Implementierung von Echtzeitdaten in die Flughafensimulation Optimode.net

Implementation Of Real-Time Data In The Airport Simulation Optimode.net

Armin Krain

363381

Abschlussarbeit zur Erlangung des Grades

Bachelor of Engineering

Im internationalen Studiengang

Luftfahrtsystemtechnik und -management

22. September 2016

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit „Implementierung von Echtzeitdaten in die Flughafensimulation Optimode.net“ selbstständig verfasst sowie alle benutzten Quellen und Hilfsmittel vollständig angegeben habe und dass die Arbeit nicht bereits als Prüfungsarbeit vorgelegen hat.

Ort, Datum

Armin Krain

Abstract

In 2011 the European Commission published the aim to transport 90% of Europe's travellers in 2050 from door to door within four hours. Therefore, new concepts of managing intermodal journeys are required. Within the framework project `Optimode.net`, the German Aerospace Center (Deutsches Zentrum für Luft- und Raumfahrt, abbr.:DLR) works on a more efficient cross-linking of different modes of transportation, considering aviation as a special aspect. This thesis analyzes the indication of journeys on the passenger's mobile device in `Optimode.net` and other, already existing solutions. Additionally, data interfaces are considered, which provide free access to transportation data. One of these interfaces, the `Fraport API`, is implemented into `Optimode.net`. This is achieved by programming a tool in C#, which converts the JSON-formatted flight data of the RESTful interface and writes them into a MySQL-database.

As a result, an improved data basis consisting of real-time data is created and can be used by the simulation and control station environment of `Optimode.net`. This thesis also provides approaches to create a feedback API for passengers and presents possibilities for the extension of traffic simulations by means of Web-APIs.

Inhaltsverzeichnis

Abkürzungsverzeichnis

Öffentlicher Personennahverkehr (ÖPNV) Lokaler und regionaler Massentransport von Personen.

Application Programmable Interface (API) Schnittstelle zur Datenübertragung.

Datenbankmanagementsystem (DBMS) System, welches den Zugriff auf eine Datenbank verwaltet.

Deutsche Zentrum für Luft- und Raumfahrt (DLR) Forschungszentrum für Luft- und Raumfahrt, Energietechnik, Verkehr und Sicherheit.

Extensible Markup Language (XML) Textformat zur Beschreibung von Daten.

Generic International Airport (GIA) Fiktiver Flughafen für die Simulation in Optimode und Optimode.net.

Hypertext Markup Language (HTML) Auszeichnungssprache zur Erstellung von Websites.

Hypertext Transfer Protocol (HTTP) Datenübertragungsprotokoll.

International Civil Aviation Organization (ICAO) Organisation der Vereinten Nationen, zuständig für den zivilen Luftverkehr.

Javascript Object Notation (JSON) Format zur Datenübertragung zwischen verschiedenen Programmiersprachen.

Representational State Transfer (REST) Softwarearchitekturstil.

Standard Query Language (SQL) Standardisierte Sprache zum Zugriff auf Datenbanken.

Uniform Resource Locator (URL) Zeichenkette zur Identifizierung einer Ressource.

Uniform Resource Identifier (URI) Zeichenkette zur Identifizierung einer Ressource.

World Wide Web Consortium (W3C) Gremium zur Erstellung von Standards im World Wide Web.

Glossar

Action Bar Oberer Teil einer Android App, stellt Titel der aktuellen Seite und Verknüpfungen zu Menü und Einstellungen dar [1].

Android Mobiles Betriebssystem von Google.

App Application, Programm auf Mobilgeräten.

array Nummerierte Liste von Werten.

BahnCard Kundenkarte der Deutschen Bahn. Berechtigt zu Vergünstigungen je nach Typ.

Bordkarte Ticket in elektronischer oder Papierform, das benötigt wird, um das Flugzeug betreten zu können.

Codeshare Abkommen zwischen Fluggesellschaften, um Passagiere mit Tickets anderer Airlines befördern zu können. Der Flug besitzt oftmals mehrere Flugnummern.

Diversion Umleitung eines Fluges zu einem anderen Zielflughafen.

Garbage Collector Tool zur Speicheranalyse. Wird zur Speicherreduzierung von Programmen verwendet.

JavaScript Skriptsprache, basierend auf ECMAScript [2].

Link „[grafisch hervorgehobene] Verknüpfung mit einer anderen Datei oder einer anderen Stelle in derselben Datei [die vom Benutzer z.B. per Mausklick aktiviert werden kann]“ [3].

MySQL Datenbankmanegemtsystem. Wird für `Optimode.Net` verwendet.

NuGet Tool für externe Pakete in Visual Studio.

Play Store Internetangebot von Google. Dient zum Herunterladen von Apps für Android, Filmen oder Musik.

Pushbenachrichtigung Benachrichtigungen, die ohne Öffnen der jeweiligen App auf einem mobilen Gerät erscheinen.

scrollen „eine Darstellung, die auf dem Bildschirm nicht im Ganzen erfasst werden kann, in Ausschnitten nach und nach auf dem Bildschirm verschieben [4]“.

string Englischer Begriff für Zeichenkette. Wird häufig in Programmiersprachen benutzt.

Thread Ausführungseinheit in einem Programm.

1 Einleitung

> „Wenn Sie vom Hauptbahnhof in München ... mit zehn Minuten, ohne, dass Sie am Flughafen noch einchecken müssen, dann starten Sie im Grunde genommen am Flughafen ... am ... am Hauptbahnhof in München starten Sie Ihren Flug. [...] Das bedeutet natürlich, dass der Hauptbahnhof im Grunde genommen näher an Bayern ... an die bayerischen Städte heranwächst, weil das ja klar ist, weil auf dem Hauptbahnhof viele Linien aus Bayern zusammenlaufen.“ [5]

Dies ist ein Ausschnitt einer Rede des damaligen bayrischen Ministerpräsidenten Edmund Stoiber aus dem Jahr 2002. Das Thema der Rede war eine geplante Transrapidverbindung zwischen dem Münchener Hauptbahnhof und dem Flughafen Franz-Josef-Strauß. Der Transrapid ist eine Magnetschwebebahn, entwickelt von `ThyssenKrupp` und `Siemens`, mit einer Betriebsgeschwindigkeit von bis zu 430 km/h. Im Gegensatz zu anderen Schienenfahrzeugen berührt sie den Fahrweg nicht, sondern schwebt auf einem Magnetfeld zwischen Schiene und Fahrzeug [6]. Die Planungen für das Projekt wurden wegen zu hoher Kosten im März 2008 eingestellt [7].

Die schnelle Reise zum Flughafen ist weiterhin ein relevantes Thema. Circa 42% der Originäreinsteiger in München reisen per S-Bahn, Bus oder Transferdienst am Flughafen an. Originäreinsteiger sind Passagiere, die mit bodengebundenen Verkehrsträgern am Flughafen ankommen, um ihre Flugreise dort zu starten [8]. Die schnellste Verbindung vom Flughafen zum Hauptbahnhof München dauert mit der Bahn planmäßig 41 Minuten [9]. Mit dem Transrapid läge die Zeitersparnis laut Stoiber bei 31 Minuten Fahrtzeit.

Die Europäische Kommission hat für den Luftverkehr 2050 Ziele festgelegt. Dazu gehört, dass EU-Bürger Zugang zu Informationen bekommen, um ihre Reisen aufgrund von Geschwindigkeit, wirtschaftlichen Aspekten und persönlichen Präferenzen selbstständig planen zu können. Zusätzlich besteht die Bestrebung, 90% der Reisenden innerhalb Europas eine Tür-zu-Tür-Verbindung, mit einer maximalen Dauer von vier Stunden, zu ermöglichen [10].

Eine Tür-zu-Tür-Betrachtung einer Flugreise beschreibt nicht nur den Reiseteil im Flugzeug, sondern auch die Reise vom Startpunkt zum Flughafen und den Weg vom Zielflughafen zum eigentlichen Ziel der Reise. Das Deutsche Zentrum für Luft- und Raumfahrt (DLR) forscht mit dem Projekt `Optimode.net` an der Analyse der vollständigen Reisekette.

`Optimode.net` bietet die Grundlage für diese Bachelorarbeit. Anhand dieses Projekts soll eine Untersuchung stattfinden, wie ein Passagier bei der Planung seiner Reise unterstützt werden kann und welche Möglichkeiten es für ihn gibt, während seiner Reise einen Überblick über die nächsten Reiseabschnitte bewahren zu können. Zusätzlich wird das Thema der Datenschnittstellen betrachtet.

Die folgenden Kapitel behandeln die Integration von Echtzeitdaten der Flüge am Frankfurter Flughafen in die Leitstandsumgebung Optimode.net. Dazu wird zunächst das Projekt `Optimode.net` vorgestellt. Anschließend wird die Möglichkeit der Reiseauskunft über mobile Endgeräte thematisiert und die Datenerweiterung der Leitstandsumgebung von `Optimode.net` durch Datenschnittstellen analysiert. Auf Basis der Schnittstelle des Flughafens Frankfurt wird ein Programm zur Integration der Frankfurter Daten geschrieben und dieses dokumentiert. Abschließend wird ein Fazit gezogen und es findet ein Ausblick auf die Realisierung einer möglichen eigenen Schnittstelle zur Nutzung auf mobilen Endgeräten jedes Passagiers statt.

Ziel der Arbeit ist die Integration realer Daten in das System von `Optimode.net`, um die Leitstandsumgebung realitätsnaher gestalten zu können. Es wird erhofft, das Gesamtsystem durch die Integration von Echtzeitdaten aus Frankfurt signifikant zu verbessern und neue Erkenntnisse in der Interaktion von Realdaten auf das fiktive System zu gewinnen.

2 Aktueller Stand von Forschung und Technik

Zu Beginn wird das Projekt `Optimode.net` näher vorgestellt. Dabei wird der Fokus auf die Anzeige der Reisekette für den Passagier gelegt. Anschließend werden vorhandene, mobile Anwendungen zur Ansicht von Reisen verschiedener Anbieter analysiert. Am Ende des Kapitels werden verschiedene Schnittstellen für Reisedaten untersucht.

2.1 Optimode.net

`Optimode.net` ist ein Forschungsprojekt des DLR zur Entwicklung eines interaktiven Flughafenmanagementsystems. Ziel ist unter anderem die Koordination einer Tür-zu-Tür-Reise jedes Reisenden, bei der mindestens ein Abschnitt der Reisekette eine Flugreise darstellt. Dabei wird der Fokus auf Intermodalität und die Optimierung der Reisekette von Passagieren oder Gütern gelegt.

Intermodalität ist eine besondere Form der Multimodalität. Multimodalität beschreibt die Wahl verschiedener Verkehrsträger zur Durchführung einer Reise, die jedoch aus mehreren Wegen bestehen kann. Fährt ein Passagier jeden Tag von Punkt A zu Punkt B und nutzt dafür an einem Tag das Auto, am nächsten für den gleichen Weg aber die Bahn, wird dies als multimodal bezeichnet. Beginnt der Passagier an einem Tag seinen Weg mit dem Auto und steigt dann in die Bahn um, ist sein Reiseverhalten intermodal [11].

`Optimode.net` greift dabei auf die Ergebnisse des Vorgängerprojektes `Optimode` zu, welches 2015 endete. Durch `Optimode` wurde bereits das Konzept für einen Verkehrsleitstand erarbeitet und ein Prototyp mit einem fiktiven Flughafen, auch als Generic International Airport (GIA) bezeichnet, getestet. Diese verwendete Simulationsumgebung wird für `Optimode.net` ausgebaut und erweitert. Jegliche Reiseverbindungen sind fiktiv und in einer Datenbank gespeichert. Durch den Leitstand kann der Datensatz mit den Verbindungen verändert werden, die Personen am Leitstand können somit in die Simulation eingreifen.

Ein weiterer Aspekt von `Optimode.net` ist die Untersuchung der kompletten persönlichen Passagiertrajektorie. Eine Trajektorie beschreibt die Position von Objekten, abhängig von der Zeit [12, S.7]. Wird durch Verspätung auf einem Abschnitt der Anschluss zum nächsten Abschnitt nicht mehr erreicht, soll `Optimode.net` diesen Umstand berücksichtigen und selbstständig einen neuen Weg für den Passagier bestimmen. Das System meldet

diese Veränderung an den interaktiven Leitstand. Jeder Passagier besitzt die Möglichkeit seine persönliche Reisekette über eine Anwendung für mobile Endgeräte jederzeit einsehen zu können.

Die prototypische Ansicht, dargestellt in Abbildung 1, erfolgt über eine Webansicht und ist aufgeteilt in die einzelnen Reiseabschnitte mit kleinen Symbolen am linken Bildschirmrand. Am oberen Bildschirmrand befindet sich das Logo des Projekts. Darunter ist ein Abschnitt zu den allgemeinen Informationen. Neben der aktuellen Uhrzeit befindet sich dort die Flugnummer, das Abfluggate, die Abflugzeit und Informationen über die Erreichbarkeit der Anschlüsse. Angezeigt wird die geschätzte Ankunftszeit am Gate, der geschätzte Start des Boardings und die daraus resultierende persönliche Pufferzeit zum Gate. Wird ein Teil der Reise per Schienenverkehr zurückgelegt, gibt es einen Abschnitt mit Abfahrts- und Ankunftszeit des Schienenfahrzeugs. Der Flughafen wird getrennt von der eigentlichen Flugreise betrachtet.

Im Abschnitt des Flughafens sind Ankunftszeit am Flughafen, dem Check-in und der Sicherheitskontrolle zu finden. Bei Check-in und Sicherheitskontrolle sind neben den Ankunftszeiten auch noch Wartezeiten zu finden, die je nach Wartedauer farblich gekennzeichnet sind. Der Abschnitt der Flugreise besitzt als Überschrift die Nummer des Gates und enthält die Ankunftszeit am Gate, Flugnummer und Boardingzeit. Jeder Abschnitt kann aus- oder eingeklappt werden, um bei langen Reiseketten mit vielen Abschnitten die Übersichtlichkeit aufrechterhalten zu können. Ist ein Abschnitt abgeschlossen, befindet sich ein grüner Haken hinter der Überschrift des Abschnitts. Diese Ansicht ist in Abbildung 1 zu sehen.

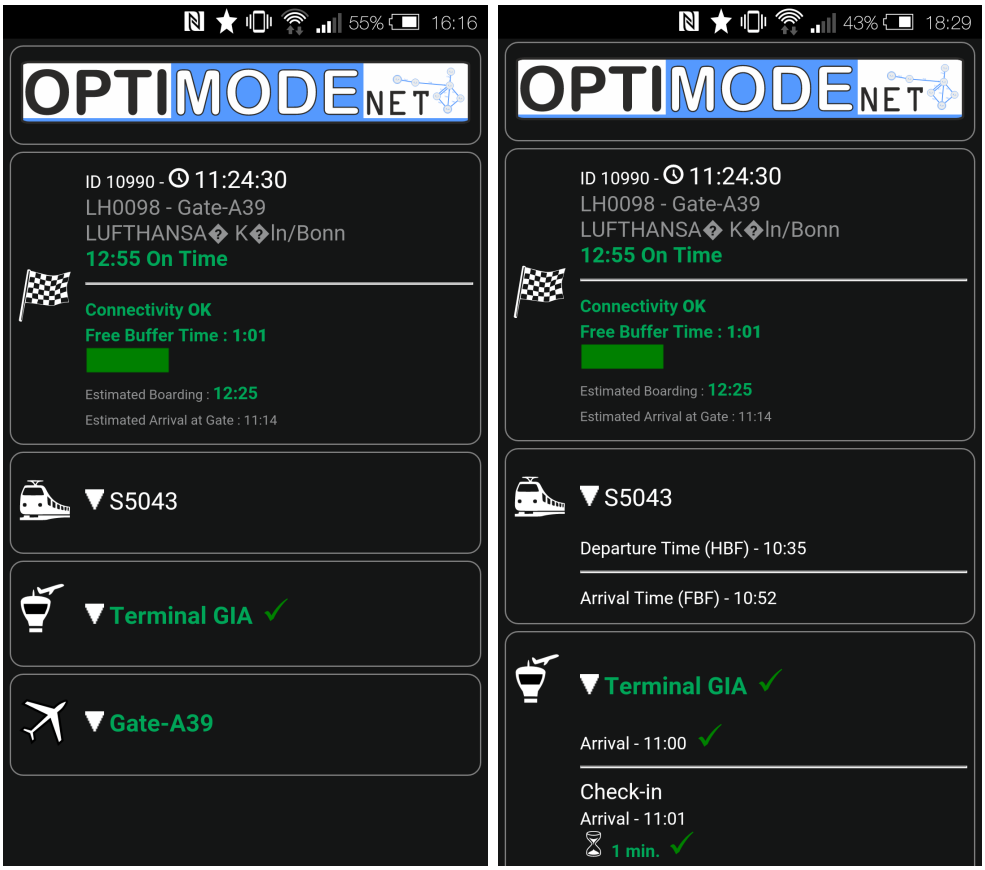


Abbildung 1: Ansicht der Webschnittstelle `Optimode.net`, links in der Zusammenfassung und rechts in ausgeklapptem Zustand [13].

2.2 Verfügbare Apps

In dieser Arbeit wird nur das mobile Betriebssystem `Android` betrachtet. Gemessen an den verkauften Smartphones besitzt `Android` in Deutschland von April bis Juni 2016 einen Marktanteil von 79,2% [14]. Dahinter folgen `iOS`, ein Betriebssystem welches von `Apple` entwickelt wurde und nur auf Geräten von `Apple` benutzt werden kann, und `Windows Phone` von `Microsoft`. Im Gegensatz zu `Apple` stellt `Google` nur die Software zur Verfügung. Hersteller von Smartphones können `Android` gegen Lizenzgebühren auf ihren Geräten installieren. Im `Play Store`, dem Zugangspunkt für Apps für `Android`, befinden sich über 700.000 Apps [15].

Im Fokus der intermodalen Reisebetrachtung dieser Arbeit steht der Luftverkehr. Für die Analyse von Apps zur Reiseplanung werden Schienenverkehr und Öffentlicher Personennahverkehr (ÖPNV) als Zubringer zum und vom Flughafen angenommen. Für die Anzeige verkehrsbezogener Daten sind bereits mehrere Apps verfügbar. Diese lassen sich, je nach Herausgeber, in drei Hauptklassen aufteilen:

- Apps von Flughäfen
- Apps von Transportdienstleistern
- Apps von Drittanbietern

In dieser Arbeit werden die Apps der Flughäfen Frankfurt und München, der Fluglinie `Lufthansa` und der `Deutschen Bahn` aus dem Bereich Schienenverkehr thematisiert. Aus den Apps der Drittanbieter wurden `Öffi`, `Google Maps` und `Quixxit` zur Analyse ausgewählt.

2.2.1 Flughäfen

FRA Airport

Die offizielle App des Flughafens Frankfurt ist zum Zeitpunkt der Betrachtung in Version 3.0.1 verfügbar [16]. Die Startseite ist in Abbildung 2(a) zu sehen. Sie ist optisch in drei Bereiche aufgeteilt. Am oberen Rand befindet sich die `Action Bar`. Sie wird häufig in `Android`-Apps verwendet und stellt einen Button zum Öffnen des Menüs und die Überschrift der aktuellen Seite dar. Im Hauptteil befinden sich ein Eingabefeld zur Stichwortsuche innerhalb der App, zusammengefasste Informationen über die nächsten Flüge, Links zu Kartendarstellungen und Zusammenfassungen zu Serviceleistungen am Flughafen. Serviceleistungen werden in folgende Abschnitte aufgeteilt:

- Flüge
- zeige Karte für ...
- Frankfurt Airport Shopping
- News
- Frankfurt Airport Rewards
- Parken am Airport
- Services am Airport buchen
- Faszination Airport
- Specials am Frankfurter Flughafen

Die einzelnen Abschnitte sind über Scrollen erreichbar und bei einer Berührung öffnet sich das jeweilige Thema als Hauptansicht. Ein Menü mit den Punkten „Live“ , „My FRA“ und „Guide“ befindet sich am unteren Teil des Bildschirms. „Live“ ist hierbei bereits grün eingefärbt, da es nach dem Start der App automatisch aufgerufen wird. Während „My FRA“ zu einer Anmeldemaske weiterleitet, die Zugriff auf einen personalisierten Bereich mit Merklisten und einem Prämienprogramm gewährt, werden nach einer Aktivierung von „Guide“ die meistgelesenen Artikel, Informationen über An- und Abreise, Sicherheitskontrollen, Gepäck und „nützliche Tipps“ angezeigt. Über das Menü in der `Action Bar` sind zusätzlich Möglichkeiten zur Navigation auf dem Flughafen, zum Scannen der Bordkarte, zum Übersetzen von Schildern am Flughafen, zur Anzeige von aktuellen Wartezeiten an Sicherheitskontrollen, zur Kontaktaufnahme mit den Herausgebern der App und zur Einsicht von Impressum und Datenschutzerklärung hinterlegt.

Die Unterseite „Flüge“ stellt Flüge von oder nach Frankfurt chronologisch dar. Die `Action Bar` dient nun direkt als Menü, über das sich die Ansichten „Abflüge“, „Ankünfte“, „Saisonflugplan“ oder „Weltkarte“ auswählen lassen. Abflüge und Ankünfte werden nach ihrer geplanten Abflugs- beziehungsweise Ankunftszeit in einer Liste sortiert. Jedes Listenelement enthält ein Logo der Fluggesellschaft, Abflug- beziehungsweise Ankunftsort, Flugnummer, Terminal und Gate, Status und geplante sowie erwartete Abflug- beziehungsweise Ankunftszeit eines bestimmten Fluges. Informationen zu Verspätungen oder Annullierungen werden dabei rot eingefärbt, bestätigte Statusmeldungen wie „Gate offen“, „gelandet“ oder „Gepäckausgabe“ werden in grüner Schrift angezeigt. Eine Berührung auf einen Flug in der Liste öffnet eine neue Seite mit erweiterten Informationen zu diesem Flug. In der `Action Bar` werden dann Start- und Zielflugort als Überschrift angezeigt. Über einer Tabelle mit den erweiterten Informationen befindet sich ein Bild, das charakteristisch für den Ankunftsort, bei Abflügen aus Frankfurt, beziehungsweise den Abflugort, bei Flügen mit Frankfurt als Zielflughafen, ist. Zu den erweiterten Fluginformationen gehören:

- eventueller Zwischenhalt
- Flugnummer und Codeshares
- Planmäßige und erwartete Ankunfts- bzw. Abflugzeit
- Name der Airline

- Flugzeugtyp
- Registrierung des Flugzeugs
- Terminal
- Ausgang bzw. Gate und Check-in Schalter

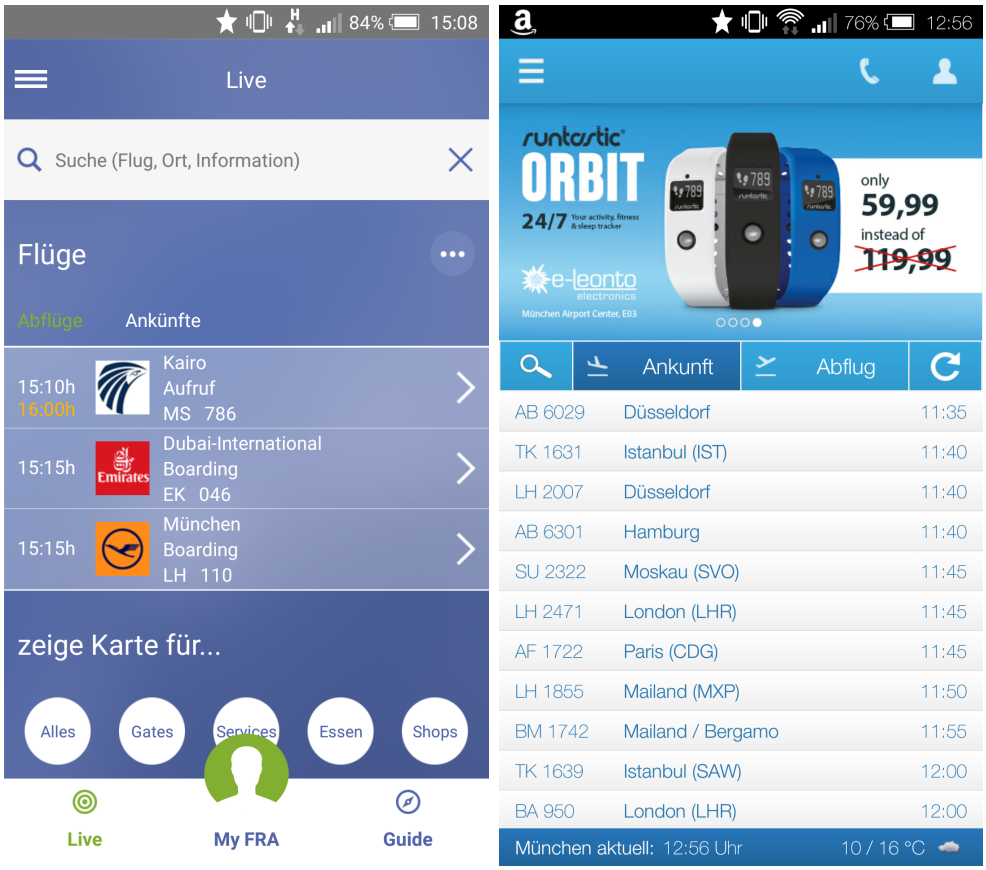
Unter dieser Tabelle befinden sich Links, um eine Navigation zum Gate oder Ausgang des entsprechenden Fluges zu starten, den Flug zur Merkliste hinzuzufügen, eine Karte zu öffnen, einen Parkplatz zu buchen, einen Scanner für Bordkarten zu starten oder die Fluginformationen per E-Mail oder SMS zu teilen. In der Übersicht der An- und Abflüge und in der Ansicht der erweiterten Informationen für jeden Flug befindet sich am unteren Bildschirmrand eine Leiste mit dem Datum und der Uhrzeit der letzten Aktualisierung der Flugdaten.

Wird in der Action Bar der Punkt „Saisonflugplan“ ausgewählt, öffnet sich eine Suchmaske, in der Flüge nach Zielort, mit Start in Frankfurt, oder Startflughafen, mit Landung in Frankfurt, Fluggesellschaft, Datum, Uhrzeit oder Flugnummer gefiltert werden können. Nach Eingabe der Daten wird eine Liste der gefilterten Ergebnisse angezeigt. Eine Berührung eines Fluges dieser Liste führt wieder zu erweiterten Informationen zu diesem Flug. Diese Seite unterscheidet sich nur geringfügig von den erweiterten Informationen, die unter den Menüpunkten „Abflüge“ und „Ankünfte“ zu finden sind. So sind kurzfristig verfügbare Angaben wie Gate und Registrierung des Flugzeugs nicht sichtbar, dafür jedoch die geplante Flugzeit, der Zeitraum in welcher diese Verbindung gültig ist und die Wochentage, an denen der Flug stattfindet.

Die Ansicht „Weltkarte“ zeigt eine Weltkarte mit allen Flughäfen an, die aus Frankfurt angeflogen werden. Wird ein Flughafen angewählt, blendet die App die aus dem Punkt „Saisonflugplan“ bekannte Suchmaske ein, wobei der ausgewählte Flughafen bereits in die Suchmaske eingetragen ist.

MUC Airport

Auch der Flughafen München bietet eine eigene App im `Play Store` an. Der Startbildschirm, aufgeführt in Abbildung 2(b), ist in drei Teile, mit einer `Action Bar` am oberen Rand, aufgeteilt. Über die `Action Bar` ist, wie auch bei der App des Flughafens Frankfurt, ein seitlich aufklappendes Menü vorhanden, über welches auf Informationen über Flüge,



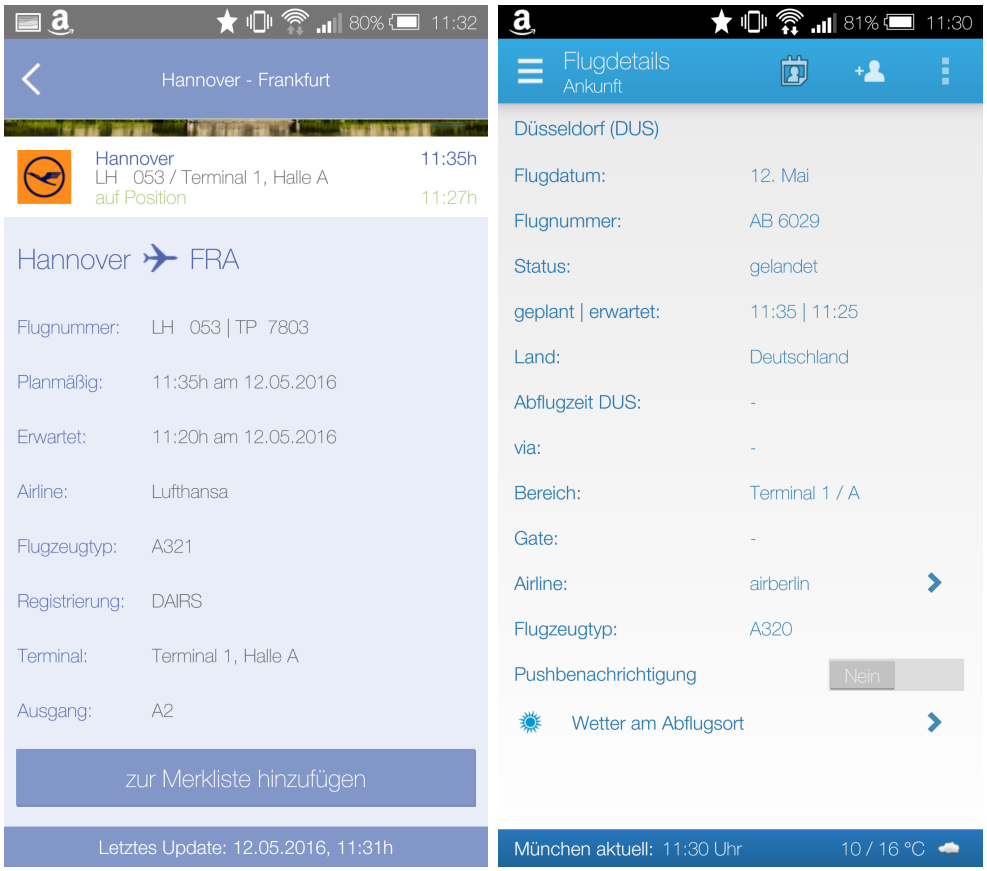
(a) Hauptansicht FRA Airport [17] (b) Hauptansicht MUC Airport [18]

Abbildung 2: Vergleich der Hauptansichten der Apps der Flughäfen Frankfurt und München

das Parken am Flughafen, Service- und Shoppingmöglichkeiten am Flughafen, Informationen über die App oder den Zugang zu einem personalisiertem Bereich zugegriffen werden kann. Der personalisierte Bereich ist, neben einem Link zu Telefonnummern des Flughafens, ebenfalls direkt über die `Action Bar` zu erreichen. Unter der `Action Bar` befindet sich eine Slideshow mit Bildern des Flughafens oder Werbung externer Unternehmen, welche bei Berührung zu der Website des Flughafens oder der werbetreibenden Unternehmen weiterleiten. Der mittlere Teil besteht aus einer Liste der nächsten An- oder Abflüge, welche an dieser Stelle mit Flugnummer, Ankunfts- beziehungsweise Abflugort und Uhrzeit angegeben werden. Nach einer Berührung eines Fluges öffnet sich eine Ansicht mit erweiterten Fluginformationen. Dazu gehören Flugnummer, geplante und erwartete Lande- beziehungsweise Startzeit, Terminal, Airline, Flugzeugtyp und Wetter am Abflug- beziehungsweise Ankunftsort. Außerdem lassen sich für jeden Flug Pushbenachrichtigungen aktivieren, welche Aktualisierungen zu dem Flug in der Benachrichtigungsleiste erscheinen lassen. Am unteren Rand befindet sich eine Leiste, die dauerhaft das aktuelle Wetter

und die Uhrzeit in München anzeigt.

In Abbildung 3 werden die erweiterten Fluginformationen von Frankfurt und München gegenübergestellt. Hierbei fällt das unterschiedliche Design auf, sowie das Logo der Fluggesellschaft und ein für den Ankunfts- beziehungsweise Abflugort charakteristisches Bild, welches in der `FRA Airport` App zusätzlich zum Text und den kleinen Logos angezeigt wird.



(a) Fluginformationen FRA Airport [17] (b) Fluginformationen MUC Airport [18]

Abbildung 3: Vergleich der Fluginformationen der Flughäfen Frankfurt und München

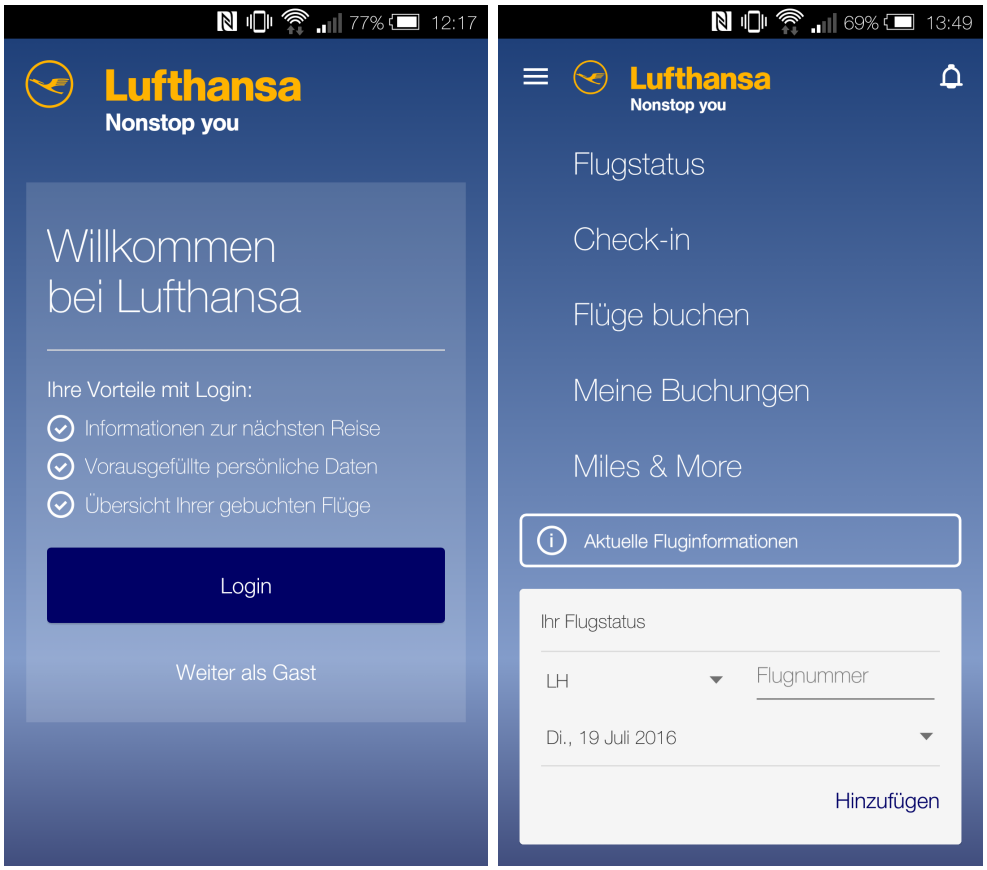
2.2.2 Transportdienstleister

Flugverkehr: Lufthansa

In Abbildung 4 befinden sich die Ansichten der `Lufthansa` App. Nach dem Start der `Lufthansa` App wird ein Willkommensbildschirm angezeigt, welcher die Möglichkeit zum Login oder der Fortsetzung als Gastzugang bietet. Bei Auswahl des Gastzugangs öffnet sich eine Übersichtsseite. In der `Action Bar` befindet sich, ähnlich zu den Apps `FRA Airport` und `MUC Airport`, ein ausklappbares Menü, das Logo und einen Werbespruch von `Lufthansa` anzeigt. Außerdem befindet sich daneben ein Symbol, welches neue Benachrichtigungen anzeigt und gleichzeitig als Link zur Übersichtsseite für Benachrichtigungen dient. Über das Menü sind folgende Funktionen erreichbar:

- Suche
- Login
- Buchungen, Bordkarten, Gepäckbelege
- Mitteilungen
- Angebote, Newsletter, Travel Guide, Miles & More
- Flugplan, Flugbuchung, Flugstatus und Check-in
- Persönliche Daten
- Info & Service, Kontakt, Feedback
- Einstellungen

Unter der Actionbar ist der Bildschirm in zwei Hauptbereiche aufgeteilt. Im oberen Bereich befinden sich Menüpunkte zum Abrufen eines Flugstatus, zum Online-Check-in, zur Buchung von Flügen, zur Übersicht von Flugbuchungen, zum Zugriff auf das Prämienprogramm Miles & More und zur Ansicht allgemeiner, aktueller Fluginformationen. Eine kurze Statusansicht befindet sich im unteren Teil. Dort muss zuerst über eine Suchmaske eine Flugnummer und ein Datum angegeben werden. Anschließend werden zusammengefasste Informationen des ausgewählten Fluges direkt auf der Übersichtsseite angezeigt. Es können auch mehrere Flüge zu dieser Statusseite hinzugefügt werden, zwischen welchen durch seitliches Scrollen gewechselt werden kann.



(a) Willkommensbildschirm (b) Hauptbildschirm als Gast

Abbildung 4: Ansichten der `Lufthansa` App [19]

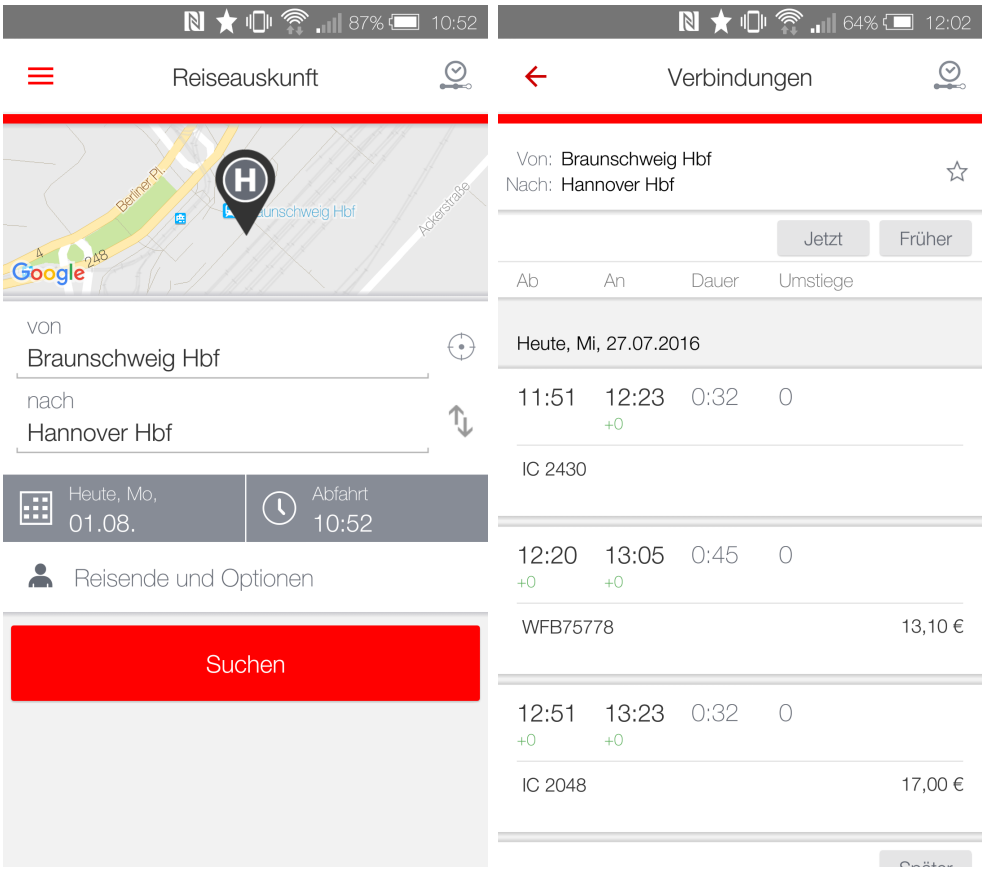
Schienenverkehr: Deutsche Bahn

Für den Bereich Schienenverkehr wird die App der Deutschen Bahn betrachtet. Die Deutsche Bahn besitzt laut eigenen Angaben im Jahr 2015 mit 70,8 % den größten Anteil des Schienenpersonennahverkehrs in Deutschland, gemessen in Zugkilometern [20]. Im Play Store hat die Deutsche Bahn 15 Apps veröffentlicht. Dazu gehören neben Programmen zur Navigation auch Apps zur Information über Baustellen, Zugstatus und Bahnhöfe sowie Spiele. In dieser Arbeit wird lediglich auf den „DB Navigator“ eingegangen, da es sich hierbei um die App zur Reiseplanung der Deutschen Bahn handelt.

Nach dem Start des Programms erhält man direkt Zugang zu einer Suchmaske [Abbildung 5(a)]. Über diese Suchmaske lassen sich Start- und Zielort, Uhrzeit und Datum sowie Informationen zu den Reisenden und weitere Optionen wie Zwischenhalte, gewünschte Verkehrsmittel oder gewünschte Umsteigezeit einstellen. In der Action Bar am oberen Rand befindet sich, wie bei den Apps der beiden Flughäfen und der `Lufthansa`, ein ausklappbares Menü. Das Menü bietet Zugriff auf Einstellungen zur Auskunft und Buchung von Reisen, auf eigene Tickets oder Ticketbuchung, auf Informationen über Verspätungen,

Carsharing, die BahnCard oder die Herausgeber der App.

Bei der Suche über die Suchmaske öffnet sich nach Eingabe der Daten eine Verbindungsübersicht [Abbildung 5(b)]. Die passenden Verbindungen werden chronologisch mit Abfahrts- und Ankunftszeit, Dauer, Anzahl der Umstiege, Zugnummer und Preis angezeigt. Bei Berührung wird auf die Seite „Reiseplan“ weitergeleitet. Neben den Informationen aus der Verbindungsübersicht werden zusätzlich noch Zwischenhalte und Gleise eingeblendet. Über einen Button kann die ausgewählte Verbindung zu einer persönlichen Reiseübersicht oder einem externen Kalender hinzugefügt, in einer Karte angezeigt oder nach aktuellen Alternativen gesucht werden.



(a) Suchmaske (b) Übersicht der Verbindungen

Abbildung 5: Ansichten der App der `Deutschen Bahn` [21]

ÖPNV

Für den ÖPNV in Deutschland gibt es keine offizielle App eines Transportdienstleisters. Der Grund hierfür ist hauptsächlich in der Struktur des Nahverkehrs in Deutschland zu finden. Anstelle eines bundesweiten Unternehmens wird er von vielen lokalen, voneinander unabhängigen Nahverkehrsunternehmen dargstellt. Etwa 450 dieser Nahverkehrsunternehmen sind im Verband Deutscher Verkehrsunternehmen organisiert. Dabei erbringen sie über 90% der Leistungen im ÖPNV-Bereich in Deutschland [22]. Viele Unternehmen haben eine eigene App oder haben sich zu Verbünden zusammengeschlossen, die wiederum ihre eigenen Apps anbieten. Untereinander sind die Apps unabhängig und in Funktion und Design unterschiedlich. Private Anbieter versuchen, die Daten möglichst vieler Nahverkehrsunternehmen in einer App zu sammeln und einheitlich darzustellen, den größten Datenumfang bietet dabei das Programm „Öffi“. Auch die Bahn zeigt in ihrer App `DB Navigator` Verbindungen des ÖPNV an. Eine Liste aller Nahverkehrsunternehmen, die ihre Daten dafür zur Verfügung stellen, ist allerdings nicht veröffentlicht.

2.2.3 Drittanbieter

Der Markt der Drittanbieter ist breit gefächert. Aus diesem Grund werden an dieser Stelle nur die Besonderheiten ausgewählter Apps angesprochen.

Öffi

Öffi ist eine App von Andreas Schildbach, welche die Fahrplandaten mehrerer Nah- und Fernverkehrsgesellschaften in einem Programm zusammenfasst. Dabei setzt sie sich aus drei Teilen zusammen:

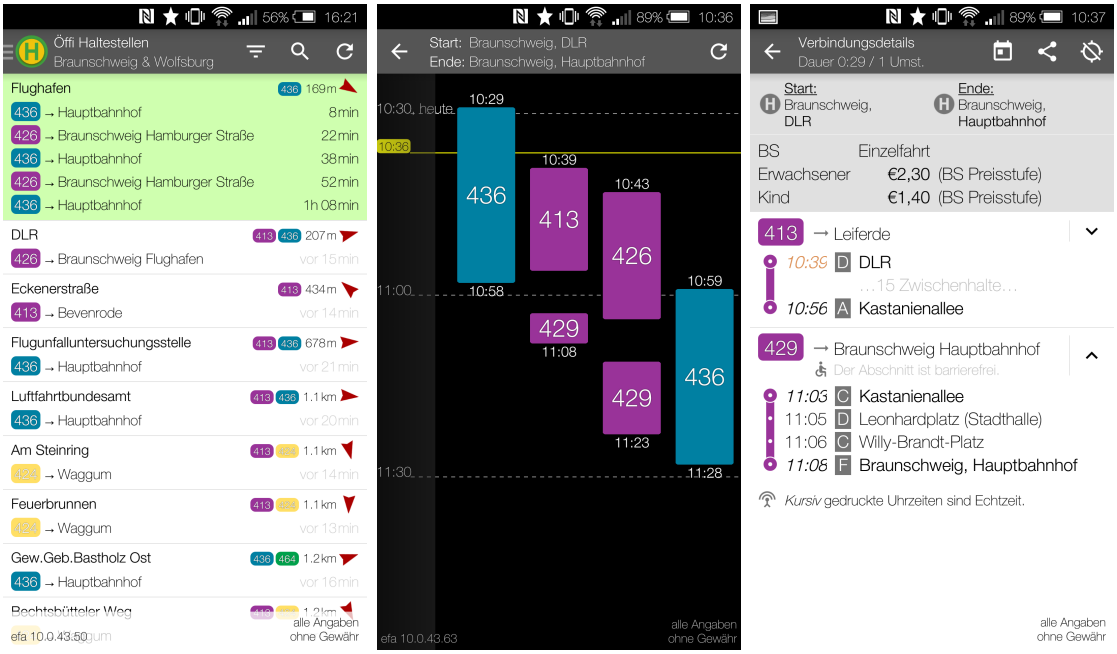
- Haltestellen: Zeigt eine Übersicht der nächstgelegenen Haltestellen und deren nächsten Abfahrten an. Nach der Auswahl einer Haltestelle werden die nächsten Abfahrten von dieser eingeblendet(Abbildung 6(a)).

- Verbindungen: Bietet eine Suchmaske zum Einstellen einer Verbindung. Dafür werden Start- und Zielpunkt sowie gewünschte Abfahrt- oder Ankunftszeit benötigt. Anschließend werden Verbindungsoptionen in einer Zeitleiste angezeigt (Abbildung 6(b)). Dort besitzt jede Verbindung eine Spalte. Die Verbindungen sind chronologisch von links nach rechts angeordnet, die Zeitleiste verläuft senkrecht nach unten. Ein Verbindungsabschnitt ist mit einem farbgien Block dargestellt und enthält als

Beschriftung die Liniennummer. Durch Berührung kann die ausgewählte Verbindung in einer erweiterten Verbindungsübersicht angezeigt werden (Abbildung 6(c)). In dieser Ansicht befindet sich eine Übersicht der Reise mit Kosteninformationen. Darunter werden die Verbindungsabschnitte mit Linienbezeichnungen nacheinander angezeigt. Dabei sind bei längeren Verbindungen nur Start- und Endhaltestelle angegeben. Jeder Verbindungsabschnitt kann erweitert werden, um die vollständigen Informationen dazu anzuzeigen.

- Netzpläne: Zeigt die Liniennetze diverser Verkehrsbetriebe als Bild an.

Diese Programmteile sind untereinander verknüpft und können sich gegenseitig aufrufen. Wird beispielsweise unter „Haltestellen“ eine Haltestelle angewählt, können über ein Menü die Punkte „Verbindungen von hier“ oder „Verbindungen nach hier“ aufgerufen werden. Diese leiten zur bereits teilweise ausgefüllten Suchmaske vom Programmteil „Verbindungen“ weiter. In Deutschland deckt `Öffi` den Großteil der Nahverkehrsgesellschaften und die Deutsche Bahn ab.



(a) Unterprogramm Haltestellen (b) Zeitleiste für gesuchte Verbindung (c) Verbindungsdetails

Abbildung 6: Ansichten der `Öffi` App [23]

Google Maps

Seit 2007 bietet das amerikanische Unternehmen `Google` in seiner Navigationssoftware `Google Maps` auch Fahrplandaten der öffentlichen Verkehrsmittel an. `Google` erweitert sein Angebot dabei schrittweise auch um Echtzeitdaten [24]. Die Abdeckung, vor allem in Deutschland, ist nicht komplett, da `Google` mit jedem Verkehrsunternehmen Verträge über die Nutzung der Daten abschließen muss. Nicht alle Unternehmen stimmen dem zu [25]. `Google Maps` hebt sich von den anderen Apps ab, da es ursprünglich zur Navigation für PKW genutzt wurde, bevor es um zusätzliche Verkehrsträger erweitert wurde.

Qixxit

Qixxit, seit 2014 verfügbar und von der `Deutschen Bahn` entwickelt, legt den Fokus auf Intermodalität [26]. Neben einem Start- und Zielort und dem Zeitpunkt der Reise zeigt die App ein Menü zum Auswählen gewünschter Verkehrsträger an. Zusätzlich zu privaten Verkehrsträgern wie PKW, Fahrrad oder dem Mietwagen, bietet `Qixxit` auch die öffentlichen Alternativen Bus, Straßen-/U-Bahn, S-Bahn/RE, Taxi, Mitfahrgelegenheiten, Carsharing, Mietrad, Fernbus, Flugzeug und Züge der `Deutschen Bahn` zur Wegberechnung an. Nach der Suche werden Verbindungen mit den angewählten Verkehrsträgern von der App angezeigt. Dabei wird neben den Verbindungsinformationen Dauer, Preis und CO_2-Ausstoß der Verbindungen angegeben.

Besonders im Bereich der Flughäfen unterscheiden sich die Apps bei ähnlichem Funktionsumfang voneinander. Bei den Transportdienstleistern zeigt die `Deutsche Bahn` Ansätze von Intermodalität und bezieht den ÖPNV in die Reisekette ein. `Quixxit` sticht im Bezug auf intermodale Reisen besonders mit der Menge an Verkehrsträgern und Verknüpfung dieser aus dem Angebot heraus.

2.3 Verfügbare APIs

Eine Möglichkeit für Unternehmen Daten zur Verfügung zu stellen sind Application Programmable Interfaces (APIs). Die Struktur dieser Schnittstellen zur Datenübertragung wird in Unterabschnitt 3.1 erklärt. Die verfügbaren APIs können, analog zu den Apps, in die Gruppen der Flughäfen, Transportdienstleister und Drittanbieter aufgeteilt werden.

2.3.1 Flughäfen

Im Bereich der Flughäfen bietet der Frankfurter Flughafen eine API mit Echtzeitdaten an. Sie wird in dieser Arbeit zur Implementierung verwendet und ist in Kapitel 3.1.3 genauer beschrieben.

2.3.2 Transportdienstleister

Lufthansa

Die `Lufthansa` bietet über ihr Portal `https://developer.lufthansa.com` Zugang zu einer Palette an APIs. Dabei werden die einzelnen Schnittstellen in die Bereiche *Reference Data*, *Operations* und *Offers* aufgeteilt. Unter dem Bereich *Reference Data* befinden sich APIs zur Informationsbeschaffung über Länder, Städte, Flughäfen, die nächsten Flughäfen, Fluglinien und Flugzeuge, während *Offers* Informationen über Lounges an den Flughäfen und Sitzpläne der Flugzeuge bietet. Lounges an Flughäfen sind Bereiche oder Räume, die nur für bestimme Passagiergruppen zugänglich sind. Meist ist der Zugang in höherpreisigen Flugtickets inbegriffen, die Lounge bietet dabei besonderen Komfort für Passagiere, oftmals auch kostenlose Verpflegungsmöglichkeiten. Der Bereich *Operations* besteht aus fünf APIs. *Arrivals Status* und *Departures Status* fordern als Übergabewert einen Flughafen und einen Start- und Endzeitpunkt und geben alle Ankünfte oder Abflüge von `Lufthansa` an diesem Flughafen in dem definierten Zeitraum zurück. *Schedules* gibt alle Verbindungen zwischen zwei Flughäfen an. Dabei ist eine Datumsangabe notwendig, optional können auch noch Uhrzeit und die Beschränkung auf Direktflüge angegeben werden. Die APIs *Flight Status* und *Flight Status by Route* geben Fluginformationen zu einem definierten Flug zurück. Diese Definition erfolgt entweder über Flugnummer und Datum oder über Start- und Zielflughafen sowie Datum.

Auf der Internetseite wird eine ausführliche Dokumentation zur Schnittstelle auf Englisch angeboten. Sie enthält eine Beschreibung des Aufbaus der Anfragen an die Schnittstelle und ihrer Antworten. Es werden Fehlermeldungen erklärt, bekannte Fehler beschrieben und ein Status angegeben, welcher anzeigt, ob die Schnittstelle zur Verfügung steht. Außerdem gibt es einen *API Playground*, in dem direkt im Internetbrowser Anfragen an die API getestet werden können.

Deutsche Bahn

Seit November 2015 setzt die Deutsche Bahn mit `http://data.deutschebahn.com/` auf

Open Data [27]. Open Data beschreibt die Veröffentlichung von Daten. Dabei unterscheidet die Bahn zwischen Datensätzen und APIs. Datensätze sind Tabellen mit Daten, die nur als kompletter Datensatz heruntergeladen werden können. Die Deutsche Bahn bietet 15 solcher Datensätze an, darunter beispielsweise Informationen zu Reisezentren, Haltestellen, Serviceeinrichtungen oder dem Mobilfunkempfang entlang des Fernverkehrsnetzes. Die Aktualisierungsrate ist, je nach Datensatz, monatlich oder jährlich. Informationen zu Parkplätzen, Aufzügen und dem Fahrplan werden als API angeboten. Dies bietet die Möglichkeit über Abfragen nur ausgewählte Teile des Datensatzes zu empfangen. Für diese Arbeit relevant ist die Fahrplan API. Sie bietet den Soll-Fahrplan des Fernverkehrs der Deutschen Bahn an. Im Gegensatz zu der `Lufthansa` oder `Fraport` Schnittstelle werden somit keine Echtzeitdaten, sondern nur geplante Abfahrten zur Verfügung gestellt. Die Schnittstelle enthält keine Informationen über Nah- oder Regionalverkehr. Es bestehen die Möglichkeiten zur Bahnhofssuche, Abfahrts- und Ankunftstafel eines Bahnhofs oder Zuglauf einer Strecke. Bei der Bahnhofssuche werden zu einem Suchbegriff passende Bahnhöfe zurückgegeben. Abfahrt- und Ankunftstafel geben Informationen zu Abfahrten oder Ankünften an einem ausgewählten Bahnhof und Zeitraum zurück. Bei der Anfrage nach dem Zuglauf wird eine Antwort mit den Halteinformationen eines, bei der Abfrage übergebenen, Zuges geliefert.

ÖPNV

Der ÖPNV besitzt keine einheitliche Schnittstelle. Da jedes Nahverkehrsunternehmen selber für seine Daten zuständig ist, ist auch die Verfügbarkeit stark unterschiedlich. Die Verkehrsbetriebe Berlin-Brandenburg und Hamburg bieten beispielsweise APIs mit Echtzeitinformationen an, während die meisten anderen Betriebe ihre Daten nur auf eigenen Internetpräsenzen anbieten und keine Schnittstelle zur Verfügung stellen [28] [29].

2.3.3 Drittanbieter

Der Markt für APIs im Bereich der Verkehrsdienstleistungen ist groß. Alleine auf `www.programmableweb.com`, einer Sammelstelle für APIs, sind 733 Schnittstellen mit dem Wort *Transportation* markiert [30]. Neben kostenpflichtigen Schnittstellen, wie beispielsweise von `FlightStats`, sind auch kostenlose Varianten verfügbar. Dabei unterscheiden sie sich stark in Daten- und Funktionsumfang. Eine Sonderstellung nimmt hier die *Directions API* von `Google` ein. Wird mit dieser APIs eine kostenfreie Software entwickelt und die

Schnittstelle nicht mehr als 2500 mal pro Tag aufgerufen, ist die Nutzung kostenlos. Dabei gewährt sie Zugriff auf den großen Datenbestand von `Google` im Bereich des Fern- und Nahverkehrs [31].

3 Vorgehensweise

Im Rahmen dieser Arbeit sollen die Flugdaten des Frankfurter Flughafens in die Leitstandsumgebung `Optimode.net` eingepflegt werden. Dazu werden zunächst die verwendeten Werkzeuge und Techniken beschrieben, die bei der Implementierung verwendet werden. Die `Fraport API` wurde ausgewählt, da sie sich mit der Beschränkung auf einen Flughafen am Besten zur Demonstration eignet. Die Schnittstelle bietet alle Flugbewegungen aus Frankfurt und kann somit einen zusätzlichen Flughafen ins `Optimode.net` System integrieren.

3.1 Application Programming Interface (API)

Im Bereich der Web Services sind APIs als Schnittstelle von Daten zwischen verschiedenen Diensten definiert. So kann ein Anbieter seine ursprünglich internen Daten über eine API anderen Diensten zur Verfügung stellen. Durch den definierten Aufbau der Schnittstelle steht ihr Inhalt anderen Diensten zur Verfügung. Durch die Verwendung weit verbreiteter Formate, besteht die Möglichkeit der einfachen Implementation der API in einer Vielzahl von Programmiersprachen [32, S.25]. Werden mehrere, offene APIs genutzt, um daraus eine neue Applikation zu gestalten, wird dies Mashup genannt [33, S.143]. Der Großteil der offiziellen APIs basiert auf einem der Webservice-Arichtekturstilen XML-RPC, SOAP und REST [32, S.29]. Die Schnittstelle des in dieser Arbeit betrachteten Flughafens Frankfurt nutzt den REST-Arichtiekturstil, von welchem eine nähere Beschreibung folgt.

3.1.1 Funktionsweise einer RESTful API

Representational State Transfer (REST) beschreibt die Architektur von netzwerkbasierter Software. Die Grundidee ist, dass ein Webservice eine Ansammlung von Ressourcen ist. Dabei werden sechs Eigenschaften vorgeschrieben:

- **Client-Server-Modell:** Bei einer Kommunikation nach diesem Modell gibt es immer zwei Teilnehmer: Ein *Client* sendet eine Anfrage, ein *Server* beantwortet diese. Diese Rollen gelten nur für einen Nachrichtenaustausch und können sich beim nächsten Schritt umkehren.

- **Zustandslosigkeit:** Jede Anfrage ist unabhängig. Sie enthält alle Informationen und benötigt keine Informationen vorheriger Anfragen.

- **Caching:** Caching beschreibt das Zwischenspeichern und Wiederverwenden von Daten. Bei einer Anfrage muss mitgegeben werden, ob Daten vom Client zwischengespeichert werden dürfen. Diese würden bei einer erneuten Anfrage benutzt, ohne die Anfrage erneut an den Server zu senden.

- **Einheitliche Schnittstelle:** Der Zugriff auf die Ressourcen eines Webservices erfolgt über eindeutige Uniform Resource Identifiers (URIs). Diese Zeichenketten sind Ressourcen zugeordnet. An den Client werden Repräsentationen von Ressourcen gesendet. Eine Ressource kann in verschiedenen Formatierungen oder Sprachen versendet werden und somit verschiedene Repräsentationen besitzen. REST-Nachrichten können Links zu weiteren Ressourcen enthalten und müssen selbstbeschreibend sein.

- **Mehrschichtige Systeme:** Der Client muss nicht direkt mit dem Server kommunizieren. Zwischen Client und Server können mehrere Hard- oder Softwareschichten liegen. Dadurch kann ein System, welches auf REST beruht, einfacher erweitert werden.

- **Code-on-Demand:** Diese Eigenschaft ist optional. Code-on-Demand beschreibt die Eigenschaft, dass Programmcode vom Client nachgeladen werden kann, sobald er gebraucht wird [34, S.76ff.].

Diese Eigenschaften werden vom Hypertext Transfer Protocol (HTTP) erfüllt, welches häufig für REST-konforme APIs genutzt wird. Auch die Fraport API benutzt dieses Protokoll. Es gibt zwei Arten von HTTP Nachrichten: *Requests* sind Anfragen vom Client an den Server, *Responses* sind die Antworten des Servers.

Requests beginnen mit einer Zeile, in welcher die Protokollversion, die HTTP-Methode und die URI der angeforderten Ressource angegeben wird. Für APIs werden hauptsächlich die Methoden *GET* und *POST* verwendet [32, S.59]. Die Methode beschreibt, wie der Client mit dem Server interagieren möchte. Während *GET-Requests* nur die Repräsentation einer Ressource anfordern, können mit der *POST*-Methode Daten auf dem Server verändert werden. Danach folgt der *Header* der Nachricht. Er enthält Namen-Werte-Paare, welche für den *Request* benötigt werden. Der dritte Teil eines *Requests* ist der *Body*. Hier befinden sich Daten, welche dem Server übergeben werden sollen. Bei einem *GET-Request* ist der *Body* meistens leer [35] [36, S.7].

Der Aufbau der Serverantwort ähnelt der eines *Requests*. Er besteht ebenfalls aus den drei Teilen Statuszeile, *Header* und *Body*. In der Statuszeile befinden sich Protokollversion,

Tabelle 1: HTTP-Statuscodes

HTTP-Code	Text	Bedeutung
200	OK	*Request* erfolgreich
400	Bad Request	Fehlerhafte Anfrage. Der Fehler liegt beim Client
500	Internatl Server Error	Fehler beim Server, kann nicht vom Client gelöst werden
301	Moved Permanently	Ressource ist unter neuem URI zu finden
404	Not Found	Keine Ressource unter dem URI auffindbar
410	Gone	Unter dem URI war eine Ressource, diese ist inzwischen nicht mehr auffindbar
409	Conflict	Der *Request* würde Ressourcen inkonsistent zurücklassen und ist daher nicht erlaubt

HTTP-Statuscode und ein kurzer Text zum Statuscode. Je nach Ergebnis der *Response*, gibt der Server einen codierten Status zurück. Aufgrund dieses Status kann direkt am Anfang der Antwort abgelesen werden, ob die Anfrage erfolgreich war oder ein Problem aufgetreten ist. In Tabelle 1 sind die häufigsten HTTP-Statuscodes aufgelistet.

Der *Header* ist analog zu dem des *Requests* aufgebaut. Er enthält Namen-Werte-Paare, die Informationen über die Antwort, beispielsweise in welchem Format sie geliefert wird, angeben. Im *Body* befindet sich die Repräsentation der angeforderten Ressource.

3.1.2 Formatierung von APIs

Daten können in verschiedenen Formaten übertragen werden. Diese vorher festgelegten Formate sind notwendig, damit der Empfänger versteht, wie die Daten aufgebaut sind und diese interpretieren kann. Betrachtet werden die zwei häufig verwendeten Formate JSON und XML. Dazu wird ein Beispieldatensatz, der in Abbildung 7 visualisiert ist, in den beiden Formaten angegeben. Der Datensatz besteht aus drei Personen, die jeweils die Eigenschaften „Vorname“ und „Nachname“ besitzen und zur Gruppe „Personen“ gehören.

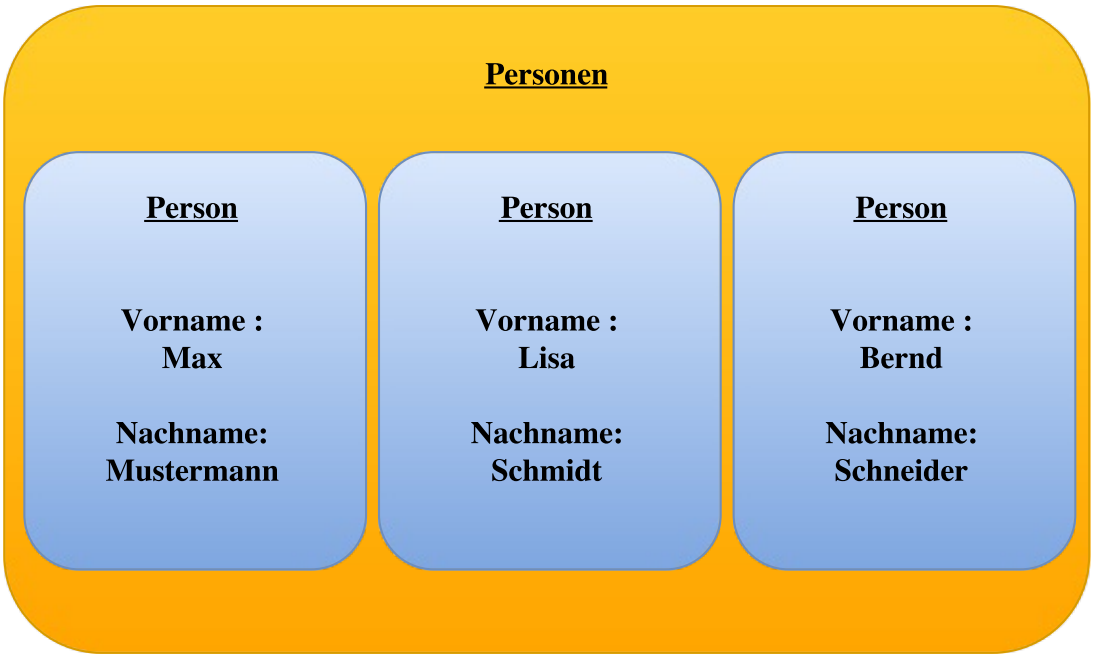


Abbildung 7: Beispieldatensatz Personen

JSON

Javascript Object Notation (JSON) ist ein Datenformat zum Datenaustausch zwischen verschiedenen Programmiersprachen. Es wurde aus der Skriptsprache JavaScript entwickelt, welche weit verbreitet ist. JSON ist für den Menschen einfach zu lesen und in seinem Aufbau an die C-Sprachfamilie angelehnt. Der Aufbau setzt sich aus Name-Wert-Paaren und Arrays zusammen, wobei mehrere Name-Wert-Paare in einem Objekt, gekennzeichnet durch geschweifte Klammern { }, zusammengefasst werden können. Als Werte werden strings, Zahlen, Objekte, arrays, die booleschen Werte `true` und `false` oder `null` akzeptiert. `null` beschreibt einen leeren, noch nicht zugewiesenen Wert. [2]

Im folgenden Beispiel wird der Beispieldatensatz in JSON dargestellt (Abbildung 8). Jedes Objekt vom Typ `Person` besitzt zwei Name-Werte-Paare : `Vorname` und `Nachname`, beides in diesem Fall vom Typ string. Die Objekte werden mit Kommata getrennt in einem array, gekennzeichnet durch die eckigen Klammern [und], angegeben. Umschlossen wird der Code von geschweiften Klammern.

```
{
        "Personen":[
                {"Vorname":"Max",
                "Nachname":"Mustermann"},
                {"Vorname":"Lisa",
                "Nachname":"Schmidt"},
                {"Vorname":"Bernd",
                "Nachname":"Schneider"}]
}
```

Abbildung 8: Beispielcode JSON

XML

Eine Alternative zur Bereitstellung von Daten bietet die Extensible Markup Language (XML). XML wurde von der World Wide Web Consortium (W3C) standardisiert und ist mit der Hypertext Markup Language (HTML) verwandt. Es wird, wie auch in JSON, nur die Struktur und der Aufbau der Daten beschrieben, jedoch nicht die Darstellung dieser. Charakteristisch für XML sind Elemente, welche durch Tags geöffnet und geschlossen werden müssen. Diese Tags sind durch die Zeichen < und > umschlossen, während das Tag zum Schließen des zu beschreibenden Elements seinen Namen mit vorangestelltem Schrägstrich enthält. Zwischen den beiden Tags steht der Inhalt des Elements. Dieser kann aus Text oder weiteren, verschachtelten Elementen bestehen [37]. Der Beispieldatensatz (Abbildung 9) wird über die Elemente `Vorname` und `Nachname` dargestellt, welche durch ein `Person`-Element umschlossen werden, welches wiederum das Element `Personen` umschließt.

```
<Personen>
        <Person>
                <Vorname>Max</Vorname>
                <Nachname>Mustermann</Nachname>
        </Person>
        <Person>
                <Vorname>Lisa</Vorname>
                <Nachname>Schmidt</Nachname>
        </Person>
        <Person>
                <Vorname>Bernd</Vorname>
                <Nachname>Schneider</Nachname>
        </Person>
</Personen>
```

Abbildung 9: Beispielcode XML

Wohlgeformte XML

Ein XML-Dokument muss bestimmten Regeln folgen, um wohlgeformt zu sein. Dies ist notwendig, um von den meisten XML-Parsern bearbeitet werden zu können. Ein Parser ist ein Programm, welches Text als Eingabeparameter erhält, diesen verarbeitet und in eine Programmstruktur umwandelt. Ist ein XML-Dokument nicht wohlgeformt, kann es bei diesem Vorgang zu Fehlermeldungen kommen [37]. Um wohlgeformt zu sein, muss ein Dokument folgende Bedingungen erfüllen:

- Es muss mindestens ein Element besitzen.
- Es muss genau ein Wurzelelement besitzen. Dieses Element muss als erstes geöffnet werden und als letztes geschlossen werden. Es muss außerdem alle anderen Elemente beinhalten.
- Die Elemente müssen korrekt verschachtelt sein. Das zuletzt geöffnete Element muss als erstes wieder geschlossen werden. Elemente dürfen nicht über Kreuz geöffnet und geschlossen werden.
- Jedes geöffnete Element muss auch wieder geschlossen werden.
- Die Namen der Elemente dürfen nicht mit einer Ziffer beginnen und dürfen nur Ziffern, Buchstaben, Bindestriche und Punkte enthalten [38].

3.1.3 Aufbau der Fraport API

Der Frankfurter Flughafen stellt Benutzern seiner Schnittstelle Echtzeitdaten aller Flugbewegungen zur Verfügung. Neben Zeiten zum An- und Abflug werden auch Informationen zum Flugzeug, Gate, Check-In Schalter oder Gepäckband minütlich aktualisiert.

Die Homepage der Datenschnittstelle des Frankfurter Flughafens ist über `https://developer.fraport.de/store/` zu erreichen. Auf dieser Seite befindet sich die Möglichkeit zur Registrierung und Anmeldung, welche für das Nutzen der Schnittstelle nötig ist. Nach der erfolgreichen Anmeldung besteht die Möglichkeit, sich Zugangsschlüssel herzustellen. Diese Schlüssel müssen bei jeder Anfrage an die Schnittstelle mitgesendet werden, über ihn kann der Betreiber der Schnittstelle die Anfrage einem Benutzer zuordnen. Neben der Erstellung des Schlüssels gibt es Möglichkeiten zur Verwaltung der Schlüssel und Zuordnung dieser zu Programmen, eine Konsole zum Testen von Anfragen und eine Dokumentation.

Die Schnittstelle bietet drei GET Anfragemöglichkeiten an. Dabei muss die Basis-URL `https://developer.fraport.de/api/flights/1.0/` jeweils vervollständigt werden :

- **flight/{airport}/arrival :** Gibt alle Ankünfte an einem Flughafen zurück. Der Flughafen wird als Parameter als Drei-Letter-Code übermittelt.
- **flight/{airport}/departure :** Gibt alle Abflüge von einem Flughafen zurück.
- **flightDetails/{airlineCode}/{flightNumber}/{origin_flight_date} :** Gibt nur die Informationen zu einem bestimmten Flug zurück. Dieser wird über Flugnummer und Abflugdatum als Parameter bestimmt.

Sofern nicht anders angegeben, wird die Antwort im JSON Format gesendet. Die einzelnen Flüge werden als `flight` Objekt in einem Array zurückgegeben. Im späteren Verlauf der Arbeit werden diese Objekte in C# implementiert. Ein Klassendiagramm ist in Abbildung A1 dargestellt und demonstriert den Objektaufbau. Jedes dieser Objekte enthält vier Strings : Die 3-Letter-Codes zum Abflug- und Zielflughafen, das Startdatum und den Flugstatus. Der Flugstatus ist dabei durch Buchstaben codiert, eine Übersetzung befindet sich in der Dokumentation der Schnittstelle. In Tabelle 2 sind alle Codes aufgeführt.

Neben diesen vier strings sind noch Objekte für die Fluglinie, den Flugzeugtyp, die Flugnummer, mögliche Codeshares und An- beziehungsweise Abflug vorhanden. Das Objekt zur Beschreibung der Fluglinie setzt sich aus drei strings zusammen, in denen die ICAO und IATA Codes und der Klarname der Fluglinie gespeichert sind. Während die IATA Zwei-Letter-Codes verwendet, benutzt die ICAO ausschließlich Drei-Letter-Codes. Für den Flugzeugtyp werden ebenfalls drei strings benutzt. Sie beschreiben den ICAO Code des Flugzeugs, den Modellnamen und die Registrierung, welche auch auf jedem zi-

Tabelle 2: Abkürzungen für Flugstatus

Buchstabe	Bedeutung
B	Busankunft in Halle B
D	Diversion
I	Undefinierte Verspätung
L	Abgebrochener Start
M	Flug auf nächsten Tag verlegt
S	Flug gestrichen
X	Gestrichener Flug, für den es einen Ersatz geben könnte
Y	Rückkehr zum Stellplatz
Z	Rückkehr zum Vorfeld

vilen Flugzeug aufgedruckt sein muss. Codeshares setzen sich aus den gleichen Attributen wie die Flugnummer zusammen: Einem Code für die Fluglinie, einer Nummernfolge für die Flugnummer und einem optionalen Anhang.

Die Objekte für An- und Abflug sind nahezu identisch. Von der Schnittstelle werden nur die Daten in Frankfurt erfasst, daher ist bei jeder Verbindung entweder das Objekt für den Anflug oder Abflug ausgefüllt. Beide besitzen Zeitstempel für geplante, erwartete und tatsächliche Start- beziehungsweise Landezeiten. Außerdem sind bei beiden Variablen für Terminal und Gate vorhanden. Während bei der Ankunft zusätzlich ein Objekt zur Gepäckabholung vorhanden ist, welches das Gepäckband und die geschätzte Ankunft der Gepäckstücke am Band beinhaltet, besitzt das Abflugsobjekt einen Zeitstempel zur Boardingzeit und ein Objekt, welche Check-In-Schalter für den Flug benutzt werden können.

3.2 Verwendete Entwicklungsumgebung

Die Implementierung wird in Visual C# mithilfe von Visual Studio 2015 vorgenommen. Visual Studio ist eine integrierte Entwicklungsumgebung von Microsoft und unterstützt unter anderem die Programmiersprachen C#, Visual Basic, F#, C++, JavaScript, TypeScript und Python. Als integrierte Entwicklungsumgebung versteht man Software, die mehrere Werkzeuge zur Entwicklung eines Programms zusammenfasst. Hauptbestandteile sind dabei ein Texteditor, in welchen der Quelltext für das Programm eingegeben wird, ein Compiler zur Übersetzung des Quelltextes in Maschinensprache, ein Debugger zum identifizieren von Fehlern im Quelltext und weitere Werkzeuge zum optimieren des Quellcodes [39].

Visual C# ist die Implementierung Microsofts der Programmiersprache C#. Die Sprache basiert auf .NET Framework und kann auf die Klassenbibliothek von .NET Framework zugreifen, welche einen großen Umfang an vorgefertigtem Code und Klassen enthält. C# ist komplett objektorientiert. Ein Objekt ist eine Instanz einer Klasse. Eine Klasse besteht aus Methoden und Eigenschaften. Methoden beinhalten den ausführbaren Code der Klasse, während die Eigenschaften aus Variablen bestehen, welche die Klasse beschreiben. Ein Objekt kann aufgrund dieses Bauplans einer Klasse erzeugt und auch während der Laufzeit des Programms wieder zerstört werden. Es kann mehrere Objekte einer Klasse geben, ein Objekt muss jedoch immer einer Klasse entstammen [40].

3.3 Datenbank

Ein zentraler Bestandteil von `Optimode.net` ist die Datenbank. Eine Datenbank ist eine strukturierte Sammlung von Daten. Verwendet wird eine relationale Datenbank. Dabei werden die Daten in mehrere Tabellen aufgeteilt, anstatt sie in einem großen Datenpool zu speichern. Die Datenbank speichert Daten dauerhaft und bietet eine Suchfunktion. Der Zugriff auf die Datenbank erfolgt durch ein Datenbankmanagementsystem (DBMS), im Fall von `Optimode.net` ist dieses MySQL. Das DBMS verwaltet dabei die Zugriffe auf die Datenbasis und liefert Ergebnisse zurück [Abbildung 10] [41, S.40ff.]. MySQL ist Open Source, kann somit kostenlos heruntergeladen und verändert werden. Zum Suchen und Verändern von Daten wird die Standard Query Language (SQL) benutzt. Diese Sprache ist durch einen ISO-Standard festgelegt und wurde 1986 entwickelt. Ein Vorteil von MySQL ist der MySQL Connector/NET. Er bietet einen direkten Zugang für .NET-Programmiersprachen, wozu auch die verwendete Programmiersprache C# gehört, auf das DBMS. Dadurch werden Datenbankabfragen aus dem Programmcode vereinfacht.

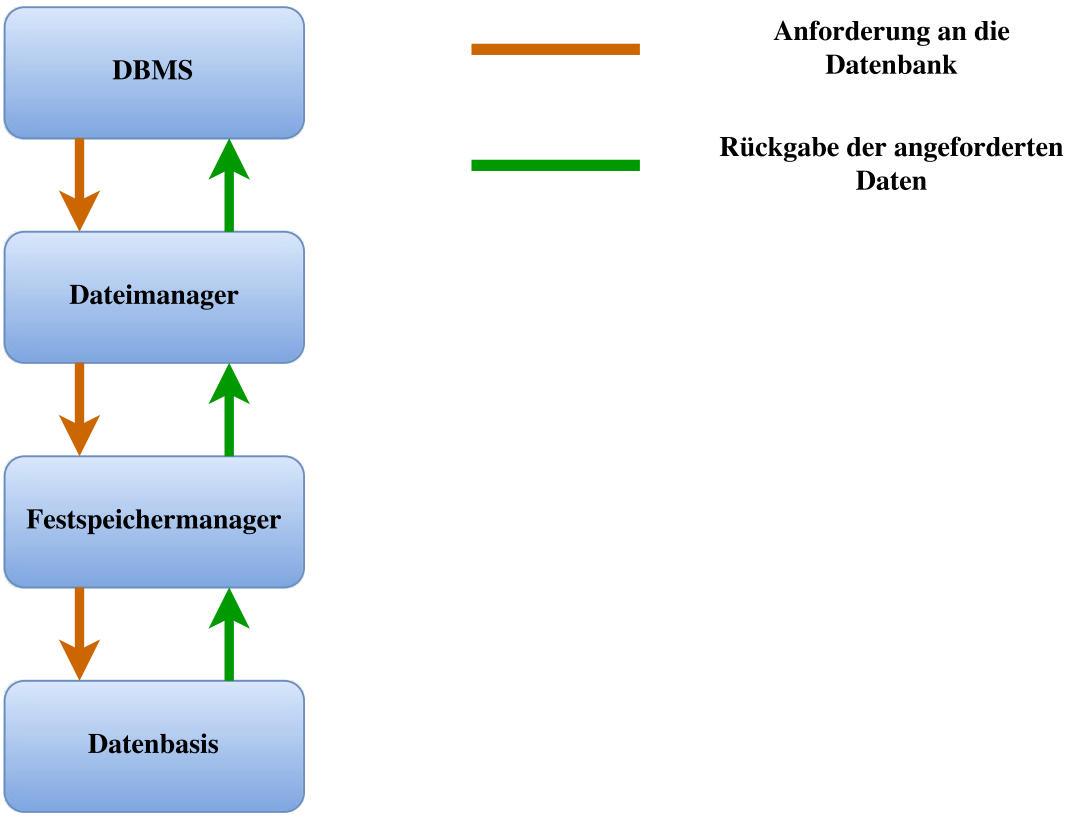


Abbildung 10: Zugriff auf eine Datenbank

4 Implementierung

Das Programm besitzt den Namen `FRAtoOptimode` und wird als Konsolenanwendung für `Microsoft Windows` geschrieben. Da während der Laufzeit des Programms keine Eingaben vom Benutzer notwendig sind, kann auf eine grafische Oberfläche verzichtet werden. Zusätzlich wird die Kompatibilität erhöht, da keine zusätzlichen Bibliotheken für die Oberfläche benötigt werden.

Logisch zusammenhängende Programmabschnitte werden in einzelne Methoden ausgelagert. Es entsteht eine optische Abgrenzung der Abschnitte, was den Quelltext übersichtlicher und verständlicher erscheinen lässt. Durch selbsterklärende Benennungen der Methoden kann erreicht werden, dass Personen, die nicht an der Entwicklung des Programmteils beteiligt waren, schnell Verständnis für die Funktion des Abschnitts erlangen. Außerdem können Teile des Quelltexts an anderer Stelle im Programm wiederverwendet werden. So reicht ein Aufruf der Methode, der Code muss nicht erneut implementiert werden. Die dadurch entstehende Verkürzung des Programmcodes erhöht die Übersichtlichkeit.

4.1 Einstellungen über die XML Datei

In dem Projektordner muss sich eine Datei mit dem Namen `settings_FRAtoOptimode.xml` befinden. In dieser Datei können Einstellungen für das Programm vorgenommen werden. Die Auslagerung der Einstellungen in eine XML-Datei hat den Vorteil, dass notwendige Informationen nicht bei jedem Programmstart neu eingegeben werden müssen, sondern automatisch aus der Datei gelesen und dort bei Bedarf verändert werden können. XML kann mit jedem Texteditor geöffnet und verändert werden und ist durch seine Struktur selbsterklärend. Das Dokument ist wohlgeformt und besitzt das Wurzelelement `settings`, unter dem die Elemente `api_settings` und `database_settings` angeordnet sind. Der Aufbau der Datei ist in Abbildung A2 zu sehen.

Unter `api_settings` sind die Einstellungen zur Verbindung mit der Schnittstelle des Frankfurter Flughafens zu finden. `authToken` beinhaltet den persönlichen Schlüssel zur Verbindung mit der Datenbank. Dieser wird bei jeder Abfrage gefordert und von dem Programm an dieser Stelle entnommen. Der Schlüssel kann auf der Homepage der `Fraport API` erstellt und verwaltet werden.

Die Elemente `type_of_request`, `airport`, `airlineCode`, `flightNumber` und `origin_flight_date` bestimmen, welche Funktion der Schnittstelle aufgerufen wird. Es kann eingestellt werden, ob nur An- oder Abflüge an einem Flughafen, alle Flugbewegungen eines Flughafens oder nur ein bestimmter Flug angefordert wird. Dazu wird unter `type_of_request` die Art der Abfrage angegeben, wobei *complete* für alle Flugbewegungen eines Flughafens voreingestellt ist. Der 3-Letter-Code des Flughafens ist unter dem Element `airport` anzugeben. Die Elemente `airlineCode`, `flightNumber` und `origin_flight_date` werden nur für die Abfrage eines bestimmten Fluges benötigt.

Die Einstellung `update_cycle` bestimmt die Aktualisierungsrate der Daten. Die Zahl wird als Gleitkommazahl in Minuten angegeben. Wird eine *0* eingetragen, werden die Flugdaten nur einmalig bei Aufruf des Programms aktualisiert. Über die Elemente `clear_db_ before_update` und `use_backup` kann bestimmt werden, ob die Datenbank bei Aufruf des Programms vor der ersten Aktualisierung der Daten gelöscht werden soll und ob Flugdaten aus der Sicherungsdatei geladen werden sollen, anstatt die Schnittstelle zu benutzen. Dies wird beispielsweise nötig, sollte die Schnittstelle nicht verfügbar sein.

Unter `database_settings` befinden sich die Elemente `server`, `port`, `databse`, `user` und `pass`. Sie dienen zur Verbindung mit der Datenbank, in welcher die Flugdaten gespeichert werden sollen.

Da das Programm nur von einem begrenzten Personenkreis am DLR benutzt werden soll, wurde auf eine Fehlerabfrage beim Umgang mit der XML-Datei verzichtet. Sollte die Datei nicht vorhanden oder falsch aufgebaut sein, stürzt das Programm ab. Dem Programm liegt eine Beschreibung mit dem Aufbau der XML-Datei bei, es wird somit davon ausgegangen, dass eine Absicherung in diesem Fall nicht nötig ist.

4.2 Programmstart und Einlesen der Einstellungen

Durch die Programmiersprache ist definiert, dass bei Programmstart die Methode `Main` aufgerufen wird. In dieser befindet sich der Ablauf des Programms, auch dargestellt im Anhang in Abbildung A3. Begonnen wird mit dem Einlesen der Programmeinstellungen aus der XML-Datei. Falls dort das Element `clear_db_before_update` auf *true* gesetzt ist, wird zuerst eine Verbindung zur MySQL-Datenbank aufgebaut. Der Verbindungsaufbau wurde in eine eigene Methode `GetConnectionToDatabase` ausgelagert.

In dieser Methode wird zunächst der `Connectionstring` gebildet. Diese Zeichenkette enthält alle notwendigen Informationen zum Verbindungsaufbau. Neben Adresse und

Port des Servers, enthält sie den Namen der Datenbank, Benutzernamen und Passwort zum Zugriff auf die Daten und Details zum Behandeln von Daten und Uhrzeiten als Datentyp. Die Informationen zum Verbindungsaufbau werden der XML-Einstellungsdatei entnommen. Die Daten wurden vor Methodenaufruf eingelesen und werden der Methode als Übergabewert übermittelt.

Der Befehl zum Öffnen der Datenbank befindet sich innerhalb eines `try-catch`-Blocks. Wird im `try` Bereich dieses Blocks ein Fehler ausgelöst, wird der `catch`-Block aufgerufen, anstatt das Programm abstürzen zu lassen. Dadurch können vorhersehbare Fehler im Programm abgefangen und behandelt werden.

Nach erfolgreichem Aufbau der Verbindung werden alle Tabelleneinträge mit der Szenario-ID *50* gelöscht. In der Datenbank besitzt jeder Flugeintrag diese ID. Dadurch können für verschiedene Simulationsdurchläufe unterschiedliche Datensätze benutzt werden. Durch Definition werden alle Flugbewegungen aus `FRAtoOptimode` mit der Szenario-ID *50* abgespeichert, während die bestehenden Flugdaten IDs von *10* bis *30* besitzen.

Anschließend wird geprüft, ob in der XML-Datei die Aktualisierung der Flugdaten in einem Zeitabstand oder eine einmalige Aktualisierung bei Programmstart gewünscht ist.

Ist die Aktualisierung nur einmalig gewünscht, oder sollen die Daten aus der Sicherungsdatei geladen werden, wird die Methode `UpdateFlights` aufgerufen. Bei einem gewünschten Intervall wird ein Timer mit der Methode erstellt. Ein Timer löst einen definierten Codeabschnitt immer aus, nachdem eine eingestellte Zeit abgelaufen ist. Das Intervall dieser Timerauslösung wird aus der XML-Datei entnommen.

4.3 Import der Fraport API

Ein Aktivitätsdiagramm dieser Methode befindet sich im Anhang in Abbildung A4. Sollen die Daten aus der API geladen werden, müssen zuerst die Uniform Resource Locators (URLs) für die API erstellt werden. Je nach Art der Abfrage ändert sich die Adresse des Zugriffs auf die Schnittstelle. Die URL besteht aus einer Basis-URL für alle Anfragen, an welche Details angehängt werden. Eine grafische Übersicht über die Zusammenstellung befindet sich in Abbildung 11. Die geforderten Einstellungen werden aus der XML-Datei übernommen.

Anschließend wird ein HTTP-Request an die API gestellt. Dafür stellt C# die Klasse `WebRequest` bereit. Es wird ein Objekt dieser Klasse erstellt und notwendige Informationen, wie der API-Schlüssel, das gewünschte Format und die maximale Wartezeit auf eine

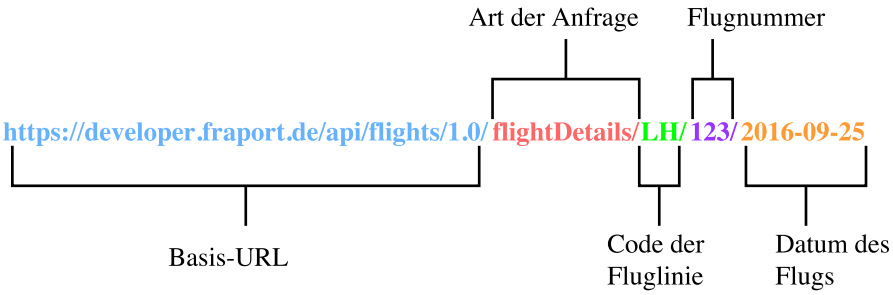


Abbildung 11: Aufbau der URL

Antwort, werden als Eigenschaften dieses Objekts definiert. Nachdem die Anfrage abgeschickt wurde, wird ihre Antwort in einem Objekt der `HttpWebResponse`-Klasse gespeichert. Der Response-Body wird zur weiteren Verarbeitung in einen string extrahiert. Die Erstellung der Abfrage und Antwort befindet sich in einer eigenen Methode `GetApiData` und ist mit einem `try-catch`-Block gegen einen Fehler im Verbindungsaufbau mit der API abgesichert. Ist in der XML-Datei das Element `use_backup` auf *true* gestellt, wird der komplette Inhalt aus der Datei `backup.txt` geladen. Diese befindet sich im selben Ordner wie das Programm und enthält einen Beispieldatensatz der An- und Abflüge von Frankfurt.

4.4 Umwandlung in das Datenbankformat

Wenn Daten aus der API empfangen oder aus der Sicherungsdatei geladen wurden, erfolgt im nächsten Schritt die Umwandlung in das Format der MySQL-Datenbank. Dazu findet zunächst eine Umwandlung des JSON-strings in einzelne Objekte statt. Die Internetseite `http://json2csharp.com/` wandelt den JSON-string automatisch in C#-Klassen um. Dabei wird über ein Textfeld die Eingabe des strings verlangt. Anschließend wird der Aufbau des JSON-Objekts als C#-Klassenstruktur ausgegeben. Der Aufbau ist in Abbildung A1 dargestellt.

Die Klassen werden durch das externe Framework `JSON.NET` automatisch befüllt. `JSON.NET` ist ein Paket zur Serialisierung und Deserialisierung von JSONstrings und über das Erweiterungsmanagementsystem `NuGet` in Visual Studio erhältlich.

Im nächsten Schritt baut die Methode `GetConnectionToDatabase` eine Verbindung zur MySQL-Datenbank auf und lädt alle Flugdaten mit der `scenario_id` *50*. Durch diesen Schritt ist dem Programm der Aufbau eines Eintrags aus der Datenbank bekannt, die Struktur muss nicht manuell in den Quelltext des Programms implementiert werden.

Jeder Flug in der MySQL-Datenbank ist in einer Zeile der Tabelle `global_flights` beschrieben. Die Tabelle besitzt 29 Spalten, die im Folgenden erklärt werden.

- **id:** Kennnummer zur Identifikation des Fluges, wird einmalig vergeben
- **callsign:** International Civil Aviation Organization (ICAO)-Code der Fluglinie und Flugnummer
- **airline_code_icao:** ICAO-Code der Fluglinie
- **direction:** Richtung des Fluges. *A* für Ankunft und *D* für Abflug
- **sibt:** Geplante In-Block Zeit. Die In-Block Zeit beschreibt den Zeitpunkt der Ankunft des Flugzeugs an der Parkposition [42]
- **tibt:** Von der Flugsicherung geplante In-Block Zeit. Kann von `sibt` abweichen
- **aibt:** Tatsächliche In-Block Zeit
- **sobt:** Geplante Off-Block Zeit. Off-Block Zeit ist der Zeitpunkt, zu welchem das Flugzeug seine Parkposition zwecks Start verlässt
- **tobt:** Von der Flugsicherung geplante Off-Block Zeit. Kann von `sobt` abweichen
- **eobt:** Aufgrund von Echtzeitdaten geschätzte Off-Block Zeit
- **atat:** Tatsächliche Off-Block Zeit
- **origin:** ICAO-Code des Startflughafens
- **destination:** ICAO-Code des Zielflughafens
- **stand:** Standplatz des Flugzeugs
- **gate_id:** Name des geplanten Gates, über das die Passagiere ein- und aussteigen können
- **target_gate_id:** Name des aktuellen Gates
- **ready_for_boarding:** Zeitpunkt, an dem das Boarding beginnen kann
- **pax:** Anzahl der gebuchten Passagiere

- **flight_type_id:** Art des Fluges, mit Ziffern codiert. Dient zur Unterscheidung von Inlands-, Kontinental- und Interkontinentalflügen sowie privaten und Linienflügen
- **schengen_flight:** Beschreibt, ob der Flug unter das Schengener Abkommen fällt
- **baggage_claim_id:** Name des Gepäckbands, an dem Passagiere ihr Gepäck empfangen können
- **aicraft_type_icao:** ICAO-Code des Flugzeugtyps
- **dep_flight_id:** Verknüpfter Abflug, wenn dieser Flug eine Ankunft ist
- **arr_fligt_id:** Verknüpfte Ankunft, wenn dieser Flug ein Abflug ist
- **lounge_id:** Lounge für Flugreisende
- **common_checkin_id:** Gruppe von Check-In-Schaltern für den Flug
- **status_id:** Codierter, aktueller Flugstatus
- **scenario_id:** Kennnummer für das Simulationsszenario des Fluges
- **checkin_counter_id:** Check-In-Schalter, die für den Flug benutzt werden können

Das Programm erstellt über einen Datenadapter ein internes Abbild der Datenbank im Format `DataTable`. Eine anschließende Schleife wird für jedes Objekt `FlightEntry`, welches einen Flug aus der API enthält, ausgeführt.

Im ersten Schritt wird eine neue, leere Zeile des Datenbankabbilds erstellt und diese mit den Daten des Objekts gefüllt. In Abbildung A5 ist abgebildet, welche Informationen der API das Programm in welche Spalte der Datenbank überträgt. Während orangene Zellen fehlende Informationen anzeigen, sind erfolgreich übertragbare Daten grün dargestellt. Die ID der Spalte wird intern ab dem Wert *500000* aufwärts gezählt, da in diesem Zahlenbereich noch keine Flugeinträge in der Datenbank vorhanden sind. Beim Übetragen in die Datenbank ändert das DBMS diesen Wert, da er fortlaufend, ähnlich einer Zeilennummerierung, hochgezählt werden muss.

Das Callsign setzt sich aus den Informationen `airlineCode`, `trackNumber` und `suffix` der API zusammen. Ob ein Flug ein An- oder Abflug ist, entscheidet das Programm, indem es den Wert `departureAirport` mit dem Flughafen aus der XML-Datei vergleicht.

Stimmen Startflughafen und Flughafen der API-Abfrage überein, wird von einem Abflug ausgegangen.

In der Datenbank ist eine Änderung des Datentyps der Spalte `gate_id` von int auf varchar notwendig, da der Flughafen Frankfurt auch Buchstaben für die Benennung der Gates verwendet. Alle Gates des fiktiven Flughafens GIA sind über Ziffern mit Einträgen in der Tabelle `global_task_station` verknüpft.

Bei dem Wert für den Flugstatus führt das Programm eine Konvertierung des Wertes durch, bevor dieser in die Datenbank übertragen wird. Die Spalte `status_id` ist durch das DBMS mit der Tabelle `global_flight_status_type` verknüpft. Dort sind die Werte *0* für pünktlich, *1* für verspätet, *2* für gestrichen, *3* für umgeleitet und *4* für undefiniert hinterlegt. Ähnliche Flugstatus übermittelt die API, jedoch sind diese mit Buchstabencodes definiert.

Obwohl die API keinen passenden Wert für die Spalte `flight_type_id` liefert, wird der Wert *1* eingetragen, da im DBMS diese Spalte mit der Beschränkung `NOT NULLABLE` hinterlegt ist. Dadurch darf die Spalte nicht leer sein, der Wert wurde nach Absprache mit dem DLR auf *1* festgelegt.

4.5 Update der Datenbank

Um einen Flug eindeutig zu identifizieren, wird die Kombination aus den Eigenschaften Flugnummer und Flugrichtung benutzt. Da einige Airlines die gleiche Flugnummer für An- und Abflug benutzen, ist eine Identifikation nur über die Flugnummer nicht möglich. Flüge müssen eindeutig identifiziert werden, um zu prüfen, ob sie bereits in der Datenbank vorhanden sind oder ob ein neuer Eintrag in der Datenbank notwendig ist. Andernfalls werden die vorhandenen Fluginformationen aktualisiert. Das Programm aktualisiert die Zeile in der Datenbank nur, wenn Änderungen bei den Fluginformationen vorliegen. Ist das nicht der Fall, wird nicht erneut auf diese Zeile in der Datenbank zugegriffen. Dadurch können überflüssige Zugriffe auf die Datenbank reduziert werden.

Nachfolgend aktualisiert die von C# implementierte Methode `Update` des Datenadapters die veränderten und neuen Zeilen der Datenbank. Diese Zeilen werden über die Eigenschaft `RowState` erkannt, die von C# automatisch bei Zugriffen auf die Datenzeilen geändert wird. Die Aktualisierung ist über einen `try-catch`-Block gegen Fehler abgesichert. Bei erfolgreichem Update zeigt die Konsole die Anzahl der aktualisierten Flüge an.

4.6 Fehlerbehandlungen

In diesem Kapitel wird auf ausgewählte Probleme und deren Lösungsweg bei der Programmierung von `FRAtoOptimode` eingegangen.

Einstellung `complete` in der XML-Datei

Die Einstellung *complete* des Elements `type_of_request` in der XML-Datei besitzt eine Besonderheit. Bei dieser Einstellung werden alle An- und Abflüge in Frankfurt geladen. Da An- und Abflüge nur einzeln von der API angefordert werden können, muss das Programm zwei Anfragen an die Schnittstelle stellen. Dadurch ändert sich der Datentyp im Quelltext. Der string mit der Antwort der API wird im Quelltext durch eine Liste von strings ersetzt. Das bietet den Vorteil, dass beliebig viele Anfragen hintereinander behandelt werden können und das Programm auch bei einer einzelnen Funktionsweise keinen Fehler auftreten lässt.

Caching bei der Anfrage an die API

Über die Einstellung `update_cycle` der XML-Datei lässt sich das Zeitintervall einstellen, in welchem das Programm die Datenbank aktualisieren soll. In der Dokumentation der Schnittstelle wird angegeben, dass die API eine Datenaktualisierungsrate von einer Minute besitzt. Bei einem `update_cycle` von einer Minute fällt jedoch auf, dass nur jede zweite Anfrage an die Schnittstelle aktualisierte Flugdaten liefert. Somit verlängert sich die eigentliche Aktualisierungsrate auf zwei Minuten. Der Grund liegt im Caching der Daten. Über das HTTP-Protokoll werden zwischengespeicherte Daten als Antwort gesendet. Nachdem Versuche erfolglos blieben, das Chaching über HTTP-Header abzuschalten, wurde die Aktualisierungsrate auf 1,1 Minuten verlängert. Nach mehreren Tests stellte sich heraus, dass bei der leicht erhöhten Rate kein Caching mehr erfolgte. Dieser Wert ist voreingestellt, in der Datei wird als Kommentar auf diese Besonderheit hingewiesen.

Zeitgleicher Zugriff auf den Datensatz

Ein bekanntes Problem bei Zugriffen auf Datenbanken ist der zeitgleiche Zugriff auf einen Datensatz. Versuchen zwei Benutzer zeitgleich einen Datensatz zu verändern, kann es zu Fehlermeldungen kommen. Da während der Entwicklung eine lokale Datenbank benutzt wurde und `FRAtoOptimode` als einziger Nutzer auf diese Zugriff hatte, musste der zeitgleiche Zugriff simuliert werden. Dazu wurde das Element `update_cycle` der XML-Datei

auf einen Wert von *0,1* gestellt. Dadurch startet das Programm den zweiten Zugriff auf die Datenbank, obwohl der erste noch nicht abgeschlossen ist. Die Folge ist eine Fehlermeldung während der Aktualisierung. Ein `try-catch`-Block um den Update-Befehl im Quelltext verhindert einen kompletten Programmabsturz und informiert den Benutzer über die Konsole sowie Log-Datei über den Fehler. Eine Aktualisierung der Datenbank wird in diesem Programmdurchlauf nicht vorgenommen.

Doppelte Einträge eines Fluges seitens der API

In einer frühen Version von `FRAtoOptimode` wurde zur eindeutigen Identifikation nur die Flugnummer benutzt. Bei genauerer Analyse der Daten ist erkennbar, dass einige Fluglinien für An- und anschließenden Abflug an einem Flughafen die identische Flugnummer verwenden. Somit war zu diesem Zeitpunkt keine eindeutige Identifikation gegeben. Als Lösung wurde die programminterne Variable `uniqueid` eingeführt. Sie setzt sich aus der Flugnummer und einem *A* für Anflüge oder einem *D* für Abflüge als Suffix zusammen.

Allerdings traten auch nach der Einführung der `uniqueid` Fehlermeldungen zu doppelten Datensätzen auf. Diese waren auf fehlerhafte Daten der API zurückzuführen. Es befanden sich in einer Anfrage an die API drei Datensätze für einen Flug mit gleicher Flugnummer, Off-Block Zeit und Registrierung. Jedoch unterschieden sich diese Daten beim Flugstatus und Zielflughafen. Da das Programm nicht in der Lage ist zu erkennen, welcher Eintrag der korrekte ist, wird immer nur der erste Datensatz eines Fluges mit gleicher `uniqueid` behandelt. Weitere Einträge aus der API ignoriert das Programm. Mehrfache Datensätze zu einem Flug waren bei Programmtests selten, führten aber vor der Problemlösung zu Programmabstürzen.

Abhängigkeiten zwischen Tabellen

Zwischen Tabellen einer Datenbank können Abhängigkeiten existieren. Dabei ist eine Spalte einer Tabelle vom Inhalt einer anderen Tabelle abhängig und nimmt nur bestimmte Werte an. Auch `global_flights` besitzt diese Abhängigkeiten. So ist beispielsweise die Spalte `status_id` an die Tabelle `global_flight_status_type` gebunden, in der die codierten Flugstatus beschrieben werden. Da die API zusätzlich zu den vorhandenen Flugstatus noch die Information über umgeleitete Flüge anbietet, wurde dieser Status der Tabelle mit dem Code *3* hinzugefügt.

Die Spalten von Gates und Gepäckbändern sind mit Tabellen verknüpft, die beispiels-

weise geografische Koordinaten dieser im fiktiven Flughafen GIA beinhalten. Die Tabelle beinhaltet jedoch keine Gates und Gepäckbänder aus Frankfurt. Beim Versuch Daten hinzuzufügen, die nicht in der verknüpften Tabelle hinterlegt sind, bricht das Programm mit einer Fehlermeldung ab. Daher wurden die Abhängigkeiten dieser Spalten aufgelöst.

Fehlende Timerauslösung nach Umwandlung in Release

Visual Studio besitzt die Möglichkeit, das Programm in zwei verschiedenen Modi, Debug oder Release, zu erstellen. Der Debugmodus wird während der Programmierung verwendet und bietet alle Möglichkeiten, Fehler im Quelltext zu entdecken. Der Releasemodus belegt hingegen weniger Speicher bei der Ausführung und ist für die fertige Version des Programms gedacht [43]. Während im Debugmodus jedoch die Intervallaktualisierung funktionierte, zeigte das Programm im Releasemodus nach der gewünschten Aktualisierungszeit keine Reaktion. Grund für dieses Verhalten war der Garbage Collector.

Der Garbage Collector ist ein Tool von Visual Studio, dass nach definierten Ereignissen den Speicherverbrauch des Programms analysiert und nicht länger benötigte Speicherbereiche für andere Programme freigibt [44]. Der Timer zur Intervallaktualisierung befindet sich in einem eigenen Thread. Ein Thread beschreibt eine Ausführungseinheit eines Programms. Er kann als Ablauflinie von Quelltext betrachtet werden. Besitzt ein System die Eigenschaft des Multithreadings, können mehrere Threads parallel ausgeführt werden. Standardmäßig besitzt ein C#-Programm einen Thread, die `Main`-Methode [40]. `FRAtoOptimode` benutzt für den Timer einen zweiten, parallelen Arbeitsthread. Dadurch können Benutzereingaben angenommen werden, während der Timer wartet, bis eine erneute Aktualisierung durchgeführt werden soll.

Im Releasemodus besitzt der Garbage Collector eine andere Verhaltensweise als im Debugmodus und gibt den Speicher mit dem Timerthread frei, da dieser für den Garbage Collector scheinbar nicht mehr vom Programm benötigt wird. Dadurch erfolgt nur eine einmalige Aktualisierung bei Programmstart. Um das Problem zu beheben, wurde die Art der Implementierung des Timers verändert. Die genaue Definition, in welchem Abstand der Timer auslösen soll, wird dem Thread nicht mehr bei Erstellung des Timers mitgeteilt. Das Auslöseintervall wird direkt nach Erstellung zugewiesen. Dadurch wird außerhalb des Threads auf den Timer zugegriffen und der Garbage Collector behält den Thread im Speicher und gibt ihn nicht mehr frei.

5 Fazit

Eine Simulation ist immer nur so gut wie die Daten, auf denen sie basiert [45, S.63]. In der Vergangenheit arbeitete das DLR mit fiktiven Verkehrsdaten innerhalb seiner Simulationsumgebung `Optimode.net`. Die programmierte Software `FRAtoOptimode` erfüllt das Ziel der Arbeit, reale Echtzeitdaten in die Datenbank des Leitstands einzuführen. Diese können benutzt werden, um die Qualität zukünftiger Simulationsdurchläufe zu verbessern. Auch bei kurz- und langfristiger Abwesenheit der Datenschnittstelle stellt das Programm Lösungen zur Verfügung. Bei kurzfristigen Ausfällen stoppt `FRAtoOptimode` die Aktualisierung des Datensatzes, bei langfristiger Unerreichbarkeit steht ein Sicherungsdatensatz zur Verfügung. Dadurch geht die Eigenschaft der Echtzeit verloren, die Qualität der Originaldaten eines Flughafens bleibt jedoch gegenüber dem fiktiven Datensatz verbessert. Die Dokumentation des Programms erleichtert nachfolgende Wartungsarbeiten an der Software und bietet Optionen für Erweiterungen.

5.1 Ausblick `Optimode.net`: Erstellung einer eigenen API

Eine mögliche zukünftige Funktion innerhalb von `Optimode.net` ist eine API, die Daten der Passagiere enthält. In dieser Schnittstelle stellen mobile Endgeräte der Reisenden beispielsweise Positionsdaten in Echtzeit zur Verfügung. Diese werden in einer Datenbank gesammelt und über eine RESTful API ausgewählten Nutzern angeboten. Dadurch kann die Simulation genauere Prognosen erarbeiten, da die Position des Reisenden durchgehend bekannt ist und nicht nur an zentralen Punkten der Reisekette erfasst wird.

Durch die Schnittstelle könnte auch Reisedienstleistern Zugriff auf die Daten gewährt werden. Dies wäre für Fluglinien eine Möglichkeit einzusehen, wie lange ihre Passagiere noch zum Gate brauchen. Zusätzlich ist die Steuerung von personalisierten Angeboten aufgrund der Position eines Reisenden denkbar.

Eine App auf dem mobilen Endgerät des Passagiers kann die Aufgabe der Datenbereitstellung übernehmen. Dabei wird die momentane Webschnittstelle als Grundfunktion zur Übersicht der Reisekette für den Passagier implementiert. Verspätungen und Änderungen der Reisekette durch den interaktiven Leitstand werden als Pushbenachrichtigungen in Echtzeit an den Nutzer mitgeteilt. Ergibt sich eine Änderung in der Reisekette, sind aber mehrere, nahezu gleichwertige Alternativwege möglich, kann der Leitstand dem Passagier über die App eine Möglichkeit zur Mitbestimmung des weiteren Weges in Abhängigkeit

von persönlichen Präferenzen bieten. Bei der Programmierung der App kann sich an den bestehenden Programmen aus Kapitel 2.2 orientiert werden.

5.2 Ausblick von FRAtoOptimode

Bis `FRAtoOptimode` in vollem Umfang in der Simulation eingesetzt werden kann, muss die selektive Deaktivierung der Aktualisierung einzelner Datensätze implementiert werden. Sobald im interaktiven Leitstand von `Optimode.net` Änderungen vorgenommen werden, müssen diese dauerhaft gespeichert bleiben und dürfen nicht durch das Programm im nächsten Aktualisierungsschritt überschrieben werden. Dafür bietet sich die Integration einer weiteren Spalte in der Datenbank an, die angibt, ob an dem Datensatz Änderungen durch den Leitstand erfolgt sind.

Die Implementierung von Flugdaten aus der API des Frankfurter Flughafens in die Datenbank von `Optimode.net` konnte erfolgreich abgeschlossen werden. Dadurch steht dem DLR eine neue Datenquelle für zukünftige Versuche zur Verfügung, die auf realen Echtzeitdaten basiert. Der objektorientierte Aufbau des Quelltextes erlaubt einfache Integration weiterer APIs auf Basis von `FRAtoOptimode`. Das Programm könnte somit eine Grundlage zur Benutzung weiterer Echtzeitdaten in der Simulationsumgebung bieten. Dazu steht die `Deutsche Bahn` in besonderem Fokus. Sollte sie ihre Strategie von Open Data weiterverfolgen und Echtzeitdaten von Zugverbindungen als API anbieten, wäre an dieser Stelle eine weitere, interessante Datenquelle für `Optimode.net` vorhanden.

A Anhang

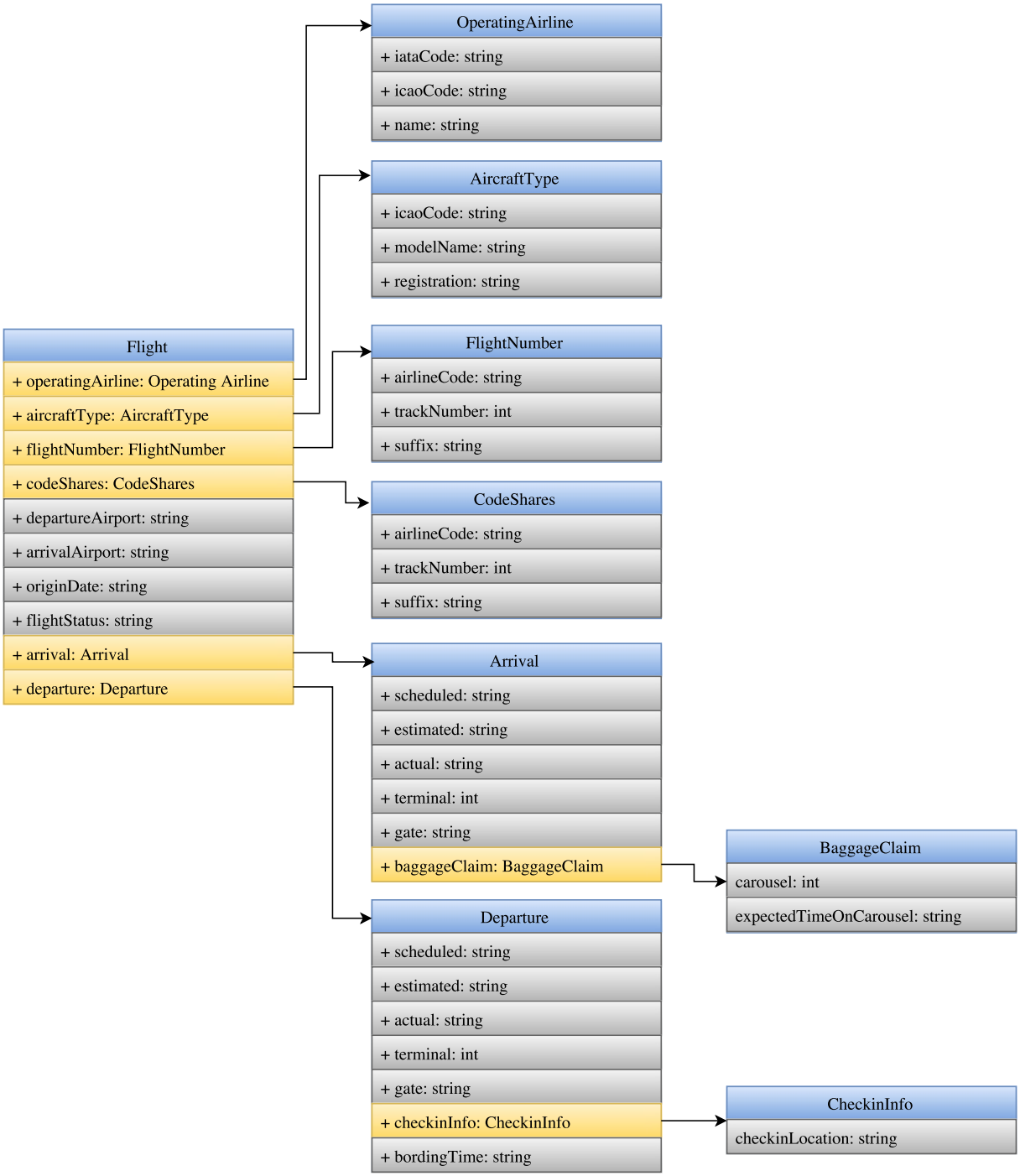


Abbildung A1: Klassendiagramm: Flight

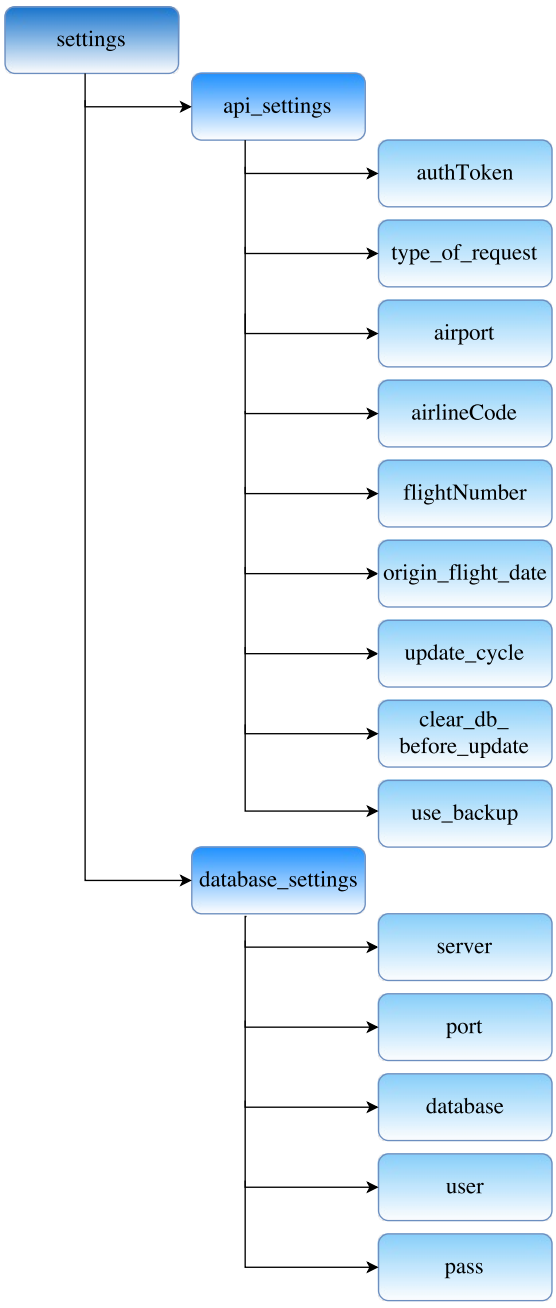


Abbildung A2: Aufbau der XML-Datei

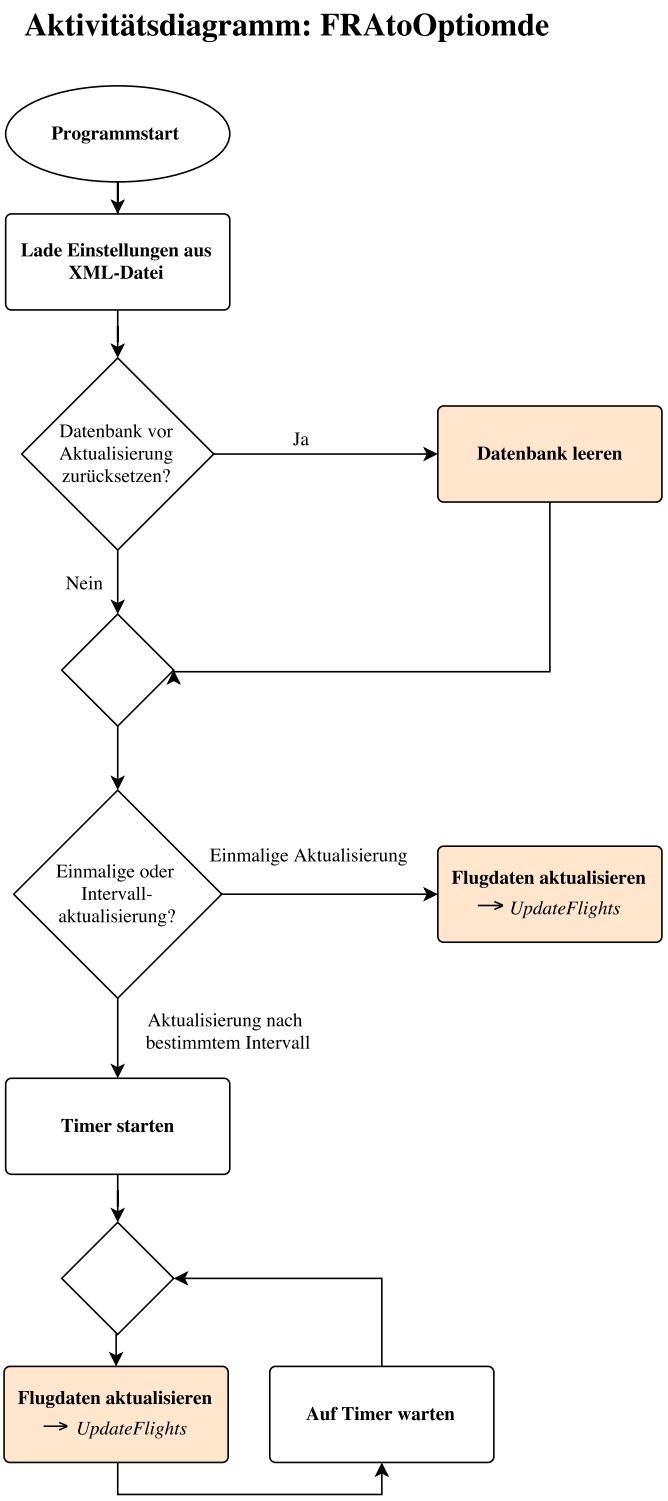


Abbildung A3: Aktivitätsdiagramm: FRAtoOptimode

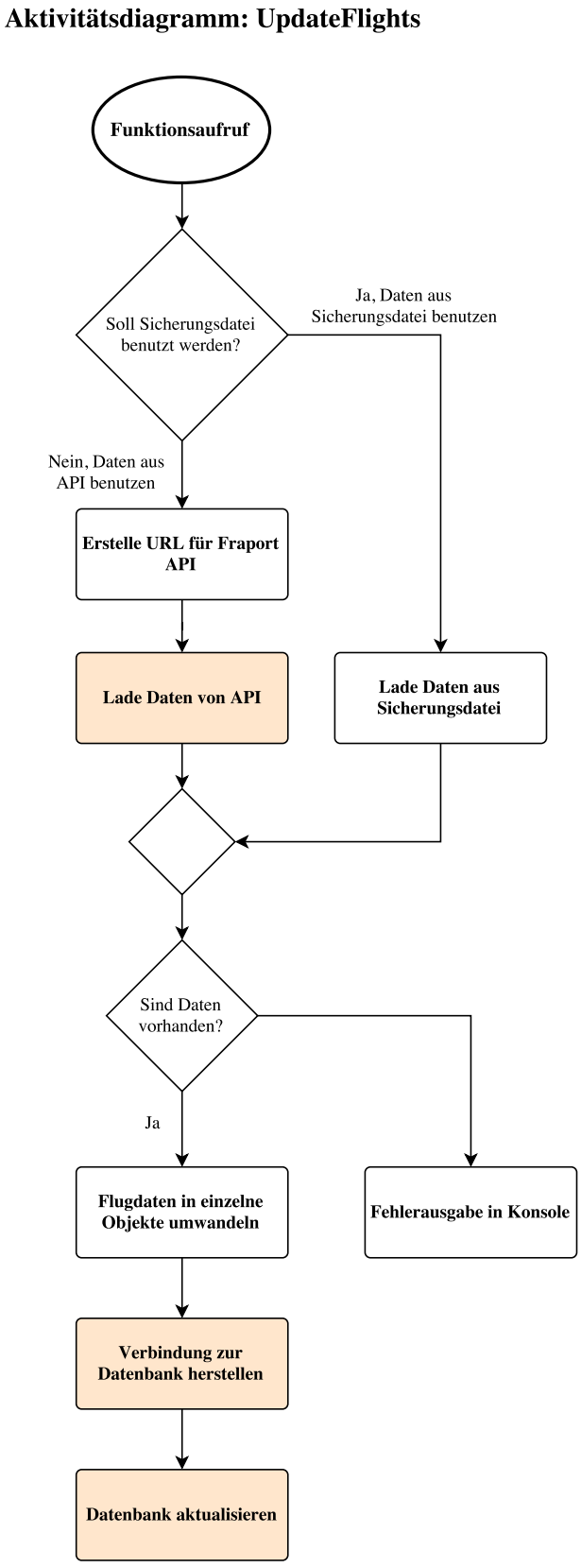


Abbildung A4: Aktivitätsdiagramm der Methode: UpdateFlights

SQL Datenbank	Abflüge	Ankünfte	Notizen
id			Ab 500000 aufwärts
callsign	flightNumber.airlineCode + flightNumber.trackNumber + flightNumber.suffix	flightNumber.airlineCode + flightNumber.trackNumber + flightNumber.suffix	
airline_icao_code	operatingAirline.icaoCode	operatingAirline.icaoCode	
direction	A	D	
sibt	arrival.scheduled		
tibt	arrival.estimated		
aibt	arrival.actual		
sobt		departure.scheduled	
tobt			
eobt		departure.estimated	
atat		departure.actual	
origin	departureAirport	departureAirport	
destination	arrivalAirport	arrivalAirport	
stand			
gate_id	arrival.gate	departure.gate	Typenumwandlung in Datenbank von int auf varchar notwendig
target_gate_id	arrival.gate	departure.gate	
ready_for_boarding		departure.boardingTime	
pax			
flight_type_id	1	1	NOT NULLABLE --> 1
schengen_flight			
baggage_claim_id	arrival.baggageClaim.carousel		
aircraft_type_icao	aircraftType.icaoCode	aircraftType.icaoCode	
dep_flight_id			
arr_flight_id			
lounge_id			
common_checkin_id		departure.checkinInfo.checkinLocation	
status_id	flightStatus	flightStatus	Umrechnung notwendig
scenario_id	50	50	
checkin_counter_id		departure.checkinInfo.checkinLocation	

Abbildung A5: Diese Tabelle zeigt, welche Daten der API in die Datenbank importiert werden. In der linken Spalte sind die Spalten der Datenbank. Daneben befinden sich Spalten für Ankünfte und Abflüge. Grüne Zellen zeigen an, dass die Information von der API in die Datenbank übertragen werden konnte, orangefarbene Zellen weisen auf fehlende Informationen hin.

Abbildungsverzeichnis

Literatur

[1] *Adding the App Bar.* http://developer.android.com/training/appbar/index.html, Stand: 29.08.2016

[2] Ecma International: *Standard ECMA-404. The JSON Data Interchange Format.* http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, Oktober 2013, Stand: 01.09.2016

[3] Duden: *Bedeutungsübersicht Link.* http://www.duden.de/rechtschreibung/Link, Stand: 24.08.2016

[4] Duden: *Bedeutungsübersicht scrollen.* http://www.duden.de/rechtschreibung/scrollen, Stand: 24.08.2016

[5] Edmund Stoiber: *Rede über das Projekt Transrapid. Neujahrs-Empfang der CSU München.* 21.01.2002

[6] ThyssenKrupp Transrapid GmbH: *Kompetenz für den Transrapid.* http://www.transrapid.de/pdf/Kompetenz%20TR-zweispr.pdf, Stand: 12.08.2016

[7] Anne Seith, Spiegel Online: *Aus für Prestigeprojekt: Transrapid in Münschen wird nicht gebaut.* http://www.spiegel.de/wirtschaft/aus-fuer-prestigeprojekt-transrapid-in-muenchen-wird-nicht-gebaut-a-543688.html, Stand: 12.08.2016

[8] Flughafen München: *Ergebnisse der Fluggastbefragung 2015.* http://www.munich-airport.de/de/company/facts/verkehr/pass_struk/index.jsp, Stand: 12.08.2016

[9] Flughafen München: *Ihr Weg zum Flughafen. An- und Abfahrt mit der Bahn.* http://www.munich-airport.de/de/consumer/anab/bahn1/index.jsp, Stand: 12.08.2016

[10] Europäische Kommission ; Generaldirektion für Forschung und Innovation ; Generaldirektion für Mobilität und Verkehr: Flightpath 2050. Europe's vision for aviation : maintaining global leadership and serving society's needs. http://dx.doi.org/10.2777/50266. – DOI 10.2777/50266

[11] Ruhren, S. Von d. ; Rindfüser, G. ; Beckmann, K.J. ; Kuhimhof, T. ; Chlond, B. ; Zumkeller, D.: *Schlussbericht FE-Nr. 70.724/2003. Forschungsprogramm zur Verbesserung der Verkehrsverhältnisse in den Gemeinden.* 2003

[12] Treiber, M. ; Kesting, A.: *Verkehrsdynamik und -simulation.* Springer-Verlag Berlin. http://dx.doi.org/10.1007/978-3-642-05228-6_2. http://dx.doi.org/10.1007/978-3-642-05228-6_2

[13] DLR: *Screenshot Webansicht von **Optimode.net**.* http://optimode.net/tr/775-LH0098_10990.html, Stand: 19.09.2016

[14] Kantar Worldpanel: *Smartphone OS sales market share.* http://www.kantarworldpanel.com/global/smartphone-os-market-share/, Stand: 10.08.2016

[15] Google: *Google Play. Übersichtsseite Android Apps.* https://play.google.com/intl/fr_de/about/apps/, Stand: 10.08.2016

[16] Fraport AG - Frankfurt Airport Services Worldwide: *App: Frankfurt Airport.* https://play.google.com/store/apps/details?id=com.infsoft.android.fraapp, Stand: 30.08.2016

[17] Fraport AG: *Screenshot der Android App **Frankfurt Airport**.* Stand: 19.09.2016

[18] Flughafen München GmbH: *Screenshot der Android App **MUC Airport, Flughafen München**.* Stand: 19.09.2016

[19] Deutsche Lufthansa AG: *Screenshot der Android App **Lufthansa**.* Stand: 19.09.2016

[20] Deutsche Bahn AG: *Zurück in die Erfolgsspur. Wettbewerbsbericht 2016.* https://www.deutschebahn.com/file/de/9801886/QK_j0jFiPywodLAw5ZLvmirXuak/11698752/data/160712_wettbewerb.pdf, Mai 2016, Stand: 28.07.2016

[21] Deutsche Bahn: *Screenshot der Android App **DB Navigator**.* Stand: 19.09.2016

[22] Verband Deutscher Verkehrsunternehmen: *Personenverkehr. Rund 10 Milliarden Fahrgäste jährlich.* https://www.vdv.de/personenverkehr.aspx, Stand: 17.08.2016

[23] Andreas Schildbach: *Screenshot der Android App Öffi - Fahrplanauskunft*. Stand: 19.09.2016

[24] Spiegel Online: *Fahrpläne: Google Maps zeigt Nahverkehr in Echtzeit an.* http://www.spiegel.de/netzwelt/web/google-maps-fahrplaene-des-oepnv-in-echtzeit-a-1036914.html, Stand: 19.08.2016

[25] Hessenschau: *Busse und Bahnen. RMV verweigert Google Maps Fahrplandaten. 21.01.16.* http://hessenschau.de/wirtschaft/rmv-will-google-maps-nicht-fuettern,verkehrsverbuende-googlemaps-100.html, Stand: 19.08.2016

[26] ARD. Ratgeber Auto-Reise-Verkehr: *Reiseerleichterung - die neue Onlineplattform Qixxit.* http://www.daserste.de/information/ratgeber-service/auto-reise-verkehr/sendung/sr/20072014-qixxit-104.html, Stand: 19.08.2016

[27] Deutsche Bahn AG: *Open Data bei der DB.* http://data.deutschebahn.com/blog/willkommen-bei-dbopendata, Stand: 23.08.2016

[28] Verkehrsverbund Berlin-Brandenburg: *API-Schnittstelle für Webentwickler.* http://www.vbb.de/de/article/fahrplan/webservices/schnittstellen-fuer-webentwickler/5070.html, Stand: 23.08.2016

[29] Hamburger Verkehrsverbund: *Datenschnittstelle für Entwickler.* http://www.hvv.de/fahrplaene/fahrplanauskunft/datenschnittstelle/, Stand: 23.08.2016

[30] ProgrammableWeb: *APIs by Category.* http://www.programmableweb.com/category-api, Stand: 23.08.2016

[31] Google: *Google Maps APIs. Preise und Nutzungsmodelle.* https://developers.google.com/maps/pricing-and-plans/?hl=de#sup_1, Stand: 23.08.2016

[32] Chow, Shu-Wai: *Web 2.0. Webseiten intelligent verknüpfen.* Franzis Verlag GmbH. – ISBN 978–3–7723–7905–5

[33] Alby, Tom: *Web 2.0. Konzepte, Anwendungen, Technologien.* 3. Auflage. Carl Hanser Verlag. – ISBN 978–3–446–41449–5

[34] Fielding, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures.* https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf, Stand: 31.08.2016

[35] Richardson, Leonard ; Ruby, Sam: *Web Services mit REST.* 1. Auflage. O'Reilly Verlag. – ISBN 978–3–89721–727–0

[36] Reschke, J. ; Fielding, R.: *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing.* https://tools.ietf.org/html/rfc7230, Stand: 31.08.2016

[37] Kersken, Sascha: *Kompendium der Informationstechnik, 2003.* http://openbook.rheinwerk-verlag.de/kit/itkomp15000.htm#Rxx355kap15000040003C01F04F1FA, Stand: 12.08.2016

[38] Bray, Tim ; Paoli, Jean ; Sperberg-McQueen, C. M. ; Maler, Eve ; Yergeau, Francois: *Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation 26 November 2008.* http://www.w3.org/TR/xml/, Stand: 03.08.2016

[39] Heise: *Entwicklungsumgebung.* http://www.heise.de/download/products/programmierung/entwicklungsumgebung/#?cat=programmierung%2Fentwicklungsumgebung, Stand: 06.09.2016

[40] Kühnel, Andreas: *Visaul C# 2012.* 6. aktualisierte und erweiterte Auflage. http://openbook.rheinwerk-verlag.de/visual_csharp_2012/1997_03_001.html#dodtpf766d18e-5ded-46aa-8e8e-625d86cb0d78, Stand: 07.09.2016

[41] Schubert, Matthias: *Datenbanken. Theorie, Entwurf und Programmierung relationaler Datenbanken.* 2. Auflage. B. G. Teubner Verlag. – ISBN 978–3–8351–0163–0

[42] EUROCONTROL: *ATM Lexicon. Scheduled In-Block Time.* http://www.eurocontrol.int/lexicon/lexicon/en/index.php/Scheduled_In-Block_Time, Stand: 13.09.2016

[43] Microsoft: *Gewusst wie: Festlegen von Debug- und Releasekonfigurationen.* https://msdn.microsoft.com/de-de/library/wx0123s5.aspx, Stand: 14.09.2016

[44] Microsoft: *Garbage Collection.* https://msdn.microsoft.com/de-de/library/0xy59wtx%28v=vs.110%29.aspx, Stand: 14.09.2016

[45] Rabe, Markus ; Spieckermann, Sven ; Wenzel, Sigrid: *Verifikation und Validierung für die Simulation in Produktion und Logistik*. Springer-Verlag Berlin. http://dx.doi.org/10.1007/978-3-540-35282-2. http://dx.doi.org/10.1007/978-3-540-35282-2