abgebrochen, wenn es keine Fahrten der Preisstufe B gibt. Sonst wird der Graph erzeugt.
Ein weiterer Unterschied zur Preisstufe D ist, dass die Fahrscheine der verschiedenen Geltungsbereiche unterschieden werden müssen. Die Variable `ticketsPerRegion` nimmt diese Funktion ein. Aus den Fahrscheinlisten der einzelnen Gebiete lässt sich zum Schluss eine gemeinsame Liste erstellen.
Zuerst wird wie bei der Preisstufe D über die Liste der möglichen Fahrscheine iteriert. In diesem Algorithmus kann ein Fahrschein ebenfalls direkt verworfen werden, wenn die Anzahl der Fahrten insgesamt geringer ist als der Schwellenwert des aktuell betrachteten Tickets (vergleiche Zeile 12). Ist dies jedoch nicht der Fall, werden die Fahrten dem jeweiligen Knoten ihres Starttarifgebiets zugeordnet.

Im nächsten Schritt wird für jedes Zentralgebiet die Anzahl der Fahrten, die maximal mit dem jeweiligen Fahrschein möglich sind, bestimmt. Eine getrennte Betrachtung der einzelnen Geltungsbereiche ist nicht möglich, da diese sich überlappen. Die Funktion `calculateMaxTrips`, welche in Zeile 18 aufgerufen wird, erstellt zunächst eine Liste an Fahrten, die innerhalb des aktuellen Geltungsbereichs liegen. Die Überprüfung ist nötig, denn wenn das Startgebiet einer Fahrt in dem aktuell betrachteten Bereich liegt, muss der Rest der Strecke nicht ebenfalls in diesem liegen. Für jeden Geltungsbereich wird daraufhin der Startindex und die Anzahl der Fahrten des maximalen Fahrtenintervalls für das aktuell betrachtete Ticket ermittelt, ähnlich zur vorher betrachteten Preisstufe. Außerdem wird die Liste selbst zurückgegeben.

Dann wird die Zentralregion mit der größten Anzahl an Fahrten bestimmt. Sollte es mehrere Regionen geben, die mit dem aktuell betrachteten Fahrschein gleich viele Fahrten abdecken, so wird die Region, in der insgesamt mehr Fahrten liegen, bevorzugt und anschließend die Region mit mehr Fahrscheinen. Mit dieser Priorisierung

wird versucht, möglichst viele Fahrscheine zusammenzufassen.
Ist die höchste Anzahl an Fahrten jedoch geringer als der Schwellenwert des betrachteten Tickets, wird abgebrochen. Sonst wird ein neues `TicketToBuy`-Objekt erstellt. In Zeile 26 wird der Geltungsbereich des Tickets ermittelt und dem Ticket zugewiesen. Des Weiteren werden dem Ticket die Fahrten aus dem Intervall zugeteilt und diese danach aus der Liste der zu optimierenden Fahrten entfernt.

Algorithmus 3 Optimierung für die Preisstufe B - Teil 1

Parameter: Fahrten die zu optimieren sind, mögliche Fahrscheine
Rückgabewert: Liste an optimalen Fahrscheinen der Preisstufe B

```
 1: Funktion FAREZONE_B(tripItems, timeTickets)
 2:     tripsB ← COLLECTTRIPS(tripItems, B)
 3:     if LENGTH(tripsB) = 0 then
 4:         return
 5:     end if
 6:     farezoneGraph ← LOADGRAPH
 7:     ticketsPerRegion                ▷ HashMap: Zentralgebiet - zugeordnete Tickets
 8:     for ticketIndex ← 1 to LENGTH(timeTickets) do
 9:         currentTicket ← timeTickets[ticketIndex]
10:         minNumTrips ← currentTicket.minNumTrips[B]
11:         while true do
12:             if LENGTH(tripsB) < minNumTrips then
13:                 break
14:             end if
15:             ADDTRIPSTONODES(farezoneGraph, tripsB)
16:             bestTicketIntervallPerFarezone                ▷ HashMap
17:             for zone in farezoneGraph.vertexSet do
18:                 maxIndex, maxNumTrips, tripsInRegion ← CALCULATEMAX-
    TRIPS(zone, currentTicket, farezoneGraph)
19:                 bestIntervallPerFarezone.zone ← maxIndex, maxNumTrips,
    tripsInRegion
20:             end for
21:             maxFarezone ← MAX(bestIntervallPerFarezone)
                                ▷ Zentralgebiet, mit dem maximalen maxNumTrips
22:             maxNum ← maxFarezone.maxNumTrips
23:             if maxNum ≥ minNumTrips then
24:                 newTicket ← newTICKETTOBUY(currentTicket, B)
25:                 ticketsPerRegion.maxFarezone.ADD(newTicket)
26:                 region ← CREATEREGION(maxFarezone, farezoneGraph)
27:                 newTicket.SETVALIDFAREZONES(maxFarezone, region)
28:                 maxStartIndex ← maxFarezone.maxIndex
29:                 maxTrips ← maxFarezone.tripsInRegion
30:                 newTicket.ADDTRIPS(maxTrips, maxStartIndex, maxNum)
31:                 DELETETRIPS(tripItems, maxTrips, maxStartIndex, maxNum)
32:                                         ▷ Weiter in Algorithmus 4
```

Algorithmus 4 Optimierung für die Preisstufe B - Teil 2

```
33:             else
34:                 break
35:             end if
36:             tripsB ← collectTrips(tripItems, B)
37:         end while
38:         for farezone in ticketsPerRegion do
39:             tickets ← ticketsPerRegion.farezone
40:             if length(tickets)> 1 then
41:                 sort(tickets)              ▷ Nach der ersten Abfahrtszeit sortieren
42:                 sumUpTickets(tickets, timeTickets, ticketIndex+1, B)
43:             end if
44:             checkTicketForOtherTrips(tripItems, tickets)
45:         end for
46:         tripsB ← collectTrips(tripItems, B)
47:     end for
48:     result                                  ▷ Liste mit allen Fahrscheinen
49:     for farezone in ticketsPerRegion do
50:         currentTickets ← ticketsPerRegion.farezone
51:         result.add(currentTickets)
52:     end for
53:     return result
54: end Funktion
```

Wenn keine Region mehr genug Fahrten für das aktuell betrachtete Ticket enthält, wird in Zeile 38 bis Zeile 47 für jede Region überprüft, ob die Fahrscheine sich zu günstigeren Fahrscheinen zusammenfassen lassen und ob weitere Fahrten durch diese Fahrscheine abgedeckt werden können. Zum Schluss wird aus den Listen für jede Region eine gemeinsame Liste an Fahrscheinen erstellt.

Preisstufe C

Der Entwurf für die Preisstufe C ist sehr ähnlich zu dem zuletzt vorgestellten Entwurf, deshalb ist der entsprechende Pseudo-Code Algorithmus 10 in Anhang B zu finden. Auch in diesem Fall gilt, dass sich die Geltungsbereiche der Regionen überlappen. Im Gegensatz zur Preisstufe B gibt es jedoch kein zentrales Tarifgebiet, sondern nur eine Liste an zugeordneten Tarifgebieten. Als Datenstruktur wurde dafür eine HashMap verwendet, welche als Schlüssel die ID der Region verwendet und als Wert die Liste an zugehörigen `FarezoneTrips` hat.

Wichtig bei dem Zuweisen der Fahrten an ihr jeweiliges Starttarifgebiet ist, dass jede Fahrt nur einmal zugewiesen wird.

Wie bereits bei der Preisstufe B wird jeweils die maximale Region bestimmt. Abschließend wird für jede Region versucht, die Fahrscheine zusammenzufassen und für weitere Fahrten zu nutzen.
Da hier kein Tarifgebiet verwendet wird, sondern eine ID zur Identifizierung, wird das Attribut der Zentralregion der Klasse `TicketToBuy` keine `Farezone` sein, sondern

ein Integer, da die Tarifgebiete ebenso über eine ID verfügen.

Preisstufe A

Für die Preisstufe A muss zwischen dem zwei-Waben-Tarif und dem Tarifgebiettarif unterschieden werden, siehe Abschnitt 2.1.1.

Für die Optimierung wird jede Abstufung der Preisstufe einzeln betrachtet, da je nach Abstufung unterschiedliche Tickets zur Verfügung stehen. Da die Optimierung für die jeweiligen Abstufungen gleich funktioniert, wurde dafür eine weitere Funktion entworfen, der in Algorithmus 6 zu sehen ist. Zuvor wird jedoch auf Algorithmus 5 eingegangen. In diesem werden zuerst Fahrscheinlisten für die beiden Tarifmöglichkeiten erstellt. Eventuell werden anschließend noch Tickets aus diesen Listen entfernt oder wieder hinzugefügt, dies ist abhängig vom Verkehrsverbund.

Im Falle des VRRs werden zunächst die Fahrten der Preisstufe A3, danach die Fahren der Preisstufe A2 und abschließend die Fahrten der Preisstufe A1 optimiert. In Algorithmus 6 werden zunächst die Fahrten der betrachteten Abstufung der Variablen `trips` zugewiesen. Dann werden von diesen Fahrten nur die Fahrten, die den zwei-Waben-Tarif nutzen, optimiert. Die Reihenfolge ist wichtig, denn nur wenn zuerst der zwei-Waben-Tarif betrachtet wird, können Fahrten, die innerhalb einer Wabe sind, von Fahrscheinen des zwei-Waben-Tarifs ebenfalls abgedeckt werden. Auf die Optimierung für zwei Waben selbst wird im folgenden Absatz eingegangen.

Eine Unterscheidung zwischen Fahrten im zwei-Waben-Tarif und Fahrten innerhalb eines Tarifgebiets ist anhand der durchkreuzten Waben möglich. Für eine Fahrt innerhalb eines Tarifgebiets gilt:

$$A = \{x_1, x_2, \dots x_n\}, \text{ wobei } x_i \text{ eine durchfahrene Wabe ist}$$
$$\forall i, j \in \{1, \dots n\} \text{ gilt: } \lfloor x_i : 10 \rfloor = \lfloor x_j : 10 \rfloor$$

Algorithmus 5 Optimierung für die Preisstufe A

Parameter: Liste zu optimierender Fahrten, mögliche Tickets
Rückgabewert: Fahrscheine für die Preisstufe A

```
1:  Funktion FAREZONE_A(tripItems, timeTickets)
2:      zweiWabenTarifTickets ← timeTickets
3:      farezoneTickets ← timeTickets
4:                  ▷ Gegebenenfalls anpassen der entsprechenden Fahrscheinlisten
5:      resultA3 ← FAREZONE_A(tripItems, zweiWabenTarifTickets, farezone-
    Tickets, A3)
6:                  ▷ Gegebenenfalls anpassen der entsprechenden Fahrscheinlisten
7:      resultA2 ←FAREZONE_A(tripItems, zweiWabenTarifTickets, farezone-
    Tickets, A2)
8:                  ▷ Gegebenenfalls anpassen der entsprechenden Fahrscheinlisten
9:      resultA1 ←FAREZONE_A(tripItems, zweiWabenTarifTickets, farezone-
    Tickets, A1)
10:     result ← A1 ∪ A2 ∪ A3
11:     return result
12: end Funktion
```

Algorithmus 6 Optimierung für die jeweilige Abstufung der Preisstufe A

Parameter: Liste zu optimierender Fahrten, mögliche Tickets für Fahrten in zwei Waben, mögliche Tickets für Fahrten in einem Tarifgebiet, Preisstufe
Rückgabewert: Fahrscheine für die jeweilige Abstufung der Preisstufe A

1: **Funktion** FAREZONE_A($tripItems, zweiWabenTickets, farezoneTickets, farezone$)
2: $trips \leftarrow$ COLLECTTRIPS(tripItems, farezone)
3: $zweiWabenTarif \leftarrow$ COLLECTZWEIWABEN(trips)
4: $resultZweiWaben \leftarrow$ ZWEIWABENOPTIMISATION(zweiWabenTarif, tripItems, zweiWabenTickets, farezone)
5: $trips \leftarrow$ COLLECTTRIPS(tripItems, farezone)
6: $farezoneTrips \leftarrow$ COLLECTFAREZONETRIPS(trips)
7: $resultFarezone \leftarrow$ FAREZONEOPTIMISATION(farezoneTrips, tripItems, farezoneTickets, farezone)
8: $result \leftarrow resultZweiWaben \cup resultFarezone$
9: **return** $result$
10: **end Funktion**

Die Fahrten im zwei-Waben-Tarif werden auf eine HashMap aufgeteilt, die als Schlüssel die zwei Waben nutzt und als Wert die Liste der zugehörigen Fahrten besitzt.

Da sich die Geltungsbereiche der Fahrscheine dieses Mal nicht überlappen können, muss nicht, wie bei den zwei vorherigen Ansätzen die maximale Region bestimmt werden, sondern es kann jede Region getrennt betrachtet werden. Deshalb wird in Zeile 3 über alle Paare von zwei Waben iteriert, die Fahrten zugewiesen bekommen haben. Für jedes dieser Paare wird nun wieder jedes Ticket ausprobiert. Dabei wird solange die aktuelle Region betrachtet, wie die Anzahl der Fahrten größer als der Schwellenwert des Tickets ist. Ist dies der Fall, wird das maximale Intervall der Fahrten für das aktuell betrachtete Ticket bestimmt. Wenn die maximale Anzahl an Fahrten geringer als der Schwellenwert ist, wird abgebrochen. Sonst wird ein neues `TicketToBuy`-Objekt erzeugt, welches das aktuell betrachtete Ticket inklusive der Preisstufe enthält. Danach werden dem Ticket die Fahrten zugewiesen und der Geltungsbereich des Tickets festgelegt (Zeile 17). Dann werden die Fahrten aus der Liste der zu optimierenden Fahrten entfernt.

Falls $maxNumTrips < minNumTrips$ gilt, wird versucht, die Fahrscheine zu kostengünstigeren Fahrscheinen zusammenzufassen sowie weitere Fahrten diesen Fahrscheinen zuzuweisen (vergleiche Zeile 21 bis Zeile 25).

Nachdem jedes Ticket für das Paar betrachtet wurde, werden die Fahrscheine dieser zwei Waben der Liste aller benötigten Fahrscheine hinzugefügt.
Nach der Optimierung der Fahrten im zwei-Waben-Tarif werden erneut die Fahrten der aktuellen Abstufung der Variablen `trips` zugewiesen. Von diesen Fahrten werden anschließend die Fahrten gesammelt, die innerhalb eines Tarifgebiets sind. Es kann nicht die Variable `trips` betrachtet werden, da diese zusätzlich Fahrten im zwei-Waben-Tarif enthält, welche keinem Ticket zugewiesen wurden. Der Entwurf für diese Optimierung ist in Anhang B zu sehen, da Algorithmus 12 sehr ähnlich zu dem bereits vorgestellten Algorithmus 7 ist. Der wesentliche Unterschied ist, dass

als Schlüssel für die `HashMap` das Tarifgebiet gewählt wird und nicht die IDs der zugehörigen Waben. Des Weiteren wird als Zentralregion für die Fahrscheine die ID des Tarifgebiets genommen und als gültiger Bereich das einzelne Tarifgebiet festgelegt.

Wie in den Konsequenzen für die betrachteten Fahrscheine bereits beschrieben wurde, wird die Preisstufe K nicht bei den Zeitfahrscheinen betrachtet, deshalb erfolgt kein entsprechender Entwurf.

Algorithmus 7 Optimierung der Fahrten im zwei-Waben-Tarif

Parameter: HashMap der zu optimierenden Fahrten, mögliche Tickets, Preisstufe
Rückgabewert: Fahrscheine für den zwei-Waben-Tarifder jeweiligen Preisstufe

```
1: Funktion zweiWabenOptimisation(zweiWabenTrips, allTrips, timeTickets, farezone)
2:     result                                          ▷ Liste aller benötigten Fahrten
3:     for zweiWaben in zweiWabenTrips do
4:         temporaryResult ▷ Fahrscheine für die aktuell betrachteten zwei Waben
5:         currentTrips ← zweiWabenTrips.zweiWaben
6:         sort(currentTrips)                          ▷ Sortiert nach der Abfahrtszeit
7:         for ticketIndex ← 1 to length(timeTickets) do
8:             currentTicket ← timeTickets[ticketIndex]
9:             minNumTrips ← currentTicket.minNumTrips[farezone]
10:            while length(currentTrips) ≥ minNumTrips do
11:                maxIndex, maxNumTrips ← bestTicketInterval(currentTrips, currentTicket)
12:                if maxNumTrips < minNumTrips then
13:                    break
14:                end if
15:                newTicket ← new TicketToBuy(currentTicket, farezone)
16:                newTicket.addTrips(currentTrips, maxIndex, maxNumTrips)
17:                newTicket.setValidFarezones(zweiWaben[0], zweiWaben)
18:                temporaryResult.add(newTicket)
19:                removeTrips(currentTrips, maxIndex, maxNumTrips)
20:            end while
21:            if length(temporaryResult)> 1 then
22:                sort(temporaryResult)  ▷ nach der ersten Abfahrtszeit sortieren
23:                sumUp(temporaryResult, timeTickets, ticketIndex + 1, farezone)
24:            end if
25:            checkTicketsForOtherTrips(allTrips, temporaryResult)
26:        end for
27:        result.add(temporaryResult)
28:    end for
29:    return result
30: end Funktion
```

5.5.2 Generalisierbarkeit

Die Optimierung für die verschiedenen Preisstufen arbeitet auf verschiedenen Datenstrukturen, weshalb diese sich nicht direkt auf andere Verkehrsverbünde übertragen lässt.
In den vier Entwürfen finden sich jedoch Funktionen, die in allen Ansätzen genutzt werden und sich für andere Verkehrsverbünde verwenden lassen. Damit muss für jeden Provider die Optimierung mit Zeitfahrscheinen manuell umgesetzt werden. Daraus resultiert, dass die gesamte Optimierung für jeden Provider anders umgesetzt werden muss, um die Zeitfahrscheine mit den Einzelfahrscheinen zu kombinieren. Deshalb wird die Klasse `MyProvider` um die abstrakte Methode `optimise` erweitert.
Im Folgenden sollen nun die Funktionen erläutert werden, welche in mehreren Ansätzen verwendet werden.

Fahrten einer Preisstufe suchen

Die Funktion `collectTrips` wird in allen Optimierungsschritten genutzt. Dabei wird eine neue Liste mit allen Fahrten erstellt, die der jeweiligen Preisstufe angehören. Für die Optimierung ist sicherzustellen, dass die Fahrten anschließend nach ihrer Abfahrtszeit aufsteigend sortiert sind.

Suche nach dem besten Ticket Intervall

Eine wichtige Funktion ist `bestTicketIntervall`. Diese Funktion sucht nach dem besten Intervall für eine Liste von Fahrten und gibt den Startindex des Intervalls sowie die Anzahl an Fahrten in diesem zurück. Die Funktion wird direkt in der Optimierung von den Preisstufen A und D aufgerufen, aber auch bei der Optimierung für die Preisstufen B und C verwendet. Voraussetzung für diese Funktion ist, dass die Fahrten nach ihrer Abfahrtszeit aufsteigend sortiert sind und sich nicht überlappen. Der zugehörige Pseudo-Code ist in Algorithmus 8 zu sehen. Zuerst wird der Startindex als unendlich angenommen und die Anzahl an Fahrten, die in dem Intervall liegen, auf null gesetzt.
Danach wird nach der ersten Fahrt gesucht, die mit dem Ticket möglich ist. Die Funktion `isValidTrip` überprüft, ob die übergebene Fahrt in den zeitlichen Gültigkeitsbereich eines Tickets fällt. Dies ist für manche Fahrscheine, wie beispielsweise das 4-StundenTicket, welches werktags erst ab 9 Uhr genutzt werden darf, nötig. Wenn eine Fahrt gefunden wurde, wird der Startindex auf die Position des Elements gesetzt und die Anzahl der Fahrten in dem Intervall auf eins. Es ist nicht nötig, weiter über die Liste zu iterieren, da ein Startindex gefunden wurde.

Wenn die Liste an Fahrten jedoch keine Fahrt enthält, für die das Ticket infrage kommt, ist der Index unendlich und es kann abgebrochen werden. Um die Laufzeit zu verbessern, wird in diesem Fall nicht nach dem größten Intervall gesucht, da das Ergebnis bereits vorher bekannt ist.

Wurde jedoch eine Fahrt gefunden, so wird anschließend das maximale Intervall gesucht. Dafür wird in Zeile 14 die Laufvariable auf den Startindex gesetzt. Dann wird überprüft, ob die aktuell betrachtete Fahrt eine gültige Fahrt für das Ticket

Algorithmus 8 Vollständige Suche nach dem besten Intervall

Parameter: Liste an Fahrten, Ticket
Rückgabewert: Startindex und Anzahl Fahrten des maximalen Intervalls

```
 1: Funktion BESTTICKETINTERVALL(tripItems, timeTicket)
 2:     maxIndex ← ∞
 3:     maxTrips ← 0
 4:     for i ← 1 to LENGTH(tripItems) do
 5:         currentTrip ← tripItems[i]
 6:         if timeTicket.ISVALIDTRIP(currentTrip) then
 7:             maxIndex ← i
 8:             maxTrips ← 1
 9:             break
10:         end if
11:     end for
12:     if maxIndex = ∞ then return
13:     end if
14:     for i ← maxIndex to LENGTH(tripItems) do
15:         currentTrip ← tripItems[i]
16:         if not timeTicket.ISVALIDTRIP(currentTrip) then
17:             continue
18:         end if
19:         counter ← 1
20:         for k ← i + 1 to LENGTH(tripItems) do
21:             startTime ← currentTrip.getFirstDepartureTime
22:             thisTrip ← tripItems[k]
23:             endTime ← thisTrip.getLastArrivalTime
24:             if ticket.ISVALIDTRIP(thisTrip) then
25:                 if endTime − startTime ≤ ticket.maxDuration then
26:                     counter ← counter + 1
27:                 else
28:                     break
29:                 end if
30:             end if
31:         end for
32:         if counter > maxCounter then
33:             maxCounter ← counter
34:             maxIndex ← i
35:         end if
36:     end for
37:     return maxIndex, maxCounter
38: end Funktion
```

ist. Dies ist zwar beim ersten Schleifendurchlauf bekannt, jedoch nicht für spätere Iterationen. Schlägt die Überprüfung fehl, wird die Fahrt nicht als Start in Betracht gezogen. Sonst wird für die aktuelle Fahrt gezählt, wie viele Fahrten noch in den Nutzungszeitraum des Tickets fallen. Dafür werden alle nachfolgenden Fahrten betrachtet, vergleiche Zeile 20. Für diese Fahrten muss ebenfalls überprüft werden, ob das Ticket sich eignet. Falls ja, muss noch überprüft werden, ob das gesamte Zeitintervall zwischen den beiden Fahrten durch das Ticket abgedeckt ist. Trifft dies zu, wird die Anzahl der Fahrten, die mit dem aktuellen Startindex möglich sind, erhöht, andernfalls wird abgebrochen. Es ist nicht nötig, noch weitere Fahrten zu betrachten, da deren Startzeit nach der aktuell betrachteten Endzeit liegt und somit ebenfalls über die Ticketdauer hinausragt.

Anschließend wird überprüft, ob die Anzahl möglicher Fahrten ab dem aktuellen Startindex größer ist als das bisher gefundene Maximum an Fahrten. Falls ja, werden sowohl das Maximum als auch der Startindex für das maximale Intervall angepasst. Dieses Verfahren basiert auf der vollständigen Suche und findet für ein Ticket die maximale Anzahl an Fahrten.

Zusammenfassen von Fahrscheinen

Eine weitere wichtige Funktion, die in allen Preisstufen benötigt wird, ist das Zusammenfassen mehrerer Fahrscheine zu einem neuen kostengünstigeren Fahrschein. Das Verfahren ist in Algorithmus 9 zu sehen.

Bevor die Funktion aufgerufen werden kann, müssen die Tickets nach ihrer ersten Abfahrtszeit sortiert sein, und die jeweiligen Fahrten jedes Tickets müssen ebenfalls aufsteigend nach der Abfahrtszeit sortiert sein.
Es wird über alle verbleibenden Fahrscheine iteriert. Die Fahrscheine, die bereits betrachtet wurden, haben einen Index kleiner als der Startindex. Für jede Iteration wird eine separate Liste an Fahrscheinen angelegt, damit die neuen Fahrscheine nicht mit dem gleichen Fahrschein ersetzt werden können.

Für das aktuell betrachtete Ticket in Zeile 3 wird das beste Intervall der alten Fahrscheine betrachtet. Im Gegensatz zu der eben vorgestellten Funktion arbeitet diese Funktion auf `TicketToBuy`-Objekten und nicht auf Objekten der Klasse `TripItem`. Des Weiteren wird nicht die maximale Anzahl an Fahrten gesucht, sondern das Ticketintervall, dessen Fahrscheine zusammen den maximalen Preis liefern. Es wird nach dem teuersten Intervall gesucht, um dieses gegen einen kostengünstigeren Fahrschein ersetzen zu können. Ein weiterer Unterschied ist, dass nicht mehr geprüft wird, ob der neue Fahrschein für eine Fahrt gültig ist, sondern für alle Fahrten, die einem Ticket zugeordnet werden. Dies geschieht in der Funktion `isNewValidTicket`. Da Algorithmus 13 sehr ähnlich zu Algorithmus 8 ist, ist der Pseudo-Code nur in Anhang B zu sehen.

Nur wenn der Preis der neuen Fahrscheine kleiner ist als der Preis des aktuell betrachteten Tickets (vergleiche Zeile 7), werden die Fahrscheine in dem Intervall durch das neue Ticket ersetzt. In diesem Fall wird ein neues `TicketToBuy`-Objekt erstellt, welches das aktuelle Ticket und die übergebene Preisstufe besitzt. Dann wird dieses Ticket in die Liste der neuen Fahrscheine eingefügt. Auch werden die Fahrten der alten Fahrscheine dem neuen Ticket zugewiesen und der Gültigkeitsbereich der alten Fahrscheine übernommen. Danach werden die alten Fahrscheine aus

Algorithmus 9 Zusammenfassen von Fahrscheinen

Parameter: Fahrscheine, die zusammengefasst werden sollen, weitere mögliche Tickets, aktuell betrachtetes Ticket, Preisstufe
Rückgabewert: Startindex und Anzahl Fahrten des maximalen Intervalls

```
1: Funktion sumUp(oldTickets, timeTickets, startIndex, farezone)
2:     for i ← startIndex to length(timeTickets) do
3:         currentTicket ← timeTickets[i]
4:         currentPrice ← currentTicket.price[farezone]
5:         newTickets ▷ Enthält alle TicketToBuy Objekte, die ein currentTicket nutzen
6:         maxIndex, maxNumTickets, maxPrice ←bestTicketIntervall( oldTickets, currentTicket)
7:         while currentPrice − maxPrice < 0 do
8:             newTicket ← new TicketToBuy(currentTicket, farezone)
9:             newTickets.add(newTicket)
10:            newTicket.addTrips(oldTicktes, maxIndex, maxNumTickets)
    ▷ Fügt die Fahrten, der Fahrscheine in dem Intervall dem neuen Ticket hinzu
11:            firstTicket ← oldTickets[maxIndex]
12:            newTicket.setValidFarezones(firstTicket.validFarezones, first-Ticket.mainID)
13:            removeTickets(oldTickets, maxIndex, maxNumTickets)
14:            sort(oldTickets)
15:            maxIndex, maxNumTrips, maxPrice ← bestTicketInter-vall(oldTickets, currentTicket)
16:        end while
17:        oldTickets.add(newTickets)
18:        sort(oldTickets)
19:    end for
20: end Funktion
```

der Liste der Fahrscheine entfernt. Zum Schluss wird wieder das maximale Intervall bestimmt.
Wenn sich keine weiteren Fahrscheine durch das aktuell betrachtete Ticket kostengünstiger ersetzen lassen, werden die aktuellen Fahrscheine in die Liste der zu ersetzenden Fahrscheine eingefügt. Diese wird daraufhin wieder chronologisch sortiert. Anschließend wird der nächste Fahrschein betrachtet.
Bei diesem Algorithmus wird zwar ebenfalls eine vollständige Suche in der Funktion `bestTicketIntervall` durchgeführt, dennoch ist diese Funktion nicht optimal, da Fahrscheine zusammenhängend betrachtet werden und die Fahrten nicht wieder aufgeteilt werden können.
Für den VRR gilt beispielsweise, dass wenn an acht aufeinanderfolgenden Tagen jeweils zwei Fahrten geplant werden, diese zunächst acht 24-StundenTicket zugewiesen werden. Die 24-StundenTicket würden durch vier 48-StundenTicket zusammengefasst werden. Als Nächstes würde das maximale Intervall für das 7-TageTicket bestimmt werden. Die ersten drei Fahrscheine würden durch dieses ersetzt werden. Damit würden die Ersten sieben Tage durch das 7-TageTicket abgedeckt und ein

48-StundenTicket würde für den achten Tag jedoch zusätzlich für den siebten Tag verwendet werden. Dadurch hätte der Nutzer für den siebten Tag zwei Tickets erworben.

Fahrscheine für andere Fahrten verwenden

Da sich durch das Zusammenfassen von Fahrscheinen neue zeitliche Geltungsbereiche ergeben, können weitere Fahrten mit den neuen Fahrscheinen abgedeckt werden. Diese Fahrten können eine geringere Preisstufe als in der aufrufenden Funktion betrachtet besitzen. Wichtig ist dabei jedoch, dass die durchkreuzten Waben beziehungsweise Tarifgebiete einer Fahrt in dem Gültigkeitsbereich des Tickets enthalten sind. Auch wird überprüft, ob die Fahrt durch das Verändern der Startzeit dem Ticket hinzugefügt werden kann. Ein Problem an dieser Stelle ist, dass der Algorithmus dadurch in ein lokales Minimum fallen kann. Der Entwurf für die Funktion ist in Algorithmus 15 in Anhang B zu sehen.

Anpassung des Klassendiagramms

Die Veränderungen, welche sich durch die einzelnen Abschnitte ergaben, sind in Abbildung 5.9 zu sehen.

Aus diesen Änderungen ergeben sich weitere Anpassungen an den einzelnen Entwürfen. So muss bei der Optimierung für die Preisstufe D jedes Ticket die Gültigkeitsbereiche zugewiesen bekommen, damit die gemeinsamen Funktionen für diese Preisstufe genutzt werden können.

5.5.3 Realisierung

Implementierung

Bei der Umsetzung des Entwurfs wurde zuerst nur die Preisstufe D umgesetzt, getrennt von der Optimierung mit Einzelfahrscheinen. Des Weiteren wurde erneut nur eine Person betrachtet. Dafür wurde die Optimierung getrennt von der Anwendung implementiert.
Anders als geplant wurden an dieser Stelle nicht die anderen Preisstufen umgesetzt, sondern stattdessen wurde dieser Ansatz mit der Optimierung von Einzelfahrscheinen für alle Preisstufen kombiniert. Dabei erfolgte die Integration des Ansatzes in die App. Dafür wurde die Funktion `optimise` der Klasse `MyVrrProvider` angepasst. Im Falle des VRRs wird die Optimierung mit Zeitfahrscheinen vor die Einzelfahrscheinoptimierung eingebunden. Dies resultiert aus dem Ergebnis der Fahrscheinanalyse in Abschnitt 3.3.4. Dabei werden zuerst möglichst viele Fahrten der Preisstufe D mit Zeitfahrscheinen abgedeckt und anschließend Einzelfahrscheine für die verbleibenden Fahrten genutzt.
Danach wurde die Anzahl der Personen erhöht. Hierbei wurden jedoch zunächst nicht die 24-StundenTickets für mehrere Personen berücksichtigt. Dabei zeigte sich, dass die Klasse `TripItem` um ein Attribut, welches angibt, wie viele Nutzer noch kein Ticket erhalten haben, ergänzt werden musste.
Auch die Verwendung von alten Fahrscheinen wurde an die neue Komponente für

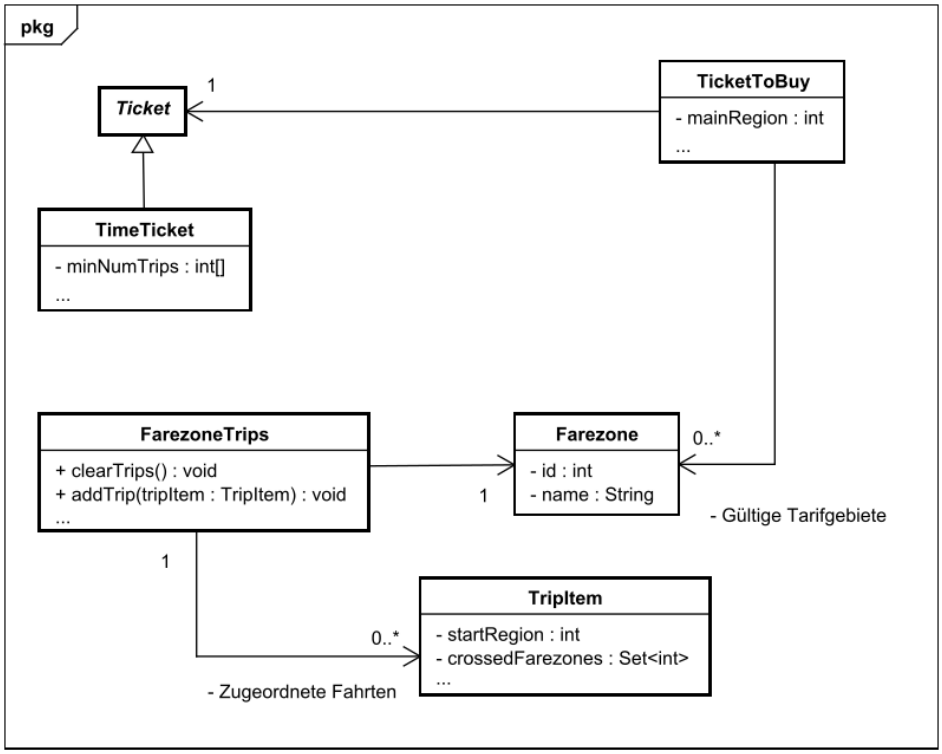


Abbildung 5.9: Anpassung des Klassendiagramms an den Entwurf für die Optimierung mit Zeitfahrscheinen.

die Preisstufe D angepasst. Wie in Abschnitt 5.4 werden, wenn später noch weitere Fahrten hinzugefügt werden, Fahrscheine, die in der Zukunft liegen, entfernt. Die zugehörigen Fahrten sollen mit den neuen Fahrten zusammen optimiert werden. Wenn ein Ticket gelöscht wird, muss die Anzahl der Nutzer ohne Ticket erhöht werden. Daraus resultiert, dass die vorherigen Komponenten ebenfalls angepasst werden müssen.
Bei der Verwendung von alten Fahrscheinen werden im Falle des VRRs zuerst die Zeitfahrscheine verwendet und schließlich die verbleibenden Fahrten mit den Einzelfahrscheinen optimiert. Dies wurde bei der Anpassung der Verwendung alter Fahrscheine berücksichtigt.
Dann wurde, wie bereits im Entwurf, die Preisstufe B und danach die Preisstufe C umgesetzt. Diese wurden zunächst außerhalb der Anwendung realisiert und nach erfolgreicher Testung eingebunden. Bei diesen beiden Preisstufen kann es vorkommen, dass der Gültigkeitsbereich des Fahrscheins nicht eindeutig ist. Aufgrund der begrenzten Leistung, die auf Smartphones zur Verfügung steht, kann der Nutzer anders als ursprünglich entworfen nicht unter den möglichen Gültigkeitsbereichen wählen. Dafür hätten parallele Optimierungen durchgeführt werden müssen, stattdessen wird der erste Gültigkeitsbereich gewählt, der den maximalen Wert erreicht.

Abschließend wurde die Optimierung der Preisstufe A umgesetzt. Dabei wurde zunächst ebenfalls auf die Integration in die Anwendung verzichtet. Für die fünf Städte, die in zwei Tarifgebiete unterteilt sind, gilt, dass die Zeitfahrscheine in beiden Tarifgebieten gültig sind. Die Umsetzung dafür wird am Beispiel der Stadt

Duisburg erläutert: Die Stadt Duisburg ist in die zwei Tarifgebiete 23 und 33 unterteilt. Fahrten, die im Tarifgebiet 33 starten, werden der Liste an Fahrten für das Tarifgebiet 23 zugewiesen. Bei der Festlegung des Gültigkeitsbereichs des Tickets werden die fünf Sonderfälle überprüft und in diesem Beispiel das Tarifgebiet 33 dem Gültigkeitsbereich hinzugefügt.
Bei der Überprüfung, ob ein Ticket für weitere Fahrten genutzt werden kann, zeigte sich, dass dies für den zwei-Waben-Tarif anders gelöst werden muss als für die anderen Preisstufen, weshalb die Klasse `TicketToBuy` um das Attribut `isZweiWaben` erweitert wurde.
Bei dem zwei-Waben-Tarif wurde festgestellt, dass dieser nicht die Funktion `checkTicketsForOtherTrips` nutzen kann, da bei dieser die Tarifgebiete, in denen die Fahrscheine gültig sind, neu erstellt werden und als ID die Wabennummer anstelle der Tarifgebiet-ID nutzen.
Zum Schluss wurde die Optimierung für mehrere Personen umgesetzt. Dafür werden, nachdem die alten Fahrscheine verwendet wurden, eine HashMap erstellt, welche als Schlüssel die Personennummer nutzt und als Wert jeweils die Reisen, bei denen entsprechend viele Personen mitfahren. Ein Beispiel ist in Abbildung 5.10 zu sehen.

				HashMap:		
Fahrt:	F_1	F_2	F_3	1	–	F_1, F_2, F_3
Personen:	2	1	3	2	–	F_1, F_3
				3	–	F_3

Abbildung 5.10: Visualisierung der Vorbereitung für die Optimierung mit mehreren Personen.

Dabei wird der ersten Person jede Fahrt zugewiesen. Die zweite Person hingegen bekommt nur die Fahrten F_1 und F_3 zugewiesen und die letzte Person nur die dritte Fahrt.

Diese Listen werden der Reihe nach mit Zeitfahrscheinen optimiert. Anschließend werden, wenn möglich, die 24-StundenTickets und 48-StundenTickets zusammengefasst, da diese, wie in Abschnitt 2.1.2 erläutert, für bis zu fünf Personen gelten. Dabei müssen zum einen die zeitlichen Nutzungsräume übereinstimmen und zum anderen die Geltungsbereiche, in denen die Fahrscheine gültig sind.
Bis jetzt wurden nur den Fahrscheinen Fahrten zugewiesen, aber die Fahrten haben noch keine Tickets zugewiesen bekommen. Dafür wird über die Fahrscheine iteriert und jeder zugeordneten Fahrt das aktuelle Ticket hinzugefügt.

Dann werden die verbleibenden Fahrten wieder zusammen in eine Liste eingefügt. Dabei wird jede Fahrt so oft eingefügt, wie sie Nutzer ohne Ticket hat. Zum Schluss wird diese Liste mit Einzelfahrscheinen optimiert.

Testen

Als Erstes wurden die allgemeinen Funktionen, welche für mehrere Preisstufen genutzt werden, mittels Komponententest überprüft. Dafür wurden die Funktionen mit verschiedenen Fahrten, welche mittels Platzhalter erzeugt wurden, und verschiedenen Fahrscheinen aufgerufen. Mit dieser Testart wurde jeweils die Korrektheit der

Optimierung für jede einzelne Preisstufe überprüft. Dafür wurden verschiedene Fahrten erstellt, welche unterschiedliche Waben durchqueren, sowie verschiedene Start- und Endzeiten haben. Für die ersten Tests einer Optimierung einer Preisstufe hatten die Fahrten die gleiche Preisstufe, im Verlauf wurden jedoch auch Fahrten anderer Preisstufen hinzugenommen. Bei den Tests der Preisstufe B und C wurden sowohl Fahrten in nicht überlappenden Gebieten betrachtet als auch Fahrten in überlappenden Gebieten. Dadurch wurde sichergestellt, dass die Fahrscheine einen festen Geltungsbereich zugewiesen bekommen und alle zugeordneten Fahrten in diesem liegen. Anschließend wurden die verschiedenen Preisstufen zusammengefügt und das Ergebnis überprüft. Im Rahmen der Integrationstests wurde überprüft, ob die Optimierung als Kombination aus den drei bisherigen Komponenten funktioniert.
Da die Optimierung nur für die Klasse `MyVrrProvider` realisiert ist, wird im Rahmen der System- und Integrationstests hier nur auf die Korrektheit für diesen Verkehrsverbund eingegangen. Für andere Verkehrsverbünde müssen gegebenenfalls die Tests angepasst sowie erneut durchgeführt werden.

Bei den Systemtests wurden erneut die Fahrten zuerst geplant und optimiert. Um die Verwendung von alten Fahrscheinen zu überprüfen, wurde wieder die Systemzeit verändert und danach Fahrten gelöscht oder editiert. Ein Problem, welches sich bei den Systemtests zeigte, waren Haltestellen, die mehr als einer Wabe zugeordnet sind. Im VRR sind Haltestellen bis zu drei Waben zugeordnet [47]. Um dieses Problem zu lösen, werden Haltestellen mit mehr als einer zugeordneten Wabe übersprungen und stattdessen die jeweils nächste Haltestelle betrachtet. Bei dieser Lösung erhalten jedoch Strecken, die komplett auf einer Grenze zwischen zwei oder mehr Waben verlaufen, keine Waben zugewiesen. Der Fall, dass ein Abschnitt einer Fahrt komplett auf der Grenze zwischen zwei Waben verläuft, wurde jedoch als unwahrscheinlich eingeschätzt und damit als bessere Lösung, als eine der möglichen Waben zufällig auszuwählen. Mithilfe von Systemtests wurde neben der neuen Komponente außerdem die angepasste Verwendung von früheren Fahrscheinen überprüft. Dafür wurde erneut wie in Tabelle 5.1 vorgegangen. Ein Unterschied zu den Einzelfahrscheinen ist jedoch, dass für die Preisstufe A noch der Wochentag berücksichtigt werden muss. Dies liegt an den abweichenden Geltungszeiträumen für das HappyHourTicket, vergleiche Abschnitt 2.2. Zusätzlich muss der angegebene Geltungsbereich der Fahrscheine überprüft werden. Dabei wird überprüft, ob die Fahrten in dem angegebenen Bereich liegen.

Hierfür wurden erneut Fahrten erstellt und die Systemzeit zwischen zwei Fahrten gestellt. Dann wurden Fahrten editiert oder gelöscht und überprüft, ob das Resultat mit dem zu erwarteten Ergebnis übereinstimmt. Dabei zeigte sich, dass nicht alle Sonderfälle korrekt abgedeckt werden. So werden Fahrten, die von dem Tarifgebiet Duisburg Nord in das Tarifgebiet Duisburg Süd in zwei Waben liegen, dem zwei-Waben-Tarif zugeordnet. Ein zugeordneter Zeitfahrschein zeigt als Geltungsbereich nur die beiden Waben an, und nicht ganz Duisburg.

5.6 Ablauf der Optimierung

Um anschaulich nachvollziehen zu können, wie die Optimierung abläuft, wird der Ablauf in einem Sequenzdiagramm dargestellt. Zur besseren Lesbarkeit ist das Sequenz-

diagramm in zwei Abbildungen (Abbildung 5.11 und 5.12) unterteilt. Das Sequenzdiagramm ist nicht vollständig und verzichtet an den Stellen der Zeitoptimierung und Einzelfahrscheinoptimierung auf eine genaue Ausführung des Kontrollflusses, da diese Schritte bereits im vorangegangenen Text erläutert wurden. Dies betrifft die Funktionen `optimisationWithOldTickets`, `timeOptimisation` und `optimisationNewTickets`.

Die Optimierung startet, wenn der Nutzer in der Ansicht seiner aktuell geplanten Fahrten auf weiter klickt. Wie in Abbildung 5.11 zu sehen ist, werden zunächst die neuen Fahrten in die Liste der alten Fahrten eingefügt, dabei wird in der Funktion `prepareOptimisation` sichergestellt, dass keine Fahrten mehrfach enthalten sind und sich die Fahrten nicht überlappen. Anschließend werden die gespeicherten Fahrscheine geladen. Danach wird die Funktion `optimise` des jeweiligen Providers aufgerufen. Im Falle des VRR werden zuerst die Fahrten entfernt, welche keine gültige Preisstufe besitzen. Des Weiteren müssen nur Fahrten betrachtet werden, die optimiert werden sollen. Diese Teilliste an Fahrten wird nun nach der Preisstufe aufsteigend sortiert. Im Anschluss werden die Fahrten auf die entsprechenden Nutzerklassen aufgeteilt. Dabei wird eine Fahrt in die Liste der Nutzerklasse eingefügt, wenn die Anzahl der reisenden Personen dieser Nutzerklasse mindestens eins beträgt.

Dazu werden die geladenen Fahrscheine betrachtet:

1. Fahrscheine, deren Fahrten alle in der Zukunft liegen, werden gelöscht. Bei den zugeordneten Fahrten wird die Anzahl der Nutzer ohne Fahrscheine erhöht.

2. Fahrscheine, die mindestens eine Fahrt in der Vergangenheit haben und mindestens eine freie Fahrt werden zum einen als benötigte Fahrscheine gemerkt und zum anderen für die Optimierung mit alten Fahrscheinen gemerkt. Für Zeitfahrscheine gilt, dass die Anzahl freier Fahrten immer den Wert `Integer_max_value` besitzt.

3. Fahrscheine, die mindestens eine Fahrt in der Vergangenheit haben, aber nicht über eine freie Fahrt verfügen, werden nur in die Liste der benötigten Fahrscheine eingefügt.

Damit ist die Vorbereitung abgeschlossen. Daran schließt sich die Optimierung in Abbildung 5.12 an. Zunächst werden die Fahrten mit alten Fahrscheinen optimiert. Dann wird die Optimierung mit Zeitfahrscheinen vorbereitet. Dafür wird die Liste der Nutzerfahrten weiter aufgeteilt: Für jeden Nutzer wird nun eine eigene Liste an Fahrten erstellt. Diese Listen werden einzeln in der Zeitoptimierung optimiert. Von jedem Nutzer werden die Fahrten ohne Zeitfahrscheine wieder in eine gemeinsame Liste eingefügt. Nachdem jeder Nutzer betrachtet wurde, wird für jede Nutzerklasse diese gemeinsame Liste an Fahrten ohne Zeitfahrscheinen mit Einzelfahrscheinen optimiert. Dann werden die drei Listen alte Fahrscheine, neue Zeitfahrscheine und neue Einzelfahrscheine für jede Nutzerklasse zusammengefügt. Zum Schluss wird die Ticketliste gespeichert, vergleiche Abbildung 5.12. Damit ist die Optimierung abgeschlossen.

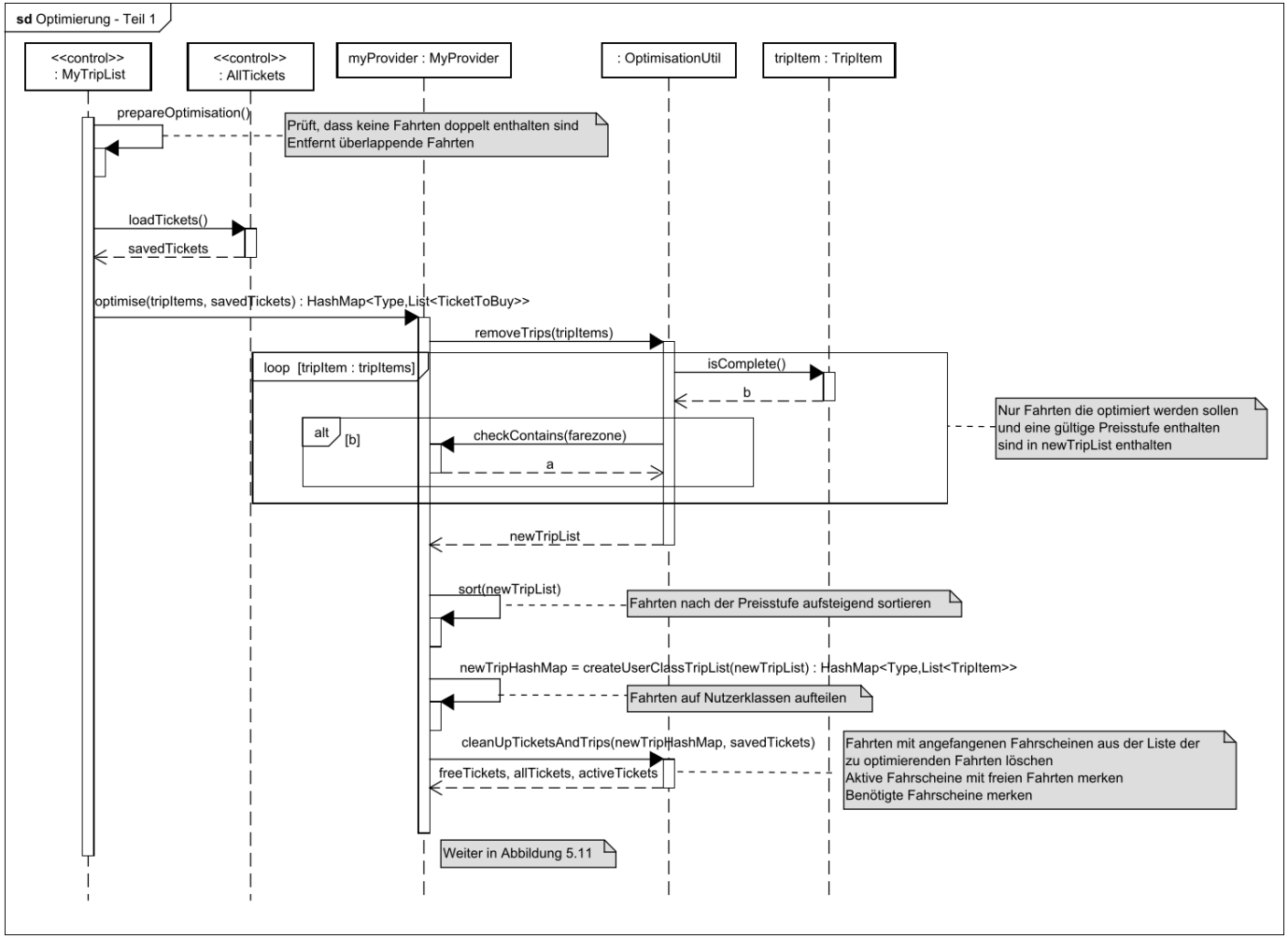


Abbildung 5.11: Sequenzdiagramm der Optimierung - Teil 1

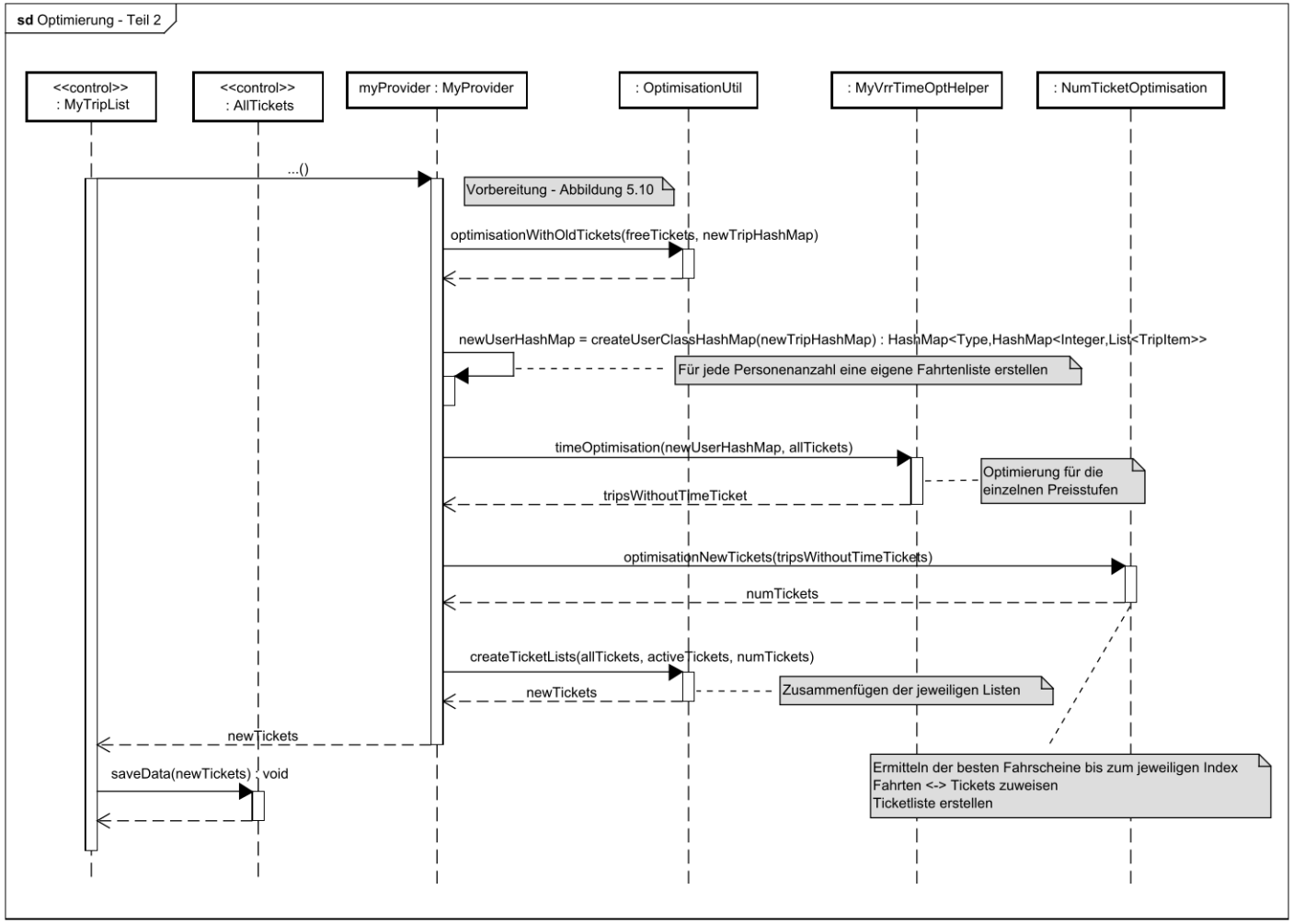


Abbildung 5.12: Sequenzdiagramm der Optimierung - Teil 2

5.7 Aktualisieren von Verbindungsdaten

Auch wenn in der ursprünglichen Planung nach der Optimierung die App fertig implementiert sein sollte, so wurden bis zu diesem Zeitpunkt die Verbindungsdaten einmal abgerufen und seit dem nicht mehr verändert. Das ist besonders bei Fahrten, die in der Zukunft liegen, nicht sinnvoll, da jene eventuell ausfallen können oder durch Verspätung eines Abschnitts die Fahrt nicht mehr wie geplant stattfinden kann.
Um jedes Mal die gleiche Anfrage zu starten, wurde das Klassendiagramm um die Klasse `MyURLParameter` in Abbildung 5.13 erweitert. Die Klasse `MyURLParameter` besitzt als Attribute die Einstellungen, die zum Zeitpunkt der Abfrage aktuell waren, die vom Nutzer angegebene Startzeit, Start- und Zielpunkt und eventuell einen Zwischenhalt. Die Möglichkeit zum Aktualisieren hat der Nutzer in verschiedenen

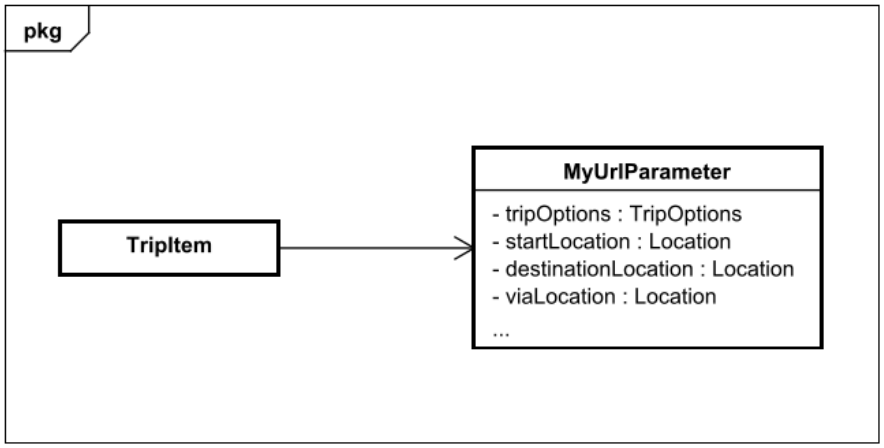


Abbildung 5.13: Erweiterung des Klassendiagramms um die Klasse `MyUrlParameter` um Fahrten zu aktualisieren.

Schritten, jeweils in der Detailansicht von:

- Planung einer einzelnen Fahrt
- Übersicht von noch nicht optimierten Fahrten
- Übersicht von zukünftigen Fahrten
- Fahrscheinübersicht

In den beiden zuletzt genannten Fällen muss die Fahrt ebenfalls in der entsprechenden anderen Liste aktualisiert werden.
Für diese Komponente gilt, dass nur Systemtests durchgeführt wurden. Eine Überprüfung der einzelnen Komponente erfolgte dadurch ebenso wie die erfolgreiche Integration. Auf die Durchführung der Tests wird in Kapitel 6.1 eingegangen, da in Rahmen dieses Tests die gesamte Anwendung getestet wurde. Hier werden die Ergebnisse und Probleme vorgestellt.

Bei den Tests zeigte sich, dass nicht immer die gleiche Fahrt erneut gefunden wurde. Sollte die Verbindung nicht gefunden werden mit der vom Nutzer angegebenen Startzeit, so wird eine neue Anfrage gestartet, welche jedoch stattdessen die Startzeit der Fahrt als Abfahrtszeit nutzt. Schlägt auch dieser Versuch fehl, wird der Nutzer darüber informiert, dass keine Informationen mehr zu der Verbindung gefunden werden konnten. Dann kann der Nutzer selbst die Fahrt löschen oder aber eine einfache Fahrplanauskunft starten. Da die Fahrten nicht immer bei einer neuen Serveranfrage gefunden werden, wird auf eine automatische Aktualisierung beim Starten der Aktivität *über geplante Fahrten informieren* verzichtet.

Das vollständige Klassendiagramm ist in Abbildung B.2 in Anhang B zu sehen. In Anhang B.3 in der Abbildung B.4 ist das umgesetzte endgültige Layout zu sehen. Aufgrund des Umfangs der Optimierung konnten im Rahmen der Arbeit die Einschränkungen, die zu Beginn getroffen wurden, nicht behoben werden.

Kapitel 6

Evaluation

Im vorherigen Kapitel wurde entsprechend des in Kapitel 3 vorgestellten Entwicklungsmodells (siehe Abbildung 4.9) für jeden Entwicklungsschritt die drei Teststufen Komponententest, Integrationstest und Systemtest durchgeführt. In diesem Kapitel wird ein abschließender Systemtest durchgeführt sowie im Rahmen dessen ein Vergleich zwischen den ursprünglichen Anforderungen und der realisierten Anwendung gezogen. Im Anschluss wird die Optimierung der App mit einer manuellen Optimierung für reelle Fahrten verglichen. Zum Schluss wird, wie im Entwicklungsmodell zu sehen ist, die Akzeptanz durch eine kleine Nutzergruppe überprüft.

6.1 Systemtests

Zunächst wird die Durchführung des finalen Systemtests erläutert. Danach erfolgt ein Abgleich zwischen den Anforderungen an die Anwendung mit der fertigen App.

6.1.1 Abschließender Systemtest

Im abschließenden Systemtest wurde überprüft, ob durch die letzte Komponente, das Aktualisieren von Verbindungen, andere Komponenten beeinträchtigt werden. Die Aktualisierung musste in verschiedenen Bereichen getestet werden.

Damit Veränderungen an den Fahrten beobachtet werden können, mussten Fahrten gefunden werden, welche verspätet waren, deren Verspätung sich aber noch verändert. Um zu überprüfen, ob die veränderten Fahrdaten auch in der Fahrscheinübersicht angezeigt werden, mussten zunächst Fahrten optimiert werden. Dann wurde überprüft, ob das Aktualisieren in der Fahrscheinübersicht zu einer Veränderung der Fahrtenliste führt. Zusätzlich wurde die Aktualisierung in der Detailansicht von noch nicht gespeicherten Fahrten überprüft.
Zudem wurden die vorherigen Systemtests wiederholt, um sicherzustellen, dass alle Komponenten erfolgreich umgesetzt wurden.

Außerdem wurden weitere Tests durchgeführt, um für jeden Fahrschein und jede Preisstufe zu überprüfen, ob die Geltungsbereiche korrekt angezeigt werden. Dabei wurde festgestellt, dass für Fahrten in zwei benachbarten Waben der Stadt Duisburg, die jedoch in den verschiedenen Tarifgebieten liegen, dem zwei-Waben-Tarif zugeordnet werden. Die entsprechenden Fahrscheine sind nur in diesen zwei Waben

gültig und nicht in ganz Duisburg. Ebenso wurde überprüft, ob für jeden Fahrschein die Art des Entwertens korrekt angegeben wird. Dafür wurden unter anderem für jeden Fahrschein und jede Preisstufe Fahrten geplant, sodass nur dieser Fahrschein genutzt wird. In der Detailansicht der einzelnen Fahrten wurde dann überprüft, ob die Anzeige korrekt ist. Dabei müssen für jede Preisstufe die Angaben der App zu denen passen, die für das Entwerten benötigt werden. Im Falle der Preisstufe K für Einzelfahrscheine werden beispielsweise sowohl die Start- als auch die Zielhaltestelle benötigt. In der Fahrscheinanzeige wurde außerdem noch überprüft, ob die gruppierte Anzeige die richtigen Fahrscheine anzeigt und diese Anzeige verständlich ist. Durch das Hinzufügen der letzten Komponente konnten keine Veränderungen an den anderen Komponenten festgestellt werden. Die Optimierung blieb vollständig funktionsfähig. Bei der Überprüfung der Anzeige von Geltungsbereichen, -dauer und Entwertung konnten keine weiteren Probleme festgestellt werden.

Damit wurde die Realisierung der App abgeschlossen. Im Folgenden wird die Anwendung mit den geplanten Anforderungen verglichen.

6.1.2 Vergleich der App mit den Anforderungen

Da die Optimierung aufwendiger als erwartet war, wurden nicht alle geplanten Funktionen und Anforderungen realisiert. Insbesondere die Optimierung der Zeitfahrscheine hat viel Aufwand benötigt. Dies liegt vor allem an den verschiedenen Gebietsstrukturen für jede einzelne Preisstufe.
Die Optimierung basiert auf den geplanten Fahrten des Nutzers in der App. Die Planung von Fahrten ist in der Anwendung realisiert worden. Dabei wurden viele geplante Funktionen umgesetzt, aufgrund des Optimierungsumfangs und der begrenzten Zeit, die zur Verfügung stand, wurden vor allem Komfortfunktionen gestrichen, welche jedoch nachträglich hinzugefügt werden können.
Die Überprüfung der Optimierung lieferte bei den Tests sowohl für eine Person als auch für mehrere Personen gute Ergebnisse. Ob dies jedoch auch für reale Fahrten zutrifft, wird in dem Vergleich der Optimierungen (vergleiche Abschnitt 6.2) festgestellt.
Abschnitt 3.1.3 zeigt, dass nicht alle geplanten Anforderungen umgesetzt worden sind. Fehlende Komfortfunktionen sind unter anderem:

- Berücksichtigung von Favoriten oder alten Suchanfragen bei der Suche nach Haltestellen.
- Das Fehlen der Schritt-für-Schritt Navigation bedeutet keinen Informationsverlust für den Nutzer, da die Informationen bereits in der Detailansicht einer Fahrt enthalten sind.
- Das Teilen von Fahrten mit den Kontakten
- Das Speichern von Fahrten im Kalender.
- Das Anzeigen der letzten Verbindung mit nur einem Klick ist nicht möglich, der Nutzer kann aber manuell frühere oder spätere Verbindungen anfordern und so die letzte Fahrt eines Tages finden.

Zwar kann der Nutzer wie in der VRR App in der Planung einer Fahrt direkt auf die Einstellungen zugreifen und die einzelnen Ortsangaben löschen. Ein Vertauschen konnte aber aus Platzgründen nicht realisiert werden. Die Einstellungen des Nutzers werden bei der Verbindungsabfrage berücksichtigt, dabei hat der Nutzer aber nicht die detaillierten Einstellungsmöglichkeiten wie beabsichtigt. Dennoch lassen sich die berücksichtigten Verkehrsmittel wählen. Außerdem kann der Nutzer zwischen drei Profilen für das Gehtempo und die Barrierefreiheit entscheiden. Da die Optionen in Öffi nicht erweitert wurden, können keine Weiteren berücksichtigt werden. Bei dem Formular wird für das Ausfüllen der jeweiligen Felder keine zusätzliche Ansicht geöffnet. Das Hinzufügen eines Zwischenhaltes ist nur bei der einfachen Fahrplanauskunft möglich. Wie in Abschnitt 5.2.2 erläutert, wird die Preisstufe für die Optimierung zwingend benötigt, jedoch wird diese für Fahrten mit Zwischenhalten bei einer Abfrage an die Schnittstelle nicht angegeben. Für Fahrten, die optimiert werden, muss der Nutzer, wie beabsichtigt, die Anzahl reisender Personen festlegen.
In der Übersicht von möglichen Verbindungen werden wie geplant die Nutzerangaben Start, Ziel, Datum und Uhrzeit angezeigt. Die Darstellung aller Verbindungen erfolgt in Listendarstellung. Dabei ist für jede einzelne Fahrt die Abfahrts- und Ankunftszeit, die Dauer, die Preisstufe sowie die Abfolge der Verkehrsmittel direkt abzulesen. Anders als beabsichtigt wird für jede Linie die gleiche Farbe genutzt. Um die einzelnen Verkehrsmittel zu unterscheiden, werden stattdessen verschiedene Icons genutzt, vergleiche Abbildung B.4b. In der Abfolge der Verkehrsmittel sind jedoch nicht die Warte- oder Umstiegszeiten ablesbar. Die Informationen dazu ließen sich nicht wie gewünscht aus den Klassen von Öffi ermitteln.
Die gleichen Zeiten sollten genauso in der Detailansicht enthalten sein, konnten dort folglich aber ebenfalls nicht realisiert werden. In der gleichen Ansicht sollte der Nutzer über Störungen informiert werden, dies ist aber nur für Verspätungen und Gleiswechsel möglich. Die Informationen werden in Öffi für die VRR-Schnittstelle nicht angeboten. Wie im Entwurf vorgesehen, lassen sich für die einzelnen Verbindungen die Zwischenhalte anzeigen. Für Abschnitte, die individuell zurückgelegt werden, wurde eine Navigation umgesetzt. Im Gegensatz zur ursprünglichen Planung wird jedoch eine weitere Anwendung benötigt, da das Einbinden der Navigations-API kostenpflichtig ist. Dafür sind alle geplanten Kriterien in der Übersicht umgesetzt worden. Wie bei der Durchführung der Systemtests in Kapitel 5 jedoch festgestellt wurde, sind die Angaben nicht immer vollständig. So fehlen beispielsweise die Angaben zum Gleis oder Bussteig. Auch die Benennung der Haltestellen ist nicht einheitlich. Diese werden jedoch von der Datenbank des VRRs vorgeben und können somit nicht vereinheitlicht werden.
Eine automatische Information über Verspätungen wurde nicht realisiert, weil bei den Systemtests festgestellt wurde, dass manche Verbindungen nicht erneut gefunden wurden. Der Nutzer kann sich aber manuell über Veränderungen bei den einzelnen Fahrten informieren.
Dafür wurde eine neue Komfortfunktion während der Implementierung hinzugefügt: Das Kopieren von Fahrten, damit der Nutzer bei gleichen Fahrten nicht erneut die gleichen Eingaben machen muss, sondern nur die Uhrzeit und das Datum wählt.
Einige Funktionen konnten nicht wie geplant realisiert werden, andere wurden anders als geplant umgesetzt. Ein Beispiel dafür ist das Editieren von Fahrten, welches

von dem Aktivitätsdiagramm abweicht. Statt zu ermitteln, welche Änderungen sich aus der veränderten Fahrt für die Fahrscheine ergeben, wird der Nutzer nur zu Beginn darüber informiert, dass die ermittelten Fahrscheinkosten eventuell nicht mehr die minimalen Kosten darstellen könnten.
Aus der Analyse der bestehenden Fahrscheinberater und Fahrscheine ergaben sich Konsequenzen, welche nur teilweise umgesetzt wurden. Wie gewünscht werden abweichend vom VRR-Ticketberater die Uhrzeiten der einzelnen Fahrten bei der Optimierung berücksichtigt. Neben der Geltungsdauer von Fahrscheinen werden die weiteren Vorschriften der Fahrscheine eingehalten. Dabei gilt, dass die Regeln nicht vollständig ausgenutzt werden. So werden die Sonderregeln nur am Wochenende und nicht an Feiertagen berücksichtigt. Dass der Nutzer am Wochenende drei Kinder unentgeltlich mitnehmen kann, wurde nicht betrachtet. Die Regel wurde nicht verwendet, um zunächst alle Fahrscheine zu berücksichtigen und die Spezialfälle zu minimieren. Für die Zeitfahrscheine gilt, dass diese einen festgelegten Geltungsbereich zugewiesen bekommen. Jeder Zeitfahrschein besitzt einen vom Verkehrsverbund vorgegebenen Gültigkeitsbereich. Alle zugeordneten Fahrten liegen in diesem. Ist der Bereich nicht eindeutig, kann der Nutzer nicht wie beabsichtigt unter den möglichen Bereichen wählen. Dies liegt an der begrenzten Leistung, die auf Smartphones zur Verfügung steht und damit eine parallele Optimierung nicht durchführbar wäre. Nicht wie im VRR-Ticketberater muss der Nutzer keine minimale Häufigkeit für eine Fahrt angeben, sondern kann die Fahrten entsprechend oft kopieren. Im Gegensatz zur Anwendung FAIRTIQ, wird, wie zuvor überlegt, keine zusätzliche Anwendung zur Fahrplanauskunft benötigt. Dafür ist jedoch eine weitere Anwendung zum Ticketkauf nötig. Daraus resultiert, dass der Nutzer selbst für den Kauf und das Entwerten von Fahrscheinen verantwortlich ist. Sollte er eine Fahrt nicht wie angegeben antreten, so muss der Nutzer die Angaben in der App anpassen. Wie geplant ist der Preis für die Fahrten im Voraus bekannt. Die Anwendung benötigt keinen Zugriff auf den Nutzerstandort.
Bei der Optimierung werden, wie angestrebt, alle hier betrachteten Fahrscheine berücksichtigt. Ausgenommen davon war bereits in der Analyse das Ticket2000. Dabei gilt, dass bei Einzelfahrscheinen jeweils die minimale Anzahl an Fahrten beachtet wird, bis sich dieser Fahrschein rentiert. Für Zeitfahrscheine hingegen wird neben diesem Schwellenwert noch berücksichtigt, ob der Preis eines Fahrscheins mit mehr Leistungsumfang geringer ist als der Kauf mehrerer Fahrscheine. Das Analyseergebnis, dass Zeitfahrscheine gegenüber Einzelfahrscheinen zu bevorzugen sind, wurde bei der Optimierung ebenfalls berücksichtigt.
Ein Problem für die Optimierung ist, dass der Nutzer auch Verbindungen außerhalb des Geltungsbereiches des VRRs planen kann, diese können dabei jedoch nicht berücksichtigt werden. Da der Algorithmus einmal zusammengefasste Fahrscheine nicht aufteilt, muss im Falle des VRRs das 48-StundenTicket als letzter Fahrschein bei der Optimierung betrachtet werden. Sonst würde der Algorithmus in einem lokalen Minimum stehen bleiben, vergleiche Abschnitt 5.5.2 – „Zusammenfassen von Fahrscheinen“. Ein weiteres Problem ist die getrennte Betrachtung von Nutzerklassen. So wäre es beispielsweise im Falle von acht Erwachsenen und einem Kind für eine Fahrt der Preisstufe C am kostengünstigsten ein 10erTicket der Preisstufe C zu kaufen und diesen Fahrschein neun Mal zu entwerten. Der Algorithmus hingegen

gibt vor ein 10erTicket der Preisstufe C zu kaufen und acht Mal zu entwerten sowie ein Einzelticket der Preisstufe D für Kinder.
Durch die Betrachtung von alten Einzelfahrscheinen vor der Optimierung mit neuen Zeitfahrscheinen kann der Algorithmus in ein lokales Minimum fallen. So könnten beispielsweise nicht mehr genug Fahrten für einen Zeitfahrschein vorhanden sein. Dies ist nicht kostenoptimal, da die verbliebenen Fahrten zu einem späteren Zeitpunkt verwendet werden können.
Ein großes Problem tritt auf, wenn der Nutzer eine neue zu optimierende Fahrt hinzufügt, welche mit bestehenden Fahrten überlappt, jedoch vor diesen beginnt. In diesem Fall wird die neue Fahrt optimiert, die alten Fahrten werden jedoch gelöscht und nicht mehr optimiert. Die Zuordnung von Fahrscheinen bleibt hingegen bestehen. Die korrekte Lösung wäre hingegen, die neue Fahrt nicht hinzuzufügen und nur die alten Fahrten zu betrachten.

6.2 Optimierungsvergleich

Um zu überprüfen, ob die Optimierung zu einem korrekten Ergebnis kommt, wurde die Liste an Fahrten, welche ich im März unternommen habe, in den August übertragen, um gültige Verbindungsauskünfte zu erhalten. Diese Fahrten wurden dann in der App optimiert und das Ergebnis mit einer händischen Optimierung verglichen. Eine Übersicht über die Fahrten, Preise und Fahrscheine ist in Anhang C.2 zu sehen. Dabei wurden jeweils die Fahrten für einen Tag hinzugefügt und die Fahrten bis zu diesem Zeitpunkt optimiert. Für die händische Optimierung wurden die einzelnen Fahrscheine mit ihren zugehörigen Fahrten notiert sowie der daraus resultierende Gesamtpreis. Von der App hingegen wurde nur der aktuelle Gesamtpreis notiert.

Dabei stellte sich heraus, dass die händische Optimierung einen Preisvorteil von etwa 30 € lieferte, vergleiche Tabelle C.1. Der Preis für die Fahrscheine lag manuell ermittelt bei 139,28 €, während der Algorithmus 172,88 € als Minimum fand. Der Grund für die große Abweichung von etwa 20 % liegt an der hohen Anzahl an Fahrten mit zwei oder mehr Personen. Für diese Anzahl an Personen gibt es günstigere 24- und 48-StundenTicket, welche bei der händischen Optimierung genutzt wurden. Der Algorithmus hingegen nutzt die Fahrscheine nur, wenn für die jeweilige Anzahl an Personen sich der entsprechende Fahrschein lohnt. Jedoch unterscheidet sich die minimale Anzahl an Fahrten für Fahrscheine für eine Person und Fahrscheine für mehrere Personen.

In diesem Versuch zeigte sich eine Schwäche in der Annahme, dass stets die gleiche Person die App nutzt und als erste Person fährt, da dies im Falle der notierten Fahrten nicht immer der Fall war.
Um zu überprüfen, ob die App für eine reisende Person bessere Ergebnisse liefert, wurden erneut die gleichen Fahrten betrachtet. Für jede Fahrt wurde jedoch jedes Mal ein Erwachsener angenommen. Auch für diese Untersuchung wurde die Optimierung tageweise durchgeführt. Bei der manuellen Optimierung wurden die Fahrscheine mit ihren zugehörigen Fahrten und deren Gesamtpreis notiert und für die App der jeweilige Gesamtpreis, vergleiche Tabelle C.2.

In diesem Fall war der Unterschied zwischen beiden Ansätzen deutlich geringer. Zwar konnte in einem Zwischenschritt händisch ein Preisvorteil von 2,80 €(4,49 %)

erzielt werden, vergleiche nach Fahrt 18 in Tabelle C.2. Dieser Vorteil wurde jedoch nicht über alle Fahrten erhalten, am Ende bestand ein Aufpreis von 50 Cent gegenüber der App. Dies resultiert daraus, dass nicht jedes Mal alle Fahrten erneut betrachtet wurden, sondern versucht wurde, möglichst Fahrscheine zusammenzufassen, ohne jedoch alle Fahrten erneut zu betrachten.
Für eine Person wurde ebenfalls verglichen, wie viel im Vergleich zum Kauf von Einzelfahrscheinen gespart werden kann. Dafür wurde zunächst die Anzahl der Fahrten für jede Preisstufe gezählt. Dabei wurden 15 Fahrten der Preisstufe A1, zwei der Preisstufe A3 und acht der Preisstufe B gezählt. Wird für jede Fahrt ein Einzelticket gewählt, spart der Nutzer mit der App 17,01 %. Werden die Einzeltickets zu 4erTickets und 10erTickets gruppiert, beträgt die Ersparnis mit der App im Vergleich 7,31 %.
Abschließend lässt sich sagen, dass die Optimierung für mehrere Personen in der Anwendung kein optimales Ergebnis liefert und der Ansatz überarbeitet werden muss. Das Ergebnis der App für eine Person ist hingegen deutlich besser. Für beide Fälle betrug die Laufzeit der Optimierung unter einer Sekunde in der App von Hand hingegen mehrere Minuten.

6.3 Akzeptanztest

Als Kunden wurden vier Freunde gefragt, ob sie die App testen würden. Zu der App erhielten sie einen Fragebogen siehe Anhang C.1. Ziel des Fragebogens war es, zum einen Schwachstellen in dem Aufbau der Anwendung zu finden und Probleme bei der Handhabung zu finden. Ein wesentlicher Fokus lag darauf, ob die Nutzer die benötigten Fahrscheine aus den jeweiligen Anzeigen korrekt ermitteln konnten anderenfalls muss diese Anzeige noch verbessert werden. Des Weiteren sollte festgestellt werden, ob die benötigten Fahrscheine einfach zu ermitteln sind oder ob dies dem Nutzer eher schwerfällt. Zusätzlich wurden die Nutzer nach der Laufzeit der Optimierung gefragt, dazu wurde, bevor in die Ansicht der zukünftigen Fahrten gewechselt wird, ein Dialog angezeigt, in welchem die Laufzeit in Sekunden und Millisekunden angegeben wurde.

Außerdem wurden die Nutzer zu Problemen im Umgang mit der App sowie nach Verbesserungsvorschlägen befragt.
Die Auswertung der Fragebögen zeigte, dass alle Befragten mit der Laufzeit der Optimierung zufrieden waren. Zwar wurden maximal vier Fahrten optimiert, dennoch dauerte dies auf den vier verschiedenen Geräten (*bq Aquaris X, Pocophone F1, Motorolla G6, OnePlus 5*) weniger als 100 ms. Die Geräte nutzen unter anderem die Android Versionen 9.0 und 10.0, wodurch die erwartete Aufwärtskompatibilität bestätigt wurde.
Ein Problem war, dass vor allem Freunde, die keine Zwischenstände der App kannten, das Symbol zum Speichern der Fahrt als nicht intuitiv empfanden. Eine weitere Ursache war, dass sie nicht wussten, dass sie Fahrten speichern können. Zwar werden die Buttons in der Übersicht der zu optimierenden Fahrten als nicht intuitiv bezeichnet, jedoch waren die Funktionen aus dem Ausschlussprinzip herleitbar. Des Weiteren wurde die Bedienung nach einem erfolgreichen Durchlauf als deutlich intuitiver und einfacher empfunden.

Kapitel 7

Fazit

Zunächst wird die Entwicklung der App zusammengefasst werden. Dann werden die Probleme und Schwierigkeiten der App gesammelt. Abschließend wird ein Ausblick auf mögliche Verbesserungs- und Ausbaumöglichkeiten gegeben.

7.1 Zusammenfassung der Entwicklung

Im Rahmen der Arbeit wurde ein Algorithmus zur Kostenoptimierung von Fahrscheinen entwickelt. Betrachtet wurde dafür beispielhaft der VRR, dessen Tarifsystem und Preisgestaltung für mehrere Personen und Fahrten undurchsichtig ist. Dabei setzt die App eine feste Nutzergruppe voraus. Für die Verwaltung von Fahrscheinen ist der Nutzer verantwortlich.
Die Optimierung wurde in eine App eingebunden, in welcher der Nutzer die Fahrten zunächst plant, bevor diese optimiert werden. Sollte der Nutzer eine Fahrt doch nicht wie angegeben antreten, so muss er die Fahrt editieren oder löschen. Da der Fokus der Arbeit auf dem Entwurf und der anschließenden Realisierung der Optimierung lag, ist das Layout und Design des User Interfaces zu vernachlässigen.
Als Basis für die im Rahmen dieser Arbeit entwickelte App wurde Öffi gewählt. Neben der Optimierung wurden weitere Funktionen und Optionen hinzugefügt, die aus der Betrachtung bestehender Anwendungen zur Fahrplanauskunft und Fahrscheinberatung resultieren.
Entsprechend dem angepassten inkrementellen Entwicklungsmodell wurde als erstes die Planung von Fahrten, dann die Optimierung mit Einzelfahrscheinen, Zeitfahrscheinen und deren Kombination realisiert. Dafür wurde zunächst jede Komponente entworfen, implementiert und zum Schluss getestet.
Die eingebundene Optimierung wurde danach mit einer manuellen Optimierung für reale Fahrten verglichen. Abschließend wurde mithilfe einer kleinen Nutzergruppe überprüft, ob die Anwendung intuitiv zu bedienen und leicht verständlich ist.

7.2 Resümee

Wie in dem Vergleich zwischen realisierter Anwendung und geplanten Anforderungen (vergleiche Abschnitt 6.1) festgestellt wurde, sind nicht alle geplanten Funktionen

realisiert. Bei den fehlenden Funktionen handelt es sich jedoch hauptsächlich um Komfortfunktionen, wie beispielsweise die Schritt-für-Schritt Navigation. Die fehlenden Funktionen bedeuten keinen Informationsverlust für den Nutzer.
Das Planen von Fahrten, damit diese optimiert werden können, konnte wie im Vorhinein festgelegt realisiert werden. Da Öffi nur um die Optimierung erweitert wurde, konnten keine weiteren Optionen realisiert werden. Der Nutzer kann beispielsweise die berücksichtigten Verkehrsmittel und unter drei Stufen der Barrierefreiheit wählen. Dabei besteht jedoch Verbesserungspotenzial im Layout der App.

Bei der Planung von Fahrten mit Zwischenhalt muss der Nutzer hingegen manuell zwei Fahrten planen. Die Option, einen Zwischenhalt anzugeben, musste entfernt werden, weil die für die Optimierung benötigte Preisstufe in diesen Fällen stets unbekannt ist. Das gleiche Problem tritt auf, wenn der Nutzer eine Verbindung mit Fernverkehr wählt. Eine zukünftige Lösung wäre es, diesen auszuschließen.
Das Ziel der Arbeit, einen Algorithmus zur Kostenoptimierung von Fahrscheinen zu entwerfen, ist gelungen.

Der Vergleich von manueller Optimierung und in die App eingebundener Optimierung (vergleiche Abschnitt 6.2) zeigte, dass der Ansatz für mehrere Personen überarbeitet werden muss. Für diesen Fall liefert die Optimierung der App kein zufriedenstellendes Ergebnis, da die preisliche Abweichung sehr groß ist. Die geringe Laufzeit ist in diesem Fall kein Ausgleich. Die Optimierung für eine Person hingegen erzielte in der Anwendung für alle Fahrten nicht nur ein besseres Ergebnis, sondern die preisliche Abweichung insgesamt war geringer. Des Weiteren ist der Kaufvorschlag der App besser als der Kauf von Einzelfahrscheinen. Neben dem besseren Ergebnis war auch die Laufzeit deutlich geringer. Zusätzlich ist bei dem Vergleich zu berücksichtigen, dass vor der manuellen Optimierung eine Einarbeitung in das Tarifsystem erfolgen muss.
In dem Vergleich der Optimierung zeigte sich, dass die Annahme, dass stets die gleichen Personen die App nutzen, in der Realität nicht zutreffend ist. Da der Nutzer selbst für den Kauf von Fahrscheinen verantwortlich ist, muss er darauf hingewiesen werden, dass er die Fahrscheine jeweils nur für die nächste Fahrt und erst kurz vor Fahrtantritt kaufen sollte. Dies liegt daran, dass alle zukünftigen Fahrscheine bei der nächsten Optimierung entfernt werden und die Fahrten neu optimiert werden.
Bei der Optimierung werden das Tarifgebiet und -system des exemplarisch betrachteten Verkehrsverbundes berücksichtigt sowie dessen Vorschriften eingehalten. Reist der Nutzer mit mehreren Personen, erhält jeder einen Fahrschein zugewiesen.
Obwohl der Fokus der Arbeit auf dem Entwurf der Optimierung lag, ist laut der Nutzerbefragung die App nach dem ersten Benutzen intuitiv zu bedienen.
Insgesamt wurde der Aufwand unterschätzt. Insbesondere die Optimierung mit Zeitfahrscheinen stellte sich als komplexer als erwartet heraus. Die Informationen zu dem Tarifsystem waren in verschiedenen Dokumenten verteilt und sind nicht kundenfreundlich in einer Broschüre auffindbar.

7.3 Ausblick

Um die Optimierung zu verbessern und zu beschleunigen, bietet es sich an, diese auf einen Server auszulagern. Wenn die Optimierung nicht mehr auf dem Handy

läuft, kann, statt mit einer Anzahl an Nutzern, mit konkret ausgewählten Nutzern für jeden Nutzer eine Liste an Fahrscheinen erstellt werden. Dies würde die Einschränkung des festen Nutzerkreises aufheben, sowie für eine rechtlich korrekte Nutzung der Fahrscheine sorgen. Dabei würden keine persönlichen Daten benötigt werden, eine zufällige Nutzernummer reicht dafür aus. In diesem Schritt könnte der dritte Optimierungsansatz, eine Fahrt an ihren Umstiegspunkten aufzuteilen, realisiert werden. Auch der ursprüngliche Plan, dass der Nutzer den Gültigkeitsbereich auswählt, statt das erste maximale Gebiet zu wählen, könnte dann realisiert werden.

Eine zusätzliche Ausbaumöglichkeit wäre die Ausweitung der Optimierung auf andere Regionen. Die Optimierung könnte in dem Sinne verbessert werden, dass der Nutzer Vorschläge erhält, einzelne Fahrten zu verschieben, damit diese in den Geltungsbereich anderer Zeitfahrscheine fallen und er weniger oder günstigere Fahrscheine benötigt. Davon würde die Betrachtung von Zeitfahrscheinen profitieren, weil so gegebenenfalls nur noch einzelne Fahrtabschnitte mit Einzelfahrscheinen abgedeckt werden müssen, statt der ganzen Fahrt.

Eine weitere Ausbaumöglichkeit wäre es, den Verkauf der Fahrscheine in die Anwendung zu integrieren. Ebenso könnte eine Editieren und Löschen Funktion für Fahrscheine hinzugefügt werden, genauso wie eine manuelle Eingabe von Fahrscheinen. Zusätzlich könnten weitere Fahrscheine berücksichtigt werden, wie etwa die Tickets der Deutschen Bahn, die *Welcome Card Ruhr* oder das *Zusatzticket*, welches im Rahmen der Arbeit gar nicht betrachtet wurde. Zudem könnten Fahrschein-Abonnements berücksichtigt werden, sodass für Fahrten innerhalb des Geltungsbereiches des Fahrscheins dieser genutzt wird, für Fahrten außerhalb jedoch eine Optimierung durchgeführt wird. Des Weiteren könnten die fehlenden Funktionen und Designs umgesetzt werden.

Aus der Auswertung der Nutzerbefragung lassen sich ebenfalls weitere Verbesserungsmöglichkeiten entnehmen. Ein Vorschlag ist zum Beispiel die Funktionen editieren, löschen und duplizieren nicht nur in der Übersicht der Fahrten anzuzeigen, sondern auch in der Detailansicht der jeweiligen Fahrten. Ebenfalls wurde eine Überarbeitung des Layouts vorgeschlagen, beispielsweise durch ein permanentes Menü im unteren Bildschirmbereich. Damit könnte das Problem, den Unterschied zwischen den Funktionen „einzelne Fahrt“ und „mehrere Fahrten“ zu erkennen, gelöst werden. Ein Lösungsvorschlag war keine getrennte Betrachtung dieser Ansichten, sondern stattdessen eine Checkbox, über die der Nutzer angibt, ob die Fahrt optimiert werden soll oder nicht.

Um die Funktionsweise der App zu erläutern, würde es sich anbieten, beim ersten Starten ein Tutorial anzuzeigen. Dadurch würde der Nutzer die verschiedenen Funktionen der App sehen und die Funktionen der einzelnen Buttons, welche für die Testgruppe nicht vollständig erkennbar waren, verstehen.

Anhang A

Analyse

A.1 Vergleich bestehender Anwendungen

Anwendung	Kriterien		
	Barrierefreiheit	Gehtempo	Verkehrsmittel
VRR-App	✓	✓	ÖPNV: ✓; Fahrrad: ✓; Sharing-Angebote: ✗
MUTTI	✓	✓	ÖPNV: ✓; Fahrrad: ✓; Sharing-Angebote: ∼
MyHannover	✗	✗	ÖPNV: ✓; Fahrrad: ✗; Sharing-Angebote: ✗
Jelbi	✗	✗	ÖPNV: ✓; Fahrrad: ✗; Sharing-Angebote: ✓
DB Navigator	✗	✗	ÖPNV: ✓; Fahrrad: ✗; Sharing-Angebote: ✗
Moovit	✗	✗	ÖPNV: ✓; Fahrrad: ✓; Sharing-Angebote: ✓
Citymapper	✗	✗	ÖPNV: ✓; Fahrrad: ✓; Sharing-Angebote: ✓
Öffi	∼	✓	ÖPNV: ✓; Fahrrad: ✗; Sharing-Angebote: ✗

Anwendung	Kriterien		
	Barrierefreiheit	Gehtempo	Verkehrsmittel

Tabelle A.1: Vergleich aller betrachteten Anwendungen bezüglich ihrer Optionen für die Routenoptimierung und berücksichtigte Verkehrsmittel.

Anwendung	Kriterien bei der Planung von Fahrten			
	Auswahl von Haltestellen	Wahl des Zeitpunkts	Zwischen-halte	Komfort-Funktionen
VRR-App	✓	✓	✗	✓
MUTTI	✓	✓	✓	✓
MyHannover	✓	∼	✓	✓
Jelbi	✓	∼	✗	✗
DB Navigator	✓	✓	✓	✓
Moovit	✓	∼	✗	✓
Citymapper	✓	∼	✗	✗
Öffi	∼	✓	✓	✓

Tabelle A.2: Vergleich der Anwendungen, wie übersichtlich die Planung einer Fahrt ist.

Anwendung	Kriterien bei der Auswahl einer Fahrt			
	grafische Ansicht	listen Ansicht	unterschiedliche Farben für einzelne Linien	früher / später
VRR-App	✓	✓	∼	✓
Mutti	✓	✓	∼	✓
MyHannover	✗	✓	∼	✓
Jelbi[1]	✗	✓	✓[2]	✗
DB Navigator	✗	✓	✗	✓
Moovit[1]	✗	✓	✓[3]	✗
Citymapper[1]	✗	✓	✓[3]	✗
Öffi	✓	✗	∼	✓

Tabelle A.3: Vergleich aller hier betrachteten Anwendungen bezüglich der Darstellung von möglichen Verbindungen.
[1] Nur die ÖPNV Auskunft betrachtet; [2] einzelne Farben für U-Bahn und S-Bahn Linien; [3] einzelne Farben für S-Bahn Linien

Anwendung	Kriterien bei der Auswahl einer Fahrt					
	Anzahl Umstiege	Startzeit	Endzeit	Dauer	Preisstufe	Umstiegs-zeiten
VRR-App	∼	✓	✓	✓	∼	∼
Mutti	∼	✓	✓	✓	∼	∼
myHannover	✓	✓	✓	✓	✻[2]	∼
Jelbi[1]	∼	∼	✓	✓	∼	✓
DB Navigator	✓	✓	✓	✓	✗[3]	✗
Moovit[1]	∼	∼	✗	✓	✗	✗
Citymapper[1]	∼	∼	✗	✓	✗	✗
Öffi	∼	✓	✓	✗	∼	∼

Tabelle A.4: Vergleich der Anwendungen in der Darstellung einer einzelnen möglichen Verbindung.
[1] Nur die ÖPNV Auskunft betrachtet; [2] in der Stadt Hannover gilt nur eine Tarifstufe, vgl. [48]; [3] Auskunft über die Preisstufe erhält der Nutzer in der Kaufansicht

Anwendung	Kriterien für die Detailansicht einer Fahrt				
	Übersicht	Zwischen-halte	Verspätungen / Probleme	Navi-gation	Zeiten einer Linie
VRR-App	∼	✓	✓	✓	✓
Mutti	✓	∼	✓	✓	✓
MyHannover	∼	✓	✓	✓	✓
Jelbi[1]	∼	✓	✻	∼	✓
DB Navigator	✓	✓	✓	✓	✓
Moovit[1]	∼	✓	✻	✓	∼
Citymapper[1]	✗	✓	✻	✓	✓
Öffi	✓	✓	✓	✗	✓

Tabelle A.5: Vergleich aller betrachteten Anwendungen bezüglich der Detailansicht einer Fahrt.
[1] Nur die ÖPNV Auskunft betrachtet

A.2 Ersparnisse Zeitfahrscheine

Preisstufe A1		
Fahrschein	Anzahl Fahrten	Ersparnis in %
7-TageTicket	11	10,70
Ticket1000	31	0,42
Ticket2000	35	1,70
Preisstufe A2		
7-TageTicket	12	0,12
Ticket1000	33	1,95
Ticket2000	37	2,56
Preisstufe A3		
7-TageTicket	13	6,64
Ticket1000	34	1,37
Ticket2000	38	12,08
Preisstufe B		
7-TageTicket	8	5,00
Ticket1000	24	2,11
Ticket2000	26	1,34
Preisstufe C		
7-TageTicket	11	4,02
Ticket1000	16	7,30
Ticket2000	16	0,69
Preisstufe D		
7-TageTicket	11	0,55
Ticket1000	17	7,25
Ticket2000	18	4,29

Tabelle A.6: Ersparnis und minimale Anzahl an Fahrten für die jeweiligen Preisstufen.

Anhang B

Optimierungsentwürfe

B.1 Funktionsentwürfe

Algorithmus 10 Optimierung für die Preisstufe C - Teil 1

Parameter: Fahrten die zu optimieren sind, mögliche Fahrscheine
Rückgabewert: Liste an optimalen Fahrscheinen der Preisstufe B

```
1: Funktion FAREZONE_C(tripItems, timeTickets)
2:     tripsC ← COLLECTTRIPS(tripItems, C)
3:     if LENGTH(tripsC) = 0 then return
4:     end if
5:     farezoneMap ← LOADFAREZONMAP
6:     ticketsPerRegion                ▷ HashMap: Regionen ID - zugeordnete Tickets
7:     for ticketIndex ← 1 to LENGTH(timeTickets) do
8:         currentTicket ← timeTickets[ticketIndex]
9:         minNumTrips ← currentTicket.minNumTrips[C]
10:        while true do
11:            if LENGTH(tripsC) < minNumTrips then
12:                break
13:            end if
14:            ASSIGNTRIPSTOREGION(farezoneMap, tripsC)
15:            bestIntervallPerRegion          ▷ HashMap: ID der Region
    - Startindex des maximalen Intervalls, Anzahl Fahrten im maximalen Intervall,
    Fahrten in der Region
16:            for id in farezoneMap do
17:                maxIndex, maxNumTrips, tripsInRegion ← CALCULATEMAX-
    TRIPS(id, currentTicket, farezoneMap)
18:                bestIntervallPerRegion.id ← maxIndex, maxNumTrips,
    tripsInRegion
19:            end for
20:            maxID ← MAX(bestIntervallPerRegion)
                                    ▷ Region, mit dem maximalen maxNumTrips
21:            maxNum ← maxID.maxNumTrips
22:                                          ▷ Weiter in Algorithmus 11
```

Algorithmus 11 Optimierung für die Preisstufe C - Teil 2

```
23:             if maxNum ≥ minNumTrips then
24:                 newTicket ← newTicketToBuy(currentTicket, C)
25:                 ticketsPerRegion.maxID.add(newTicket)
26:                 region ← createFarezoneSet(farezoneMap.maxID)
27:                 newTicket.setValidFarezones(maxID, region)
28:                 maxStartIndex ← maxID.maxIndex
29:                 maxTrips ← maxID.tripsInRegion
30:                 newTicket.addTrips(maxTrips, maxStartIndex, maxNum)
31:                 deleteTrips(tripItems, maxTrips, maxStartIndex, maxNum)
32:             else
33:                 break
34:             end if
35:             tripsC ← collectTrips(tripItems, C)
36:         end while
37:         for id in ticketsPerRegion do
38:             tickets ← ticketsPerRegion.farezone
39:             if length(tickets)> 1 then
40:                 sort(tickets)              ▷ Nach der ersten Abfahrtszeit sortieren
41:                 sumUpTickets(tickets, timeTickets, ticketIndex+1, C)
42:             end if
43:             checkTicketForOtherTrips(tripItems, tickets)
44:         end for
45:         tripsC ← collectTrips(tripItems, C)
46:     end for
47:     result                                  ▷ Liste mit allen Fahrscheinen
48:     for id in ticketsPerRegion do
49:         currentTickets ← ticketsPerRegion.farezone
50:         result.add(currentTickets)
51:     end for
52:     return result
53: end Funktion
```

Algorithmus 12 Optimierung der Fahrten innerhalb eines Tarifgebiets

Parameter: HashMap der zu optimierenden Fahrten, mögliche Tickets, Preisstufe
Rückgabewert: Fahrscheine der jeweiligen Preisstufe

```
1: Funktion TARIFGEBIETOPTIMISATION(tarifgebietTrips, allTrips, timeTickets, farezone)
2:     result                                          ▷ Liste aller benötigten Fahrten
3:     for tarifgebiet in tarifgebietTrips do
4:         temporaryResult   ▷ Fahrscheine für das aktuell betrachtete Tarifgebiet
5:         currentTrips ← tarifgebietTrips.tarifgebiet
6:         SORT(currentTrips)                          ▷ Sortiert nach der Abfahrtszeit
7:         for ticketIndex ← 1 to LENGTH(timeTickets) do
8:             currentTicket ← timeTickets[ticketIndex]
9:             minNumTrips ← currentTicket.minNumTrips[farezone]
10:            while LENGTH(currentTrips) ≥ minNumTrips do
11:                maxIndex, maxNumTrips ← BESTTICKETINTERVALL(currentTrips, currentTicket)
12:                if maxNumTrips < minNumTrips then
13:                    break
14:                end if
15:                newTicket ← new TICKETTOBUY(currentTicket, farezone)
16:                newTicket.ADDTRIPS(currentTrips, maxIndex, maxNumTrips)
17:                newTicket.SETVALIDFAREZONES(tarifgebiet.id, tarifgebiet)
18:                temporaryResult.ADD(newTicket)
19:                REMOVETRIPS(currentTrips, maxIndex, maxNumTrips)
20:            end while
21:            if LENGTH(temporaryResult)> 1 then
22:                SORT(temporaryResult)   ▷ nach der ersten Abfahrtszeit sortieren
23:                SUMUP(temporaryResult, timeTickets, ticketIndex + 1, farezone)
24:            end if
25:            CHECKTICKETSFOROTHERTRIPS(allTrips, temporaryResult)
26:        end for
27:        result.ADD(temporaryResult)
28:    end for
29:    return result
30: end Funktion
```

Algorithmus 13 Intervall mit dem höchsten Ticketpreis suchen - Teil 1

Parameter: Liste an alten Fahrscheinen, Ticket
Rückgabewert: Startindex und Anzahl Fahrten des maximalen Intervalls sowie Summe der einzelnen Ticketpreise des Intervalls

```
 1: Funktion BESTTICKETINTERVALL(oldTickets, timeTicket)
 2:     maxIndex ← ∞
 3:     maxNumTickets ← 0
 4:     maxPrice ← 0
 5:     for i ← 1 to LENGTH(tripItems) do
 6:         currentTicket ← oldTickets[i]
 7:         if ISNEWVALIDTICKET(currentTicket, timeTicket) then
 8:             maxIndex ← i
 9:             maxNumTickets ← 1
10:             break
11:         end if
12:     end for
13:     if maxIndex = ∞ then return
14:     end if
15:     for i ← maxIndex to LENGTH(oldTickets) do
16:         currentTicket ← oldTickets[i]
17:         if notISNEWVALIDTICKET(currentTicket, timeTicket) then
18:             continue
19:         end if
20:         farezone ← currentTicket.farezone
21:         counter ← 1
22:         price ← currentTicket.price[farezone]
23:         for k ← i + 1 to LENGTH(oldTickets) do
24:             thisTicket ← oldTickets[k]
25:             if ISNEWVALIDTICKET(thisTicket) then
26:                 endTime ← thisTicket.lastArrivalTime
27:                 startTime ← currentTicket.firstDepartureTime
28:                 if endTime − startTime ≤ ticket.maxDuration then
29:                     counter ← counter + 1
30:                     price ← price + thisTicket.price[farezone]
31:                 else
32:                     break
33:                 end if
34:             end if
35:         end for
36:                                                  ▷ Weiter in Algorithmus 14
```

Algorithmus 14 Intervall mit dem höchsten Ticketpreis suchen - Teil 2

```
37:         if price > maxPrice then
38:             maxPrice ← price
39:             maxIndex ← i
40:             maxNumTickets ← counter
41:         end if
42:     end for
43:     return maxIndex, maxCounter, maxPrice
44: end Funktion
```

Algorithmus 15 Verwenden von Fahrscheinen für andere Fahrten

Parameter: Fahrten ohne Fahrscheine, Fahrscheine für andere Fahrten

```
 1: Funktion checkTicketsForOtherTrips(tripItems, tickets)
 2:     for ticketToBuy in tickets do
 3:         timeTicket ← ticketToBuy.ticket
 4:         for i ← 1 to length(tripItems) do
 5:             currentTrip ← tripItems[i]
 6:             if currentTrip.farezone > ticketToBuy.farezone then
 7:                 break                    ▷ Fahrten sind nach Preisstufen sortiert
 8:             else if not timeTicket.isValidTrip(currentTrip) then
 9:                 continue
10:             else if not tripInRegion(currentTrip, ticketToBuy) then
11:                 continue
12:             end if
13:             startTime ← Min(currentTrip.firstDepartureTime, ticketToBuy.firstDepartureTime)
14:             endTime ← Max(currentTrip.lastArrivalTime, ticketToBuy.lastArrivalTime)
15:             if endTime − startTime < timeTicket.maxDuration then
16:                 ticketToBuy.addTrip(currentTrip)
17:                 sort(ticketToBuy.trips)
18:             end if
19:         end for
20:     end for
21: end Funktion
```

B.2 Vollständiges Klassendiagramm

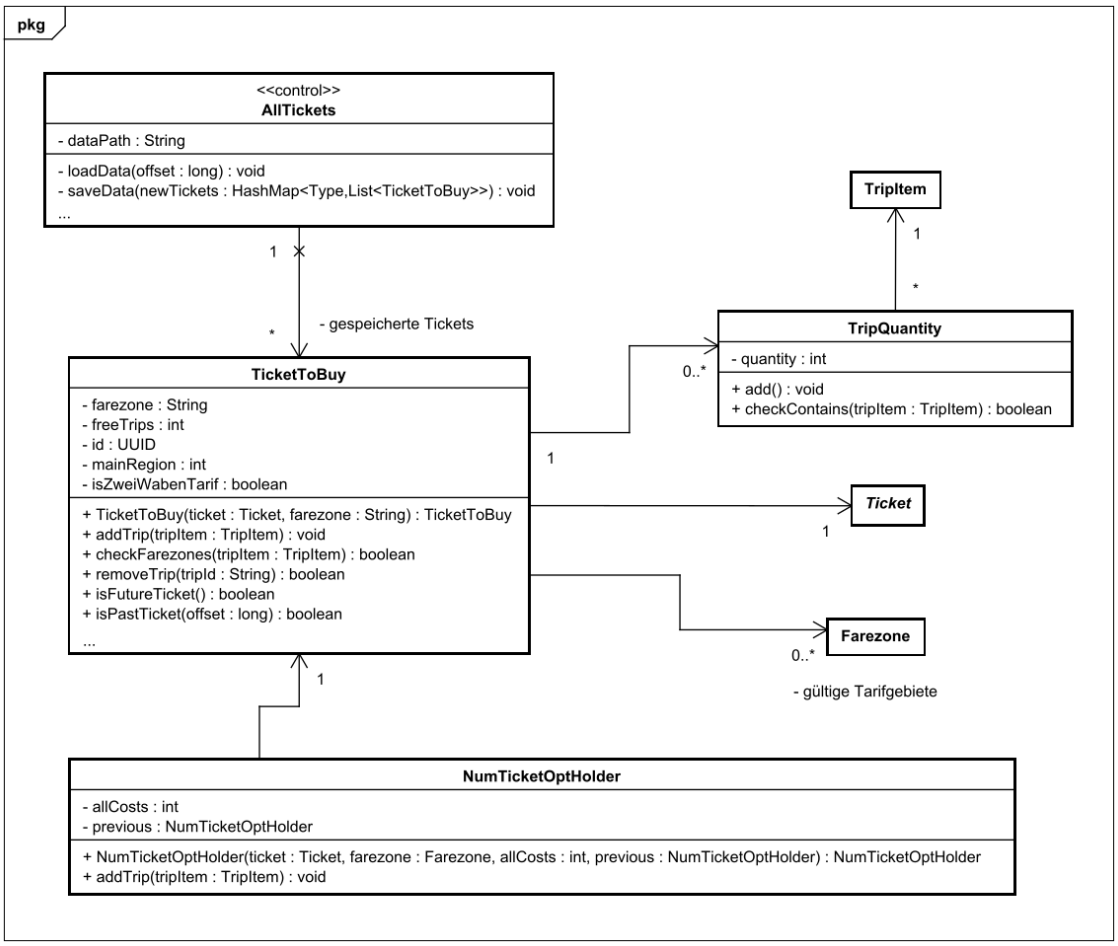


Abbildung B.1: Fahrscheine im globalen Zusammenhang.

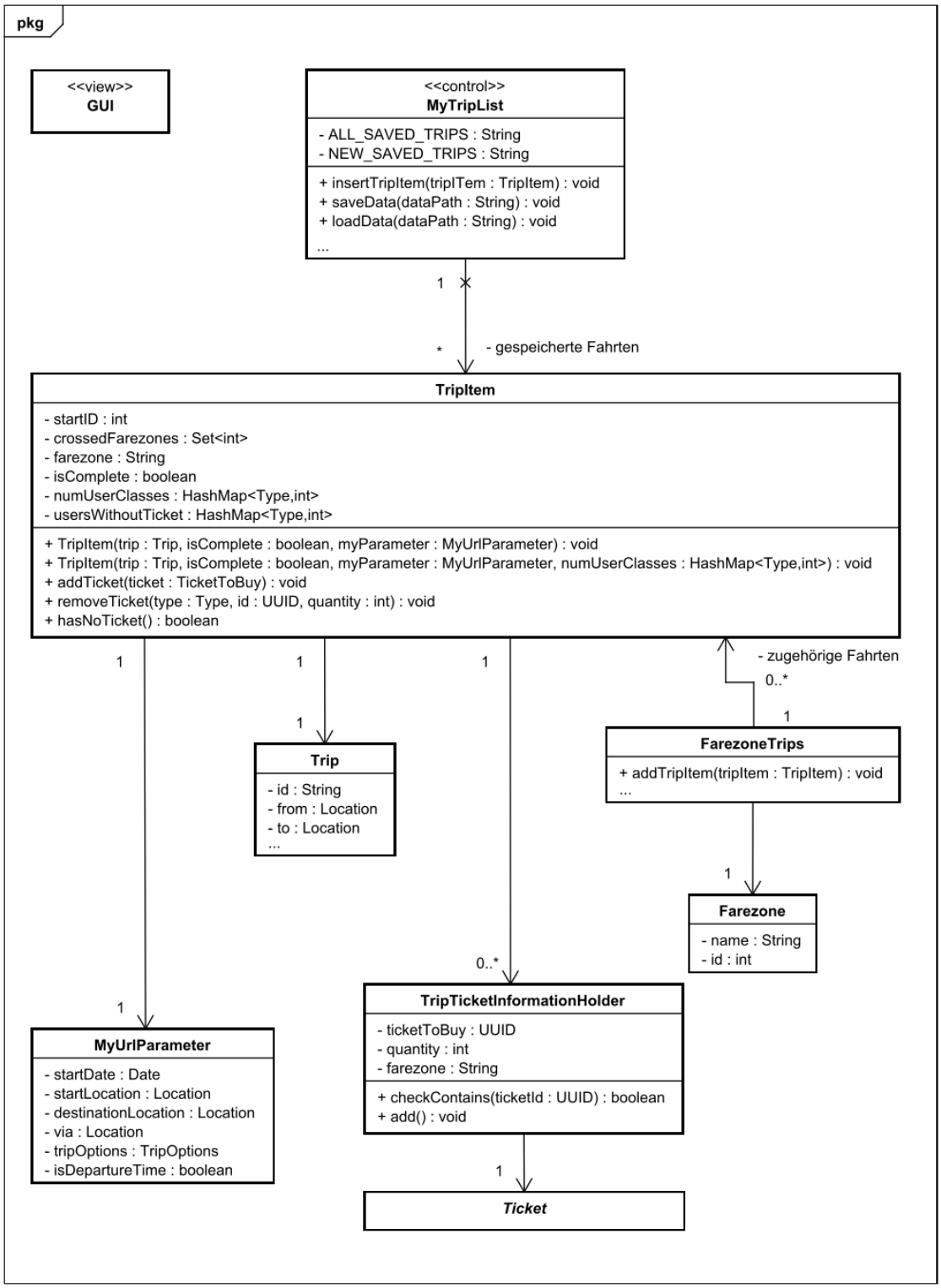


Abbildung B.2: Fahrten im globalen Zusammenhang.

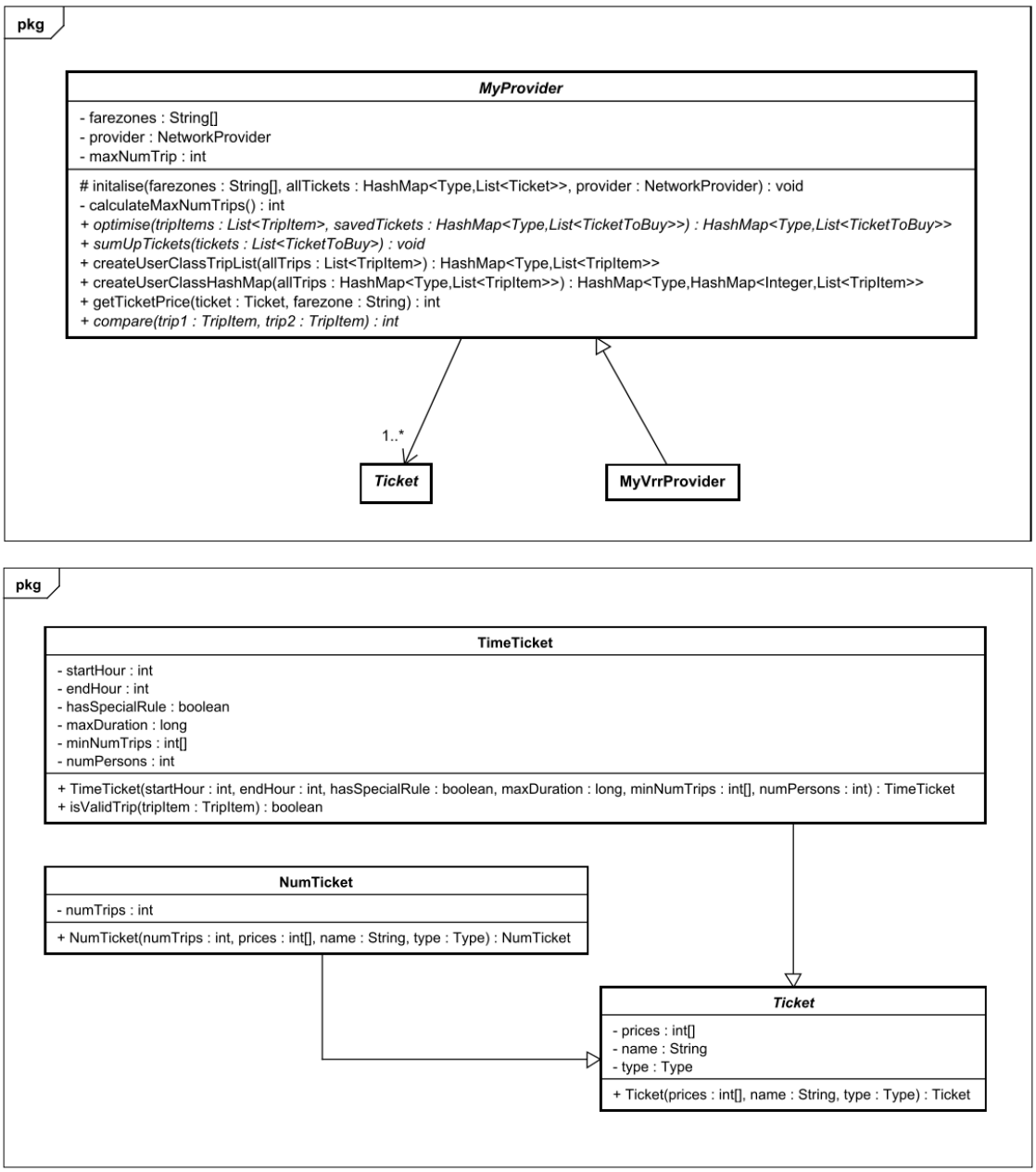
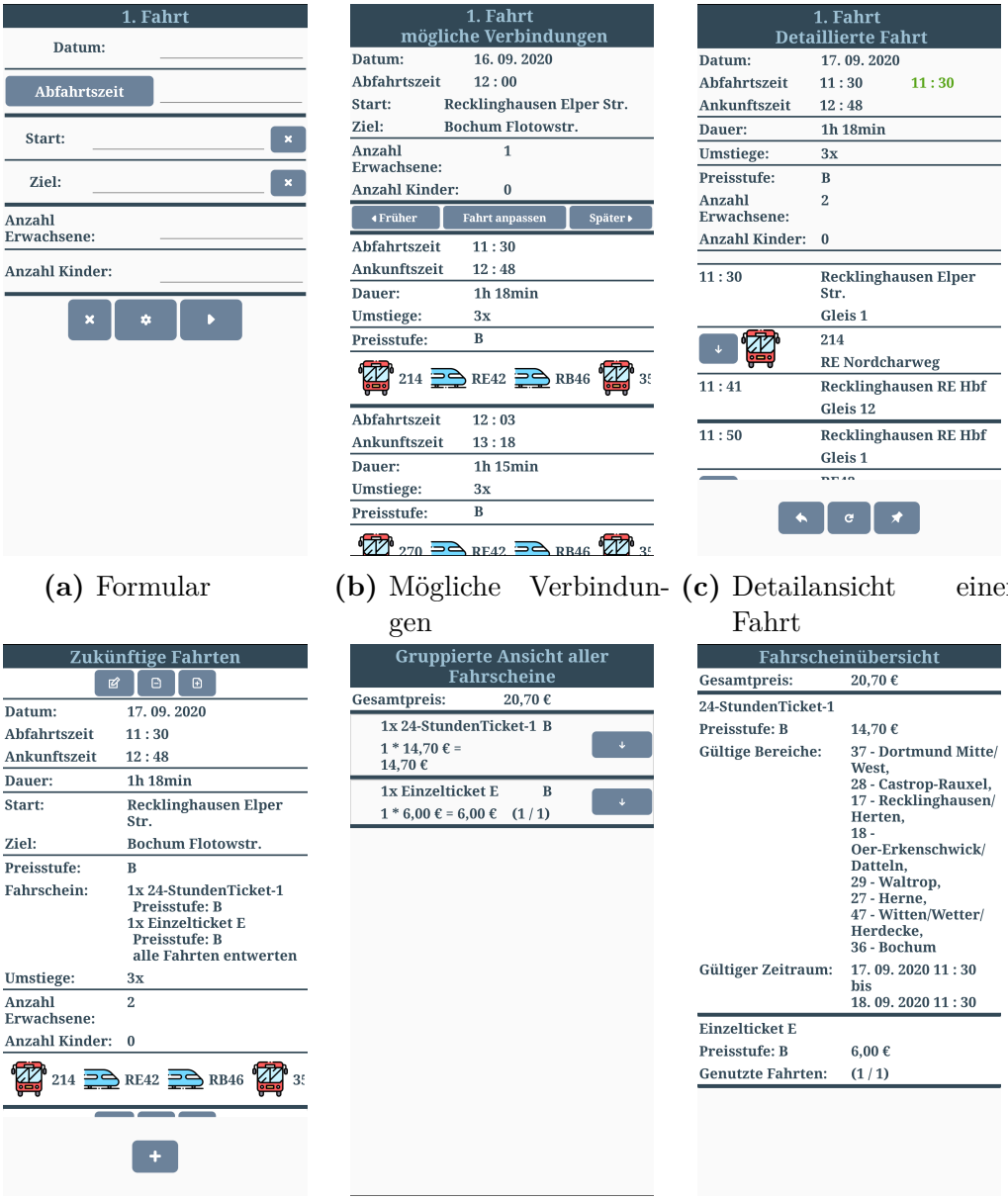


Abbildung B.3: Verbliebene Klassen im globalen Zusammenhang.

B.3 Realisiertes Layout



(a) Formular **(b)** Mögliche Verbindungen **(c)** Detailansicht einer Fahrt

(d) Übersicht über alle Fahrten **(e)** Gruppierte Fahrscheinübersicht **(f)** Anzeige der einzelnen Fahrscheine

Abbildung B.4: Realisierte Ansichten.

Anhang C

Testen

C.1 Fragebogen für den Akzeptanztest

Aufgabe 1:
Bitte lösche zuerst alle vorherigen Daten der App, bevor Du mit der Beantwortung der Fragen beginnst.

Frage 1.1:
Welches Handy benutzt Du?

Antwort (Hersteller, Modell):

Aufgabe 2:
Plane eine einzelne Fahrt am nächsten Dienstag.
Du willst um $11^{\underline{15}}$ von**Dortmunder Hbf** nach **Essen Steele S** fahren.

Frage 2.1:
Wie eindeutig fandest Du, welche Buttons Du drücken musstest ?

Antwort (Skala: 1 (gut) - 5 (schlecht)):

Frage 2.2:
Fandest Du die Abfolge der Schritte und Ansichten logisch?

Antwort (Skala: 1 (gut) - 5 (schlecht)):

Frage 2.3 (?, K, A1, A2, A3, B, C, D):
Welche Preisstufe hat die Fahrt?

Antwort:

Frage 2.4:
Welche Farbe hat die Fahrt in der Liste der geplanten Fahrten?

Antwort:

Frage 2.5:
Hattest Du Probleme beim bedienen der App? Wenn ja, welche?

Antwort:

Aufgabe 3:
Für die Fahrten

- Nächste Woche Donnerstag, Ankunftszeit $12^{\underline{00}}$
  Herten Mitte → **Bochum Hbf**, 2 Erwachsene
- Nächste Woche Freitag, Abfahrtszeit $11^{\underline{00}}$
  Dortmund Hbf → **Bochum Hbf**, 2 Erwachsene, nur *S-Bahnen* erlaubt

besitzt Du keine gültigen Fahrscheine. Plane die Fahrten und beantworte die folgenden Fragen:

Frage 3.1:
Wie intuitiv fandest Du die Bedienung der App?

Antwort (Skala: 1 (gut) - 5 (schlecht)):

Frage 3.2:
Fandest Du die Abfolge der Schritte und Ansichten logisch?

Antwort (Skala: 1 (gut) - 5 (schlecht)):

Frage 3.3:
Empfandest Du die Laufzeit der Optimierung zu lang?
Notiere bitte zusätzlich die angezeigte Zeit.

Antwort (Zu lang, Geht so, Genau richtig, Weiß nicht):

Frage 3.4:
Welche Fahrscheine benötigst Du?

Antwort:

Aufgabe 4:
Bitte plane nun die folgenden Fahrten, für die Du ebenfalls keine gültigen Fahrscheine besitzt:

- In 14 Tagen, Abfahrtszeit $11^{\underline{00}}$
  Gelsenkirchen Hbf → **Düsseldorf Hbf**, 2 Erwachsene, kein Fernverkehr der DeutschenBahn

- In 15 Tagen, Abfahrtszeit $8^{\underline{00}}$
  Düsseldorf Hbf → **Dortmund Hbf**, 1 Erwachsener
- In 15 Tagen, Ankunftszeit $11^{\underline{00}}$
  Dortmund Hbf → **Gelsenkirchen Hbf**, 1 Erwachsener

Frage 4.1:
Welche Fahrscheine benötigst Du für die Fahrt von **Gelsenkirchen** nach **Düsseldorf**?

Antwort:

Frage 4.2:
Welche Fahrscheine benötigst Du für alle Fahrten, die Du bis jetzt geplant hast?

Antwort:

Frage 4.3:
Wie einfach fandest Du die vorherige Frage zu beantworten?

Antwort (Skala: 1 (gut) - 5 (schlecht)):

Aufgabe 5:
Bitte lösche nun die Fahrt, die in 15 Tagen von **Düsseldorf** nach **Dortmund** führt.

Frage 5.1:
Wie intuitiv fandest Du die Bedienung?

Antwort (Skala: 1 (gut) - 5 (schlecht)):

Frage 5.2:
Ändert sich dadurch die Liste der Fahrscheine?

Antwort (Ja - Nein - Weiß nicht):

Aufgabe 6:
Editiere nun die Fahrt, welche in 15 Tagen von **Dortmund** nach **Gelsenkirchen** fährt. Erhöhe dabei die Anzahl reisender Personen **jeweils** um eins.

Frage 6.1:
Wie intuitiv fandest Du die Bedienung?

Antwort (Skala: 1 (gut) - 5 (schlecht)):

Frage 6.2:
Fandest Du die Reihenfolge der Ansichten logisch?

Antwort (Skala: 1 (gut) - 5 (schlecht)):

Frage 6.3:
Empfandest Du die Laufzeit der Optimierung als zu lange? Notiere bitte zusätzlich die angezeigte Zeit.

Antwort (Zu lang, Geht so, Genau richtig, Weiß nicht):

Aufgabe 7:
Sonstiges

Frage 7.1:
Hattest Du Probleme beim bedienen der App?

Antwort:

Frage 7.2:
Hast Du Anmerkungen oder Verbesserungsvorschläge?
(Bitte konzentriere Dich dabei vor allem auf die Funktionsweise der App sowie die Reihenfolge der Ansichten, und weniger auf das Layout!)

Antwort:

C.2 Überprüfung der Optimierung

	Datum	Uhrzeit	Start	Ziel	Preisstufe	# Personen
1	12.08.20	18^{30}	Herten Elisabethstr.	Herten Josefstr.	A1	3 E
2	12.08.20	19^{00}	Herten Josefstr.	Herten Elisabethstr.	A1	3 E
	Manuell: App:	$9,57€$ $9,57€$	3x Happy Hour Ticket	1,2		
3	13.08.20	12^{30}	Herten Elisabethstr.	Herten Uhlandstr.	A1	1E
4	13.08.20	15^{00}	Herten Elisabethstr.	Herten Uhlandstr.	A1	1 E
5	13.08.20	19^{00}	Herten Uhlandstr.	Herten Elisabethstr.	A1	2 E
	Manuell: App:	$7,20€$ $6,38€$ $2,80€$ $\sum$ $16,38€$ $\sum$ $13,58€$	1x 24-StundenTicket 1 Erwachsener, Preisstufe A 2x HappyHourTicket Einzelticket A2	1,2,3,4 1,2 5		
6	15.08.20	9^{33}	Herten Schlägel- und Eisenstraße	Herten Elisabethstr.	A1	1 E
7	15.08.20	10^{00}	Herten Elisabethstr.	Herne Archälogiemuseum / Kreuzkirche	B	3 E
8	15.08.20	16^{00}	Herne Archäologiemuseum / Kreuzkirche	Herten Gertrudenstr.	B	3 E

9	15.08.20	17^{00}	Herten Gertrudenstr.	Herten Schlägel- und Eisenstraße	A1	3 E
10	15.08.20	23^{00}	Herten Schlägel- und Eisenstraße	Herten Elisabethstr.	A1	2 E
	Manuell:	16,38€	Vorherige Fahrscheine			
		22,70€	24-StundenTicket 3 Erwachsene, Preisstufe B	6,7,8,9,10		
		$\sum$ 39,08€				
	App:	$\sum$ 68,78€				
11	19.08.20	8^{30a}	Herten Elisabethstr.	Herten Helenenstr. 12	A1	2 E
12	19.08.20	9^{00}	Herten Helenenstr. 12	Herten Elisabethstr.	A1	2 E
	Manuell:	39,08€	Vorherige Fahrscheine			
		10,50€	4er Ticket	11, 12		
		$\sum$ 49,58€				
	App:	$\sum$ 78,18€				
13	20.08.20	12^{00a}	Herten Elisabethstr.	Bochum Hauptbahnhof	B	2 E
14	20.08.20	13^{00}	Bochum Hauptbahnhof	Bochum Karl-Friedrich-Str. 123	A3	2 E
15	20.08.20	13^{15}	Bochum Karl-Friedrich-Str. 123	Bochum Hauptbahnhof	A3	2 E
16	20.08.20	17^{00}	Bochum Hauptbahnhof	Herten Elisabethstr.	B	2 E
	Manuell:	49,58€	Vorherige Fahrscheine			
		18,70€	24-StundenTicket 2 Erwachsene, Preisstufe B	13, 14, 15, 16		

		$\sum$ 68, 28€				
	App:	$\sum$ 98, 08€				
17	22.08.20	17^{00}	Herten Elisabethstr.	Bochum Gilsingstr. 15	B	4 E
18	22.08.20	22^{00}	Bochum Gilsingstr. 15	Herten Elisabethstr.	B	4 E
	Manuell:	68, 28€	Vorherige Fahrscheine			
		26, 70€	24-StundenTicket 4 Erwachsene, Preisstufe B	17, 18		
		$\sum$ 94, 98€				
	App:	$\sum$138, 48€				
19	23.08.20	16^{00}	Herten Elisabethstr.	Essen Luttropp	B	2 E
20	23.08.20	20^{00}	Essen Luttrop	Herten Elisabethstr.	B	2 E
	Manuell:	68, 28€	Vorherige Fahrscheine			
		26, 70€	24-StundenTicket 2 Erwachsene, Preisstufe B	17, 18		
		35, 50€	48-StundenTicket 2 Erwachsene, Preisstufe B	17, 18, 19, 20		
		$\sum$122, 48€				
	App:	$\sum$156, 08€				
21	27.08.20	14^{00}	Herten Elisabethstr.	Recklinghausen Herzogswall 38	A1	2 E
22	27.08.20	18^{00}	Recklinghausen Herzogswall 38	Herten Elisabethstr.	A1	2 E
	Manuell:	122, 48€	Vorherige Fahrscheine			
		8, 40€	2x 4-StundenTicket	21, 22		

		$\sum 130,88€$				
	App:	$\sum$ 117, 58€				
23	01.09.20	11^{00}	Herten Elisabethstr.	Herten Josefstr.	A1	2 E
24	01.09.20	11^{30}	Herten Josefstr.	Herten Gelsenkirchenerstr.	A1	2 E
25	01.09.20	12^{30}	Herten Gelsenkirchenerstr.	Herten Elisabethstr.	A1	2 E
	Manuell:	130, 88€	Vorherige Fahrscheine			
		8, 40€	2x 4-StundenTicket	21, 22		
		$\sum 139,28€$				
	App:	$\sum$ 172, 88€				

Tabelle C.1: Vergleich der manuellen Optimierung mit der Optimierung der App für mehrere Personen.
[a]: Ankunftszeit.

	Datum	Uhrzeit	Start	Ziel	Preisstufe
1	12.08.20	18^{30}	Herten Elisabethstr.	Herten Josefstr.	A1
2	12.08.20	19^{00}	Herten Josefstr.	Herten Elisabethstr.	A1
	Manuell:	3, 19€	Happy Hour Ticket	1,2	
	App:	3, 19€			
3	13.08.20	12^{30}	Herten Elisabethstr.	Herten Uhlandstr.	A1
4	13.08.20	15^{00}	Herten Elisabethstr.	Herten Uhlandstr.	A1
5	13.08.20	19^{00}	Herten	Herten	A1

	Datum	Uhrzeit	Start	Ziel	Preisstufe
			Uhlandstr.	Elisabethstr.	
	Manuell:	7,20€	24-StundenTicket	1,2,3,4	
			1 Erwachsener, Preisstufe A		
		2,80€	Einzelticket A2	5	
		$\sum$ 10,00€			
	App:	$\sum$ 10,00€			
6	15.08.20	9^{33}	Herten	Herten	A1
			Schlägel- und Eisenstraße	Elisabethstr.	
7	15.08.20	10^{00}	Herten	Herne	B
			Elisabethstr.	Archälogiemuseum /	
				Kreuzkirche	
8	15.08.20	16^{00}	Herne	Herten	B
			Archäologiemuseum /	Gertrudenstr.	
			Kreuzkirche		
9	15.08.20	17^{00}	Herten	Herten	A1
			Gertrudenstr.	Schlägel- und Eisenstraße	
10	15.08.20	23^{00}	Herten	Herten	A1
			Schlägel- und Eisenstraße	Elisabethstr.	
	Manuell:	10,00€	Vorherige Fahrscheine		
		7,20€	24-StundenTicket	6,9,10	
			1 Erwachsener, Preisstufe A		
		12,00€	2x Einzelticket B	7,8	
		$\sum$ 29,20€			
	App:	$\sum$ 29,20€			
11	19.08.20	8^{30a}	Herten	Herten	A1
			Elisabethstr.	Helenenstr. 12	
12	19.08.20	9^{00}	Herten	Herten	A1

	Datum	Uhrzeit	Start	Ziel	Preisstufe
			Helenenstr. 12	Elisabethstr.	
	Manuell:	29,20€	Vorherige Fahrscheine		
		5,60€	2x Einzelticket A2	11, 12	
		∑ 34,80€			
	App:	∑ 34,80€			
13	20.08.20	12^{00}[a]	Herten	Bochum	B
			Elisabethstr.	Hauptbahnhof	
14	20.08.20	13^{00}	Bochum	Bochum	A3
			Hauptbahnhof	Karl-Friedrich-Str. 123	
15	20.08.20	13^{15}	Bochum	Bochum	A3
			Karl-Friedrich-Str. 123	Hauptbahnhof	
16	20.08.20	17^{00}	Bochum	Herten	B
			Hauptbahnhof	Elisabethstr.	
	Manuell:	49,58€	Vorherige Fahrscheine		
		14,70€	24-StundenTicket	13, 14, 15, 16	
			1 Erwachsener, Preisstufe B		
		∑ 49,50€			
	App:	∑ 50,40€			
17	22.08.20	17^{00}	Herten	Bochum	B
			Elisabethstr.	Gilsingstr. 15	
18	22.08.20	22^{00}	Bochum	Herten	B
			Gilsingstr. 15	Elisabethstr.	
	Manuell:	7,20€	24-StundenTicket	1,2,3,4	
			1 Erwachsener, Preisstufe A		
		2,80€	Einzelticket A2	5	
		7,20€	24-StundenTicket	6,9,10	
			1 Erwachsener, Preisstufe A		

	Datum	Uhrzeit	Start	Ziel	Preisstufe
		5, 60€	2x Einzelticket A2	11, 12	
		14, 70€	24-StundenTicket	13, 14, 15, 16	
			1 Erwachsener, Preisstufe B		
		22, 10€	4erTicket B	7, 8, 17, 18	
		∑ 59, 60€			
	App:	∑ 62, 40€			
19	23.08.20	16^{00}	Herten	Essen	B
			Elisabethstr.	Luttropp	
20	23.08.20	20^{00}	Essen	Herten	B
			Luttrop	Elisabethstr.	
	Manuell:	59, 60€	Vorherige Fahrscheine		
		12, 00€	4erTicket B	19,20	
		∑ 71, 60€			
	App:	∑ 71, 10€			
21	27.08.20	14^{00}	Herten	Recklinghausen	A1
			Elisabethstr.	Herzogswall 38	
22	27.08.20	18^{00}	Recklinghausen	Herten	A1
			Herzogswall 38	Elisabethstr.	
	Manuell:	71, 60€	Vorherige Fahrscheine		
		4, 20€	4-StundenTicket	21, 22	
		∑ 75, 80€			
	App:	∑ 75, 30€			
23	01.09.20	11^{00}	Herten	Herten	A1
			Elisabethstr.	Josefstr.	
24	01.09.20	11^{30}	Herten	Herten	A1
			Josefstr.	Gelsenkirchenerstr.	

	Datum	Uhrzeit	Start	Ziel	Preisstufe
25	01.09.20	12^{30}	Herten	A1	
			Gelsenkirchenerstr.	Elisabethstr.	
	Manuell:	75, 80€	Vorherige Fahrscheine		
		4, 20€	2x 4-StundenTicket	21, 22	
		$\sum$ 80, 00€			
	App:	$\sum$ 79, 50€			

Tabelle C.2: Vergleich der manuellen Optimierung mit der Optimierung der App für eine Person. [a]: Ankunftszeit.

Literaturverzeichnis

[1] Wibke Hinz. *VRR im Wandel: Vom Unternehmens- zum Mobilitätsverbund*. Hrsg. von VRR. 20. März 2020. URL: https://www.vrr.de/de/magazin/vrr-im-wandel-vom-unternehmens-zum-mobilitaetsverbund/ (besucht am 03. 08. 2020).

[2] Thomas Michael. *Diercke Weltatlas*. 1. Aufl. Westermann Schulbuch, 2008. ISBN: 978-3-14-100700-8.

[3] Verkehsverbund Rhein Ruhr AöR, Hrsg. *Der VRR-Verbundraum*. 2019. URL: https://www.vrr.de/fileadmin/user_upload/pdf/service/downloads/tarifhandbuch/verbundraum_2019.pdf (besucht am 14. 09. 2020).

[4] Niederrhein Tourismus GmbH, Hrsg. *Der Niederrhein*. 2020. URL: https://niederrhein-tourismus.de/ (besucht am 24. 09. 2020).

[5] Regionalverband Ruhr, Hrsg. *Die Städte und Kreise der Metropole Ruhr*. URL: https://www.rvr.ruhr/?id=45 (besucht am 24. 09. 2020).

[6] Regionalverband Ruhr, Hrsg. *Kreis Recklinghausen*. URL: https://www.rvr.ruhr/politik-regionalverband/staedte-kreise/kreis-recklinghausen/ (besucht am 24. 09. 2020).

[7] Regionalverband Ruhr, Hrsg. *Ennepe-Ruhr-Kreis*. URL: https://www.rvr.ruhr/politik-regionalverband/staedte-kreise/ennepe-ruhr-kreis/ (besucht am 24. 09. 2020).

[8] Regionalverband Ruhr, Hrsg. *Kreis Wesel*. URL: https://www.rvr.ruhr/politik-regionalverband/staedte-kreise/kreis-wesel/ (besucht am 24. 09. 2020).

[9] Bergische Struktur- und Wirtschaftsförderungsgesellschaft mbH, Hrsg. *Die Bergischen Drei*. URL: https://www.die-bergischen-drei.de/startseite.html (besucht am 24. 09. 2020).

[10] Rhein-Kreis Neuss, Hrsg. *Städte und Gemeinden im Rhein-Kreis Neuss*. URL: https://www.rhein-kreis-neuss.de/de/verwaltung-politik/kreisportrait/kommunen/index.html (besucht am 23. 09. 2020).

[11] Verkehsverbund Rhein Ruhr AöR, Hrsg. *VRR-Nahverkehrsplan 2017 Bericht*. 2017. URL: https://www.vrr.de/fileadmin/user_upload/pdf/der_vrr/allgemein/vrr-nahverkehrsplan_2017.pdf (besucht am 14. 09. 2020).

[12] Verkehsverbund Rhein Ruhr AöR, Hrsg. *Tarifgebiete, Regionen & Preisstufen. Der Verbundraum*. URL: https://www.vrr.de/de/tickets-tarife/tarifgebiete-regionen-preisstufen/ (besucht am 27. 09. 2020).

[13] Verkehsverbund Rhein Ruhr AöR, Hrsg. *Der Tarif im Überblick. Für die Fahrt mit Bus und Bahn*. 2020. URL: https://www.vrr.de/fileadmin/user_upload/pdf/service/downloads/tarifinformationen/Der_Tarif_im_UEberblick.pdf (besucht am 14. 09. 2020).

[14] Verkehsverbund Rhein Ruhr AöR, Hrsg. *Wabenplan des VRRs*. Für eine Betrachtung der Waben den Link nicht mit Firefox öffnen. 1. Jan. 2020. URL: https://www.vrr.de/fileadmin/user_upload/pdf/service/downloads/tarifhandbuch/Wabenplan.pdf (besucht am 20. 08. 2020).

[15] Verkehsverbund Rhein Ruhr AöR, Hrsg. *Tickets und Preise. Für Vielfahrer und Ab-und-zu-Fahrer*. 2020. URL: https://www.vrr.de/fileadmin/user_upload/pdf/service/downloads/broschueren_vrr/Tickets___Preise_fuer_Vielfahrer_und_Ab-und-zu-Fahrer.pdf (besucht am 14. 09. 2020).

[16] Verkehsverbund Rhein Ruhr AöR, Hrsg. *Ticketmerkmale*. 1. Aug. 2020. URL: https://www.vrr.de/fileadmin/user_upload/pdf/service/downloads/tarifinformationen/Ticketmerkmale.pdf (besucht am 14. 09. 2020).

[17] Verkehsverbund Rhein Ruhr AöR, Hrsg. *VRR App - Fahrplanauskunft*. 2011. URL: https://play.google.com/store/apps/details?id=com.mdv.VRRCompanion&pcampaignid=MKT-Other-global-all-co-prtnr-py-PartBadge-Mar2515-1 (besucht am 14. 09. 2020).

[18] Andreas Schildbach, Hrsg. *Öffi Fahrplanauskunft*. URL: https://play.google.com/store/apps/details?id=de.schildbach.oeffi (besucht am 14. 09. 2020).

[19] GeoMobile GmbH, Hrsg. *Mutti*. 1. Dez. 2016. URL: https://play.google.com/store/apps/details?id=de.ivanto.bogestra (besucht am 27. 09. 2020).

[20] HaCon Ingenieurges. mbH, Hrsg. *MyHannover*. 30. März 2016. URL: https://play.google.com/store/apps/details?id=de.hafas.android.hannover (besucht am 14. 09. 2020).

[21] Berliner Verkehrsbetriebe, Hrsg. *Jelbi*. 10. Juni 2019. URL: https://play.google.com/store/apps/details?id=com.trafi.whitelabel.bvg (besucht am 14. 09. 2020).

[22] Deutsche Bahn, Hrsg. *DB Navigator*. 30. März 2010. URL: https://play.google.com/store/apps/details?id=de.hafas.android.db (besucht am 14. 09. 2020).

[23] Moovit, Hrsg. *Moovit: Buszeiten, Bahn Fahrplan & ÖPNV Info*. 16. März 2014. URL: https://play.google.com/store/apps/details?id=com.tranzmate (besucht am 14. 09. 2020).

[24] Citymapper, Hrsg. *Unsere Städte*. URL: https://citymapper.com/cities (besucht am 15. 09. 2020).

[25] Citymapper Limited, Hrsg. *Citymapper – Nahverkehr Routenplaner*. 4. Nov. 2013. URL: https://play.google.com/store/apps/details?id=com.citymapper.app.release (besucht am 15. 09. 2020).

[26] Verkehsverbund Rhein Ruhr AöR, Hrsg. *Ticketberater*. URL: http://tvr.webservice-ivv.de/tbv/vrr/cgi (besucht am 17. 08. 2020).

[27] Fairtiq AG, Hrsg. *Gültigkeitsbereich*. URL: https://fairtiq.com/de-ch/fairtiq-app/gultigkeitsbereich-fairtiq (besucht am 20. 08. 2020).

[28] Fairtiq AG, Hrsg. *Fairtiq*. 27. Apr. 2016. URL: https://play.google.com/store/apps/details?id=com.fairtiq.android (besucht am 20. 08. 2020).

[29] Andreas Schildbach. *Öffi your public transport companion. Requirements*. URL: https://oeffi.schildbach.de/ (besucht am 10. 09. 2020).

[30] Olivier Chapuis und Pierre Dragicevic. „Effects of motor scale, visual scale, and quantization on small target acquisition difficulty". In: *ACM Transactions on Computer-Human Interaction* 18.3 (Juli 2011). DOI: 10.1145/1993060.1993063.

[31] VMT GmbH, Hrsg. *Check-in/Check-out. FAQ*. Abrechnung und Zahlungsmittel. URL: https://www.vmt-thueringen.de/tickets/ticketkauf/check-in-check-out/#a2827 (besucht am 10. 09. 2020).

[32] Ilja N. Bronsthein, Konstantin A. Semendyayev, Gerhard Musiol, Heiner Muehlig. *Handbook of Mathematics*. 5. Aufl. Springer, 2007. ISBN: 978-3-540-72121-5.

[33] Verkehsverbund Rhein Ruhr AöR, Hrsg. *Ticketübersicht. Einzelfahrten*. URL: https://www.vrr.de/de/tickets-tarife/ticketuebersicht/ (besucht am 17. 08. 2020).

[34] Statista, Hrsg. *Marktanteile der führenden mobilen Betriebssysteme an der Internetnutzung mit Mobiltelefonen in Deutschland von Januar 2009 bis Mai 2020*. 18. Juni 2020. URL: https://de.statista.com/statistik/daten/studie/184332/umfrage/marktanteil-der-mobilen-betriebssysteme-in-deutschland-seit-2009/ (besucht am 20. 08. 2020).

[35] Statista, Hrsg. *Anteile der verschiedenen Android-Versionen an allen Geräten mit Android OS weltweit im Zeitraum 01. bis 07. Mai 2019*. 14. Nov. 2019. URL: https://de.statista.com/statistik/daten/studie/180113/umfrage/anteil-der-verschiedenen-android-versionen-auf-geraeten-mit-android-os/#professional (besucht am 20. 08. 2020).

[36] Google Developers, Hrsg. *<uses-sdk>*. URL: https://developer.android.com/guide/topics/manifest/uses-sdk-element (besucht am 15. 09. 2020).

[37] Verkehsverbund Rhein Ruhr AöR, Hrsg. *Testsystem Fahrgastinformation*. 18. Aug. 2018. URL: https://openvrr.de/dataset/testsystem-fahrgastinformation (besucht am 22. 08. 2020).

[38] Joachim Goll. *Methoden und Architekturen der Softwaretechnik*. 1. Aufl. Wiesbaden: Vieweg+Teubner Verlag, 2011. ISBN: 978-3-834-81578-1.

[39] Andreas Spillner. *Basiswissen Softwaretest : Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB-Standard*. 5. Aufl. dpunkt.verlag, 2012. ISBN: 978-3-864-90024-2.

[40] Maud Schlich. *Softwaretesten nach ISTQB für Dummies*. Wiley-VCH GmbH, 10. Apr. 2019. ISBN: 978-3-527-71518-3.

[41] Google Developers, Hrsg. *AsyncTask*. 10. Juni 2020. URL: `https://developer.android.com/reference/android/os/AsyncTask` (besucht am 22. 09. 2020).

[42] Google Developers, Hrsg. *Directions API Usage and Billing*. URL: `https://developers.google.com/maps/documentation/directions/usage-and-billing` (besucht am 17. 08. 2020).

[43] Google Developers, Hrsg. *SharedPreferences*. 27. Dez. 2019. URL: `https://developer.android.com/reference/android/content/SharedPreferences` (besucht am 22. 09. 2020).

[44] Google Developers, Hrsg. *Save key-value data*. 27. Dez. 2019. URL: `https://developer.android.com/training/data-storage/shared-preferences#java` (besucht am 22. 09. 2020).

[45] Google Inc., Hrsg. *Gson*. 19. März 2015. URL: `https://github.com/google/gson` (besucht am 22. 09. 2020).

[46] *Einführung in JSON*. URL: `https://www.json.org/json-de.html` (besucht am 22. 09. 2020).

[47] Verkehsverbund Rhein Ruhr AöR, Hrsg. *Haltestellen*. 17. Apr. 2017. URL: `https://openvrr.de/dataset/haltestellen` (besucht am 31. 08. 2020).

[48] Großraum-Verkehr Hannover GmbH, Hrsg. *Neue Tarife ab 1. Januar 2020*. 1. Nov. 2019. URL: `https://einfach.gvh.de/downloads/GVH_Flyer-Tarifstrukturreform-2020.pdf` (besucht am 16. 09. 2020).

Abbildungsverzeichnis

2.1 Der Verbundraum und die Regionen 4
2.2 Ausschnitt des Wabenplans 5

3.1 Beispiel für Listenansicht und grafische Ansicht. 15

4.1 Anwendungsfalldiagramm. 28
4.2 Aktivitätsdiagramm für *mehrere Fahrten planen*. 33
4.3 Geplanter Ablauf der Aktivität *einzelne Fahrt planen*. 34
4.4 Aktivitätsdiagramm für die Aktivität *Fahrt bearbeiten* 35
4.5 Klassen aus Öffi 37
4.6 Klassen für die Darstellung eines ÖPNV-Abschnitts in Öffi 38
4.7 Klassen für die Fahrscheine. 39
4.8 Entwurf des Layouts 41
4.9 Entwicklungsmodell. 42

5.1 Erweiterung Klassendiagramm: Planung von Fahrten 48
5.2 Beispiel für die rekursive Optimierung mit Einzelfahrscheinen. 49
5.3 Beispiel für die dynamische Optimierung mit Einzelfahrscheinen. . . . 50
5.4 Erweiterung Klassendiagramm: Einzelfahrscheine 52
5.5 Klassendiagramm Erweiterung: Generalisierung 53
5.6 Beispiel für die Verwendung alter Einzelfahrscheine. 55
5.7 Anpassung Klassendiagramm: Verwendung alter Fahrscheine 58
5.8 Gültigkeitsbereich 15 für die Preisstufe B 60
5.9 Anpassung Klassendiagramm: Zeitfahrscheine 72
5.10 Visualisierung der Vorbereitung für die Optimierung mit mehreren Personen. 73
5.11 Sequenzdiagramm der Optimierung - Teil 1 76
5.12 Sequenzdiagramm der Optimierung - Teil 2 77
5.13 Erweiterung Klassendiagramm: Aktualisieren von Verbindungen . . . 78

B.1 Fahrscheine im globalen Zusammenhang. 100
B.2 Fahrten im globalen Zusammenhang. 101
B.3 Verbliebene Klassen im globalen Zusammenhang. 102
B.4 Realisierte Ansichten. 103

Tabellenverzeichnis

2.1 Einzelfahrscheine des VRRs 6
2.2 Zeitfahrscheine des VRRs 7

3.1 Anwendungsvergleich: Routenoptimierung und Verkehrsmittel 12
3.2 Anwendungsvergleich: Planung einer Fahrt 13
3.3 Anwendungsvergleich: Mögliche Verbindungen 14
3.4 Anwendungsvergleich: Detailansicht einer Fahrt 16
3.5 Vergleich von 7-TageTicket, Ticket1000 und Ticket2000 mit den Einzelfahrscheinen. 23

4.1 Erläuterung der Anwendungsfälle. 30

5.1 Systemtestbeispiel für die Verwendung alter Fahrscheine 56

A.1 Vergleich aller betrachteten Anwendungen: Routenoptimierung, Verkehrsmittel 92
A.2 Vergleich aller betrachteten Anwendungen: Planung einer Fahrt . . . 92
A.3 Vergleich aller betrachteten Anwendungen: Darstellung möglicher Verbindungen 92
A.4 Vergleich aller betrachteten Anwendungen: Darstellung einer möglichen Verbindung 93
A.5 Vergleich aller betrachteten Anwendungen: Detailansicht einer Fahrt . 93
A.6 Ersparnis und minimale Anzahl an Fahrten für die jeweiligen Preisstufen. 94

C.1 Vergleich der manuellen Optimierung mit der Optimierung der App für mehrere Personen. 112
C.2 Vergleich der manuellen Optimierung mit der Optimierung der App für eine Person. 116

Codeverzeichnis

1 Optimierung für die Preisstufe D - Teil 1 59
2 Optimierung für die Preisstufe D - Teil 2 60
3 Optimierung für die Preisstufe B - Teil 1 62
4 Optimierung für die Preisstufe B - Teil 2 63
5 Optimierung für die Preisstufe A 64
6 Optimierung für die jeweilige Abstufung der Preisstufe A 65
7 Optimierung der Fahrten im zwei-Waben-Tarif 66
8 Vollständige Suche nach dem besten Intervall 68
9 Zusammenfassen von Fahrscheinen 70
10 Optimierung für die Preisstufe C - Teil 1 95
11 Optimierung für die Preisstufe C - Teil 2 96
12 Optimierung der Fahrten innerhalb eines Tarifgebiets 97
13 Intervall mit dem höchsten Ticketpreis suchen - Teil 1 98
14 Intervall mit dem höchsten Ticketpreis suchen - Teil 2 99
15 Verwenden von Fahrscheinen für andere Fahrten 99

Selbstständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.
Seitens des Verfassers bestehen keine Einwände, die vorliegende Bachelorarbeit für die öffentliche Benutzung im Universitätsarchiv zur Verfügung zu stellen.

Jena, 30. September 2020, [redacted]