Ermittlung von Wartezeitenprognosen mit Hilfe einer Simulation und deren Darstellung anhand einer mobilen Anwendung

Bachelor-Thesis

zur Erlangung des akademischen Grades B.Sc.

Florian Schädler

2077925

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

Erstprüfer: Prof. Dr. Andreas Plaß

Zweitprüfer: Dipl.-Ing. Carsten Lill

Hamburg, 8. 2. 2016

Inhaltsverzeichnis

Abstract

This thesis deals with the calculation of waiting times at the security check of Hamburg Airport and their display on an mobile application. The aim was the conception and implementation of a solution for calculating actual and future waiting times, as well as the development of an mobile application, which displays these calculated values. This study documents the development process. Initially, the requirements and tasks are clarified based on the available data (chapter 2). After that, the programming language gets defined for every task (chapter 3). Chapter 4 deals with the implementation of every single task. This includes the conception and implementation of the simulator (chapter 4.2), the prediction of the waiting times (4.3), the installation of the developed solution on a server (chapter 4.4) and the development of the mobile application (chapter 4.5). The conclusion summarizes and evaluates the results of this study. It states, that within the framework of this study a solid solution for calculating and display waiting times at security lanes was developed.

Zusammenfassung

Diese Arbeit befasst sich mit der Berechnung von Wartezeiten an der Sicherheitskontrolle des Hamburger Flughafens und deren Darstellung anhand einer mobilen Anwendung. Das Ziel des Projektes war die Konzeption und Implementierung einer Lösung für die Berechnung von aktuellen und zukünftigen Wartezeiten sowie die Entwicklung einer mobilen Anwendung als Anzeige dieser berechneten Daten. Die vorliegende Arbeit dokumentiert diesen Entwicklungsprozess. Zunächst werden aufgrund der zur Verfügung stehenden Daten die Anforderungen und zu bearbeitenden Aufgaben geklärt (Kapitel 2). Daraufhin wird die Programmiersprache für die einzelnen Aufgabengebiete des Projektes festgelegt (Kapitel 3). Kapitel 4 behandelt in vier Abschnitten die Umsetzung der einzelnen Aufgaben. Das umfasst die Konzeption und Implementierung des Simulators (Kapitel 4.2), die Vorhersage für die Wartezeiten (Kapitel 4.3), die Installation der entwickelten Lösung in einer Serverumgebung (Kapitel 4.4) und die Entwicklung der mobilen Anwendung (Kapitel 4.5). Im Fazit wird das Ergebnis dieser Arbeit zusammengefasst und bewertet. Es wird festgestellt, dass in dieser Arbeit eine solide Lösung für die Berechnung der Wartezeiten an Sicherheitskontrollen und dessen Anzeige entwickelt wurde.

1 Einleitung

Cognizant Technology Solutions ist eine IT-Dienstleistungsfirma mit Hauptsitz in Teaneck, New Jersey. Der Vorläufer der Firma wurde 1994 gegründet, zunächst jedoch unter dem Namen Dun & Bradstreet. Nach einer umfangreichen Umstrukturierung im Jahr 1996 wurde die Firma in Cognizant Corp. umbenannt. Die nächste Umstrukturierung fand im Jahr 2002 statt und gründete somit das bis heute existente Cognizant Technology Solutions. Die Firma erstreckt sich über eine große Anzahl an Geschäftsbereichen. Zu diesen gehört auch der Bereich Business Intelligence, der sich unter anderem mit der systematischen Analyse von großen Datenmengen befasst.

In diesem Bereich arbeite ich seit Dezember 2014 als Entwickler für mobile Anwendungen mit Spezialisierung auf das Android Betriebssystem. Der Flughafen Hamburg ist einer der Kunden von Cognizant. Für diesen plane und entwickle ich zurzeit in einem Team Anwendungen, welche die Protokollierung der Arbeitsschritte für die Bodenverkehrsdienste erleichtern sollen. Da ich bereits in Kontakt mit diesem Kunden stehe, hat sich der Flughafen als Plattform für meine Bachelorarbeit angeboten.

Eines der momentan existenten Probleme am Hamburger Flughafen ist die Angabe von Wartezeiten bei der Sicherheitskontrolle. In diesem Bereich gibt es weltweit unterschiedliche Lösungsansätze, welche auf unterschiedlichen Technologien basieren. Am Hamburger Flughafen wird derzeit ebenfalls nach einer Lösung für dieses Problem gesucht, jedoch sind die Fortschritte in diesem Gebiet überschaubar. Aus diesem Grund hat sich dieses Aufgabengebiet zum Thema meiner Forschungsarbeit entwickelt.

1.1 Zielsetzung

Bis zum Start des Projektes gab es am Hamburger Flughafen nur sehr wenige Fortschritte im Bereich der Wartezeitenermittlung. Darum gibt es keine Referenzen oder Vorarbeiten auf denen die Untersuchungen aufbauen. Lediglich für die Aufnahme und Bereitstellung von Daten wurden bereits Vorrichtungen installiert. Die Anordnung für die Erfassung der Daten kann der Skizze in der Abbildung 1.1 entnommen werden.

Diese Skizze spiegelt nicht exakt das Warteschlangensystem des Hamburger Flughafens wieder, sondern wird lediglich zu Erklärungszwecken verwendet. Es besteht aus 24 Kontrollspuren (Nummerierung 1 - 24). Eine Kontrollspur ist der Bereich, in

dem die Sicherheitskontrolle durchgeführt wird. Diese Kontrollspuren können entweder geöffnet (Grün) oder geschlossen (Rot) sein. Vor den Kontrollspuren reihen sich die Passagiere in die Warteschlange ein (schwarzer Kreis). An zwei der 24 Kontrollspuren sowie an den Eingängen sind Ticketscanner installiert (gelbe Raute).

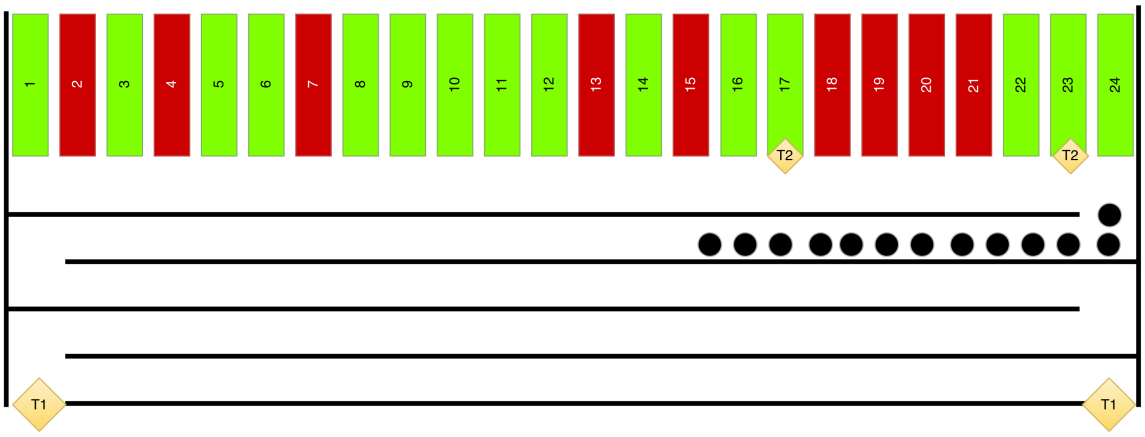

Abbildung 1.1: Skizze der Warteschlange des Hamburger Flughafens

Um die Daten zu ermitteln, welche sich für die Berechnung der Wartezeit verwenden lassen, wurden diese zwei Ticketscanner an den Kontrollspuren installiert. Wird ein Ticketscanner benutzt, so speichert dieser die Ticketnummer, den Zeitpunkt des Scannens sowie die Kennung des Scanners ab. In der Abbildung 1.1 sind diese Ticketscanner in Form einer Raute mit der Kennung *T2* exemplarisch markiert. Der Passagier passiert diese Scanner sobald er die Kontrollspur betritt. Die resultierenden Daten der T2-Ticketscanner werden in dieser Arbeit als *zweiter Zeitstempel* bezeichnet.

Des Weiteren stehen die Daten der Ticketscanner zur Verfügung, welche sich beim Eingang der Warteschlange befinden. In der Abbildung 1.1 sind diese ebenfalls exemplarisch in Form einer Raute mit der Kennung *T1* markiert. Die resultierenden Daten der T1-Ticketscanner werden im Folgenden als *erster Zeitstempel* betitelt.

Das Hauptziel des Projektes ist es, basierend auf den zur Verfügung stehenden Daten, einen Lösungsweg zu entwickeln, welcher eine möglichst präzise Aussage über die aktuelle Wartezeit bei der Sicherheitskontrolle trifft. Im weiteren Verlauf dieser Arbeit wird diese Wartezeit als *IST-Wartezeit* beschrieben.

Zum Projektstart waren die T2-Ticketscanner noch nicht installiert, sodass auf diese Daten nicht zugegriffen werden konnte. Da die Daten für die Ermittlung der IST-Wartezeit unerlässlich sind, mussten diese mithilfe einer Simulation generiert werden.

Ein weiterer Teil des Projektes besteht darin, eine Prognose über die voraussichtliche Wartezeit für einen beliebigen Zeitpunkt im Zeitraum von bis zu einer Woche zu treffen. Diese Wartezeit wird im weiteren Verlauf als *WIRD-Wartezeit* betitelt. Für die Berechnung der WIRD-Wartezeit stehen vom Hamburger Flughafen getätigte Prognosen in Form von Passagieraufkommen zur Verfügung. Das Passagieraufkommen wird in Anzahl der erwarteten Passagiere angegeben.

Die beiden errechneten Wartezeiten sollen anhand einer mobilen Anwendung dargestellt werden. Zu diesem Zweck müssen die Ergebnisse der Berechnungen auf einem Server bereitgestellt werden, damit die Anwendung auf diese Daten zugreifen kann. Die Anwendung soll auf unterschiedlichen mobilen Betriebssystemen ausführbar sein.

Aufgrund der hier beschriebenen Zielsetzung ergeben sich fünf zu bearbeitende Aufgabenfelder

1. Konzeption und Entwicklung einer Simulation zur Generierung von Zeitstempeln
2. Konzeption und Entwicklung eines Verfahrens zur Ermittlung der IST-Wartezeit
3. Konzeption und Entwicklung eines Verfahrens zur Berechnung der WIRD-Wartezeit, basierend auf den gegebenen Prognosen
4. Installation der implementierten Lösungen in einer Serverumgebung und Bereitstellung der Ergebnisse
5. Konzeption und Entwicklung einer mobilen Anwendung zum Zwecke der Darstellung der ermittelten Werte

1.2 Struktur der Arbeit

Die Durchführung des Projektes auf Basis aller Anforderungen und Zielsetzungen hat sechs Monate gedauert. In dieser Zeit habe ich beim Flughafen Hamburg gearbeitet. Die Resultate meiner Untersuchungen gliedern sich wie folgt auf:

- Ein Konzept für die Ermittlung von Wartezeiten sowie dessen Implementierung
- Eine Demonstration der ermittelten Werte mithilfe einer mobilen Anwendung

Diese Arbeit begleitet einen praktischen Arbeitsprozess und ist dementsprechend durchstrukturiert. In chronologischen Schritten werden hier die Arbeitsprozesse beschrieben. Zunächst wird die Analyse durchgeführt. Diese behandelt den derzeitigen Istzustand am Hamburger Flughafen. Des Weiteren umfasst dieser Schritt die Eruierung der verfügbaren Daten und die daraus resultierenden Aufgaben und Anforderungen. Als Folgeschritt wird die zu verwendende Programmiersprache ausgewählt. Hiernach folgt die Konzeption und Implementierung der definierten Aufgaben. Dieser

Schritt umfasst die Entwicklung des Simulators sowie die Entwicklung der Lösungsvorschläge für die IST- und WIRD-Wartezeit. Dazu gehört auch das Installieren der Lösungen auf einer Serverumgebung sowie die Entwicklung der mobilen Anwendung. Im Anschluss wird ein abschließendes Fazit vorgestellt. In dieser Arbeit wird nicht die gesamte Umsetzung des Projektes im Detail beschrieben, es werden lediglich grundlegende Elemente für die Funktion der einzelnen Aufgaben dieses Projekts beschrieben. Der Übersicht wegen werden die Klassen und Codebeispiele gekürzt dargestellt. Der Fokus dieser Arbeit liegt auf der Konzeption und Implementierung des Simulators und des Algorithmus für die Berechnung der IST-Wartezeit. Die Arbeit wird von Diagrammen, Tabellen und Bildern begleitet, welche zur Unterstützung von Beschreibungen und Erklärungen herangezogen werden.

2 Analyse

Dieses Kapitel behandelt die Untersuchung des Istzustandes in Bezug auf die Berechnung der Wartezeiten. Darüber hinaus werden die zur Verfügung stehenden Daten eruiert. Abschließend werden die aus diesen Erkenntnissen resultierenden Aufgaben und Anforderungen definiert.

2.1 Istzustand

Wie bereits erwähnt, gab es bis zum Start des Projektes nur wenig Fortschritte von Seiten des Hamburger Flughafens zu der Berechnung der IST-Wartezeit. Nicht nur beim Flughafen Hamburg, sondern auch bei weiteren Flughäfen besteht zeitweise das Problem langer Warteschlangen und eine zuverlässige Berechnung der gegenwärtigen Wartezeiten. Einige[1] Flughäfen lösen das Problem mit einem Verfahren der Firma XOVIS, welches mittels Kamerabildern eine Aussage über die Wartezeit tätigen kann. Diese Lösung erfordert eine höhere finanzielle Investition. Des Weiteren gibt es in Deutschland, speziell bei der Sicherheitskontrolle, Bereiche, die nicht von Kameras abgelichtet werden dürfen. Da die Sicherheitskontrolle in das Aufgabengebiet der Bundespolizei fällt, liegt die Freigabe zum Filmen dieser Bereiche nicht allein in der Autorität des Flughafens.

Weitere Möglichkeiten, die im Zusammenhang mit der Berechnung der IST-Wartezeit stehen, konnten im Rahmen dieser Recherche nicht ermittelt werden. Aus diesem Grund sind keine Referenzen oder Lösungsansätze vorhanden, auf die im Laufe dieses Projektes zur Berechnung der IST-Wartezeiten zugegriffen werden kann.

Für die Berechnung der WIRD-Wartezeit stellt der Flughafen Hamburg Prognosen in Form von Passagieraufkommen zur Verfügung. Diese Prognosen erstellt der Flughafen unabhängig von diesem Projekt, daher gehört die Ermittlung der Prognosen nicht zum Projektumfang. Mittels einer Formel kann anhand dieser Prognosen die WIRD-Wartezeit berechnet werden. Dazu wird das Passagieraufkommen mit einer Servicerate verrechnet. Die Servicerate wird in Zeit pro Passagier angegeben und basiert auf der Anzahl der geöffneten Kontrollspuren sowie deren Geschwindigkeit.

Eine Anwendung, welche ausschließlich das Ziel hat, Wartezeiten und die Prognose von Wartezeiten anzuzeigen, konnte im Rahmen dieser Recherche nicht ermittelt

[1] (Xovis 2016)

werden. Deshalb gilt auch hier, dass auf keine Lösungen bezüglich der Strukturierung der Benutzeroberfläche zurückgegriffen werden kann. Für die Programmstruktur der Anwendung gibt es jedoch Vorgaben vom Hamburger Flughafen.

2.2 Vorhandene Daten

Die Berechnung der IST-Wartezeit basiert auf den Daten, welche die Ticketscanner ermitteln, also der erste und zweite Zeitstempel, die Ticketnummer sowie die Kennung des benutzten Scanners. Die Herkunft dieser Zeitstempel wurde bereits im Kapitel 1.1 erklärt und kann der Abbildung 1.1 entnommen werden.

Jeder Passagier, der durch die Sicherheitskontrolle muss, hat vor dem Einordnen in die Warteschlange sein Ticket zu scannen. Somit ergibt sich eine nahezu hundertprozentige Ausbeute bei den T1-Ticketscannern (erster Zeitstempel). Die Ausbeute basiert auf der Anzahl aller Passagiere, welche die T2-Ticketscanner passiert haben und gibt den prozentualen Anteil derer an, welche ihr Ticket freiwillig gescannt haben. Da es jedoch Passagiere mit Sondertickets gibt, die keinen erfassbaren Scan-Code aufweisen, kann es keine hundertprozentige Erfassung geben.
Das Scannen der Tickets bei den T2-Ticketscannern (zweiter Zeitstempel) ist optional. Da diese Scanner erst im Laufe des Projektes in Betrieb genommen wurden, besteht keine Kenntnis über die Ausbeute dieser Datenquelle. Aufgrund dessen kann man bei der Ausbeute des zweiten Zeitstempels nur von einem geschätzten Wert ausgehen. In diesem Projekt wird dieser Wert auf 30 % geschätzt. Der Wert bezieht sich also nur auf die Anzahl der Passagiere, welche einen der T2-Scanner auch tatsächlich passieren. Nicht alle gehen an diesen Scannern vorbei, da sie lediglich bei zwei Kontrollspuren installiert wurden (siehe Abb. 1.1). Dieser geschätzte Wert basiert auf keinerlei statistischer Grundlage. Er wurde während einer Besprechung mit den Verantwortlichen des Hamburger Flughafens entwickelt und basiert auf deren Annahmen.

Im Laufe dieses Projekts konnte nicht auf reelle Daten (Zeitstempel) zurückgegriffen werden. Dieses ist dem Umstand geschuldet, dass die Scanner bis zum Projektstart noch nicht in Betrieb genommen wurden und später kein Zugriff auf die Daten der Ticketscanner möglich war.

Für die Bestimmung der WIRD-Wartezeiten stehen Prognosen in der Form von Passagieraufkommen bereit. Der Flughafen Hamburg erstellt diese Prognosen für einen Zeitraum von bis zu zwei Wochen. Für diesen Zeitraum wird das Passagieraufkommen in 15 Minuten Blöcken erstellt und gibt jeweils an, wie viele Passagiere in den 15 Minuten erwartet werden.

Für die Realisierung der mobilen Anwendung werden keine Daten benötigt, die nicht innerhalb dieses Projektes ermittelt wurden.

2.3 Aufgaben

Die Aufgaben und Anforderungen im Rahmen dieses Projekts basieren auf den fünf definierten Zielen, welche im Kapitel 1.1 behandelt wurden.

Da für die Berechnung der IST-Wartezeit keine reellen Daten zur Verfügung standen, war es erforderlich, diese zu simulieren, um eine Möglichkeit zu erschaffen, die implementierten Lösungen zu testen. Darum musste zu allererst ein Simulator entwickelt werden. Der Nutzen eines Simulators besteht darin, die Zeitstempel (erster und zweiter Zeitstempel) zu generieren.

Der Simulator soll speziell das Wartesystem des Hamburger Flughafens widerspiegeln und als ereignisorientierte Simulation konzipiert und implementiert werden. Um die Aufgabe zu definieren, müssen zunächst die Anforderungen an den Simulator ermittelt werden. Aus diesem Grund werden die einzelnen Bestandteile des Wartesystems analysiert.

Der Ankunftsprozess teilt sich beim Hamburger Flughafen auf mehrere Eingänge auf, von denen aus alle Passagiere in die gleiche Warteschlange gelangen. Weil die Passagiere alle in dieselbe Warteschlange eingereiht werden, kann der Ankunftsprozess anhand eines Eingangs simuliert werden.

Es können maximal 24 Kontrollspuren gleichzeitig geöffnet sein, von denen jede eine individuelle Servicerate haben kann. Der Simulator muss also eine Anzahl von bis zu 24 Kontrollspuren simulieren, von denen jede eine individuelle Servicerate enthält. Die Servicerate einer Kontrollspur gibt an, wie lange diese für die Abfertigung eines Passagiers benötigt.

Eine Begrenzung der Population gibt es bei dieser Warteschlange nicht, da alle Passagiere diese durchlaufen müssen, um zu den Flugzeugen zu gelangen. Da keine Alternative besteht, wird angenommen, dass sich weitere Passagiere unabhängig von der Warteschlangenlänge anstellen werden.

Als Abfertigungsprinzip wird bei dieser Warteschlange *First In First Out* (*FIFO*) angenommen. *FIFO* bedeutet, dass die Passagiere die Warteschlange in der Reihenfolge verlassen, in der sie diese betreten haben.

Neben diesen Spezifikationen muss der Simulator über eine Konfigurationsmöglichkeit verfügen, in der die Konfigurationsparameter für die Simulation definiert werden. Zu diesen Parametern zählen unter anderem der Zeitrahmen der Simulation, die Öffnung der Kontrollspuren, die Servicerate jeder einzelnen Kontrollspur sowie die Ankunftsrate. Die Ankunftsrate gibt an, in welchem zeitlichen Abstand Passagiere die Warteschlange betreten. Die Konfigurationsmöglichkeit muss so konzipiert sein,

dass sich die Konfigurationsparameter für jede Millisekunde der Simulation definieren lassen.

Die Ergebnisausgabe des Simulators soll in Form einer Liste sein, welche die generierten Daten enthält. Diese Daten müssen die Passagierkennung (id), Ankunfts-Zeitstempel (erster Zeitstempel), Servicestart-Zeitstempel (zweiter Zeitstempel), Service-Zeit, Serviceend-Zeitstempel (Zeitpunkt der vollständigen Abfertigung) sowie eine Kennung der benutzten Kontrollspur für jeden einzelnen Passagier enthalten.

Der so eingerichtete Simulator bietet mehr Daten an, als das derzeitige nur aus zwei T2-Ticketscannern bestehende Setup. Statt den Zeitstempel für Passagiere lediglich zweier Kontrollspuren zu erstellen (siehe Abbildung 1.1), generiert der Simulator sie für jeden Passagier. Diese Eigenschaft ermöglicht es, für jeden Passagier die tatsächliche Wartezeit zu ermitteln, um diese anschließend mit der berechneten IST-Wartezeit abzugleichen. Außerdem kann so getestet werden, inwiefern sich die Ergebnisse bei einer höheren Datenausbeute verhalten.

Sobald der Simulator implementiert ist, besteht die nächste Aufgabe darin, Lösungen für die Berechnung der IST- und WIRD-Wartezeiten zu konzipieren und zu implementieren. Diese werden anschließend mit dem Simulator getestet und verbessert. Die IST-Wartezeit soll eine möglichst präzise Angabe der aktuellen Wartezeit liefern. Diese muss mit dem derzeitigen aus Ticketscannern bestehendem Setup funktionieren. Die Anforderung an die WIRD-Wartezeit besteht darin, aus den getätigten Prognosen in Form von Passagieraufkommen eine Wartezeit zu berechnen.

Im Anschluss gilt es, diese entwickelten Lösungen in Form von Algorithmen auf einem Anwendungsserver[2] zu installieren. Ein Anwendungsserver stellt eine Umgebung auf einem Server bereit, in der Anwendungen ausgeführt werden können. Auf diesem Server können stets die Algorithmen zur Berechnung der Wartezeit ausgeführt werden und dessen Ergebnisse mittels einer REST-Schnittstelle[3] bereitgestellt werden. Eine REST-Schnittstelle stellt Ressourcen (Daten) in Form eines Webservices zur Verfügung.

Die mobile Anwendung soll so konzipiert sein, dass eine permanente Kommunikation zur REST-Schnittstelle stattfindet. So ist die verlässliche Anzeige der aktuellsten Werte gewährleistet. Neben der technischen Umsetzung soll die Benutzeroberfläche eine übersichtliche Angabe der Wartezeiten bereitstellen. Eine weitere Anforderung ist die Unabhängigkeit von Betriebsplattformen. Wenn eine Anwendung plattformunabhängig[4] ist, kann sie auf verschiedenen Endgeräten mit unterschiedlichen Betriebs-

[2](Barry, Douglas K. 2016)
[3](Rodriguez, Alex 2015)
[4](E-Teaching 2015)

systemen ausgeführt werden. Aus diesem Grund muss die Anwendung mithilfe eines Frameworks[5] realisiert werden, das dieses unterstützt. Ein Framework gibt den Entwicklungsrahmen für die zu programmierende Anwendung vor.

Die Abbildung 2.1 verleiht einen Überblick, inwiefern die einzelnen Aufgaben nach erfolgreicher Implementation miteinander interagieren. Auf dem Server werden die IST- und WIRD-Wartezeiten berechnet. Das geschieht anhand des in dieser Arbeit entworfenen Lösungswegs mithilfe der Daten aus der Datenbank. Da der Zugriff auf die Datenbank im Laufe des Projektes nicht möglich war, wurde der Simulator implementiert, um diese Daten bereitzustellen. Die REST-Schnittstelle greift auf die berechneten Wartezeiten zu und stellt diese der bereit. Die mobile Anwendung greift auf die REST-Schnittstelle zu, um auf diesem Wege die berechneten Wartezeiten zu bekommen, um sie anschließend anzuzeigen.

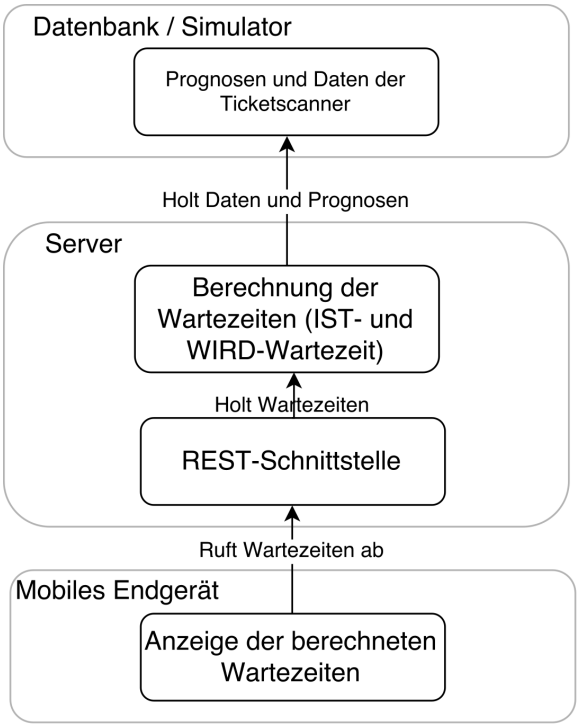

Abbildung 2.1: Interaktion zwischen den entwickelten Elementen

[5](Rouse, Margaret 2015)

3 Programmiersprache

Für die Ermittlung der Wartezeiten muss die Programmiersprache keine besonderen Eigenschaften aufweisen. Da die Algorithmen jedoch auf den Anwendungsservern des Hamburger Flughafens ausgeführt werden sollen, bietet es sich an, die Algorithmen in der Programmiersprache zu entwickeln, die diese Anwendungsserver unterstützen. Die Anwendungsserver werden mit der Software JBoss Enterprise Application Platform[1] betrieben. Diese ermöglicht unter anderem das Ausführen von Java Anwendungen. Darum fällt die Wahl der Programmiersprache auf Java.

Der Simulator soll ein bereits existentes System, in dem Elemente mit verschiedenen Eigenschaften vorkommen, nachbilden. Deshalb bietet es sich an, diesen nach dem objektorientierten Prinzip[2] zu gestalten. Bei der objektorientierten Programmierung werden Daten und Funktionen in Objekten zusammengefasst. Dies hat den Vorteil, dass die Elemente des Warteschlangensystems in unterschiedliche Klassen gefasst werden können, deren Eigenschaften und Attribute in dieser definiert werden. Eine *Klasse*[3] legt fest, welche Methoden und Attribute ein Objekt beinhaltet und definiert so den inneren Zustand eines Objekts. Sobald eine Klasse initialisiert wurde, wird diese als *Objekt* betitelt. Objekte können schlicht als Datenspeicher benutzt werden oder auch Logik beinhalten. Die Aufteilung der einzelnen Elemente des Warteschlangensystems in Klassen bietet eine übersichtliche und durchstrukturierte Lösung für die Entwicklung des Simulators. Weil die Algorithmen für die Ermittlung der Wartezeiten in Java programmiert werden, welches auch eine objektorientierte Programmiersprache ist, fällt die Wahl für den Simulator ebenfalls auf Java.

Für die Anwendungsentwicklung gibt es diverse Frameworks, die die Plattformunabhängigkeit ermöglichen. Das sind unter anderem *Cordova* und *Appcelerator*. Beide Frameworks setzen die Verwendung von JavaScript voraus. Aus diesem Grund fällt hier die Wahl auf JavaScript.

[1] (JBoss 2016)
[2] (Lahres, Bernhard und Rayman, Gregor 2006)
[3] (IT-Handbuch 2007)

4 Konzeption und Implementierung

Dieses Kapitel befasst sich mit der Umsetzung der definierten Ziele und beschreibt diese chronologischer Reihenfolge.

Das Kernstück dieses Projektes ist die Ermittlung der IST-Wartezeit. Aufgrund des Fehlens der Daten, die für das Testen der Lösungswege für die IST-Wartezeit erforderlich gewesen wären, besteht der erste Schritt aus der Implementierung des Simulators.

Danach beginnt die Entwicklung von Lösungswegen für die IST-Wartezeit, die anschließend implementiert und getestet werden.

Daran schließt sich die Einpassung der Lösung für die Berechnung der WIRD-Wartezeit an. Diese wird aus den bereitgestellten Prognosen ermittelt.

Nach der erfolgreichen Implementierung beider Lösungswege werden diese auf dem bereits eingerichteten Anwendungsserver installiert.

Im Anschluss wird ein Konzept für Benutzeroberfläche und technische Funktionalität der mobilen Anwendung entworfen und diese mithilfe der ausgewählten Technologien realisiert.

4.1 Begriffsdefinition

Bevor die einzelnen Arbeitsschritte erläutert werden, sollen zunächst noch einige Begriffe definiert werden, die im Verlauf dieser Arbeit regelmäßig auftauchen.

Dieses Projekt ist mit der Programmiersprache Java nach dem objektorientierten Prinzip aufgebaut. In diesem Zuge werden die Bestandteile, aus denen ein Warteschlangensystem besteht, in Klassen gefasst. Jedes Objekt besitzt eine eindeutige Kennung, bei der Klasse des Passagiers ist es die Ticketnummer, bei der Klasse der Kontrollspur die Kennung des zugehörigen T2-Ticketscanners. So lässt sich jedes Objekt eindeutig identifizieren und dem korrespondierenden Passagier oder der korrespondierenden Kontrollspur zuordnen. Die Klasse für den Passagier trägt den Namen *Pax*, die Klasse für die Kontrollspur den Namen *Counter*. Wird ein *Pax-Objekt* erwähnt, wird also das Objekt beschrieben, welches die Daten eines bestimmten Passagiers enthält. Das *Counter-Objekt* enthält alle Daten über die zugehörige

Kontrollspur. Die Begriffe Pax-Objekt und Counter-Objekt beschreiben demnach die jeweilige Abbildung des im echten Warteschlangensystem existenten Passagiers und der existenten Kontrollspur. Sie enthalten alle Daten, die über diese ermittelt werden können.

4.2 Der Simulator

Der Simulator ist ein unverzichtbarer Bestandteil des Projektes, da er jene Daten liefert, auf denen die Berechnung der IST-Wartezeit basiert. Mithilfe der vom Simulator erstellten Daten wird es ermöglicht, die konzipierten Algorithmen zu testen.

Nach Definition des Vereins Deutscher Ingenieure ist eine Simulation die *„Nachbildung eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigen Modell“* [1]. Weiterhin heißt es, dass das Ziel einer Simulation daraus besteht, die Erkenntnisse aus dieser in die Realität zu übertragen[2]. Das entspricht exakt der Aufgabe der Simulation in diesem Projekt. Der Simulator simuliert die Zeitstempel aus dem Warteschlangensystem und speichert diese als Pax-Objekte ab. Das Resultat aus der Simulation ist eine Liste aus Pax-Objekten, in denen die generierten Daten gespeichert sind. Jedes dieser Pax-Objekte beinhaltet die Ticketnummer, den ersten und zweiten Zeitstempel sowie die Kennung der benutzten Kontrollspur. Der Simulator liefert also mehr Daten als die tatsächlich verfügbaren Ticketscanner bei den Warteschlangen des Flughafens, da er für jeden Passagier den ersten und zweiten Zeitstempel abspeichert.

Da die Simulation zu Testzwecken benötigt wird, ist ihre Implementierung als ereignisorientierte Simulation[3] unvermeidbar. Der Fortschritt einer ereignisorientierten Simulation wird durch Ereignisse vorgetäuscht. Diese Ereignisse lösen Kausalzusammenhänge aus, welche den Simulationsfortschritt voranschreiten lassen. In diesem Zusammenhang hat dies den Vorteil, dass die Zeitstempel nicht von der tatsächlichen Zeit definiert, sondern aufgrund der Ereignisse generiert werden. So ist es möglich, einen beliebig großen Zeitraum innerhalb kürzester Zeit zu simulieren. Die Zeit der Simulation ist abhängig von der Hardware des ausführenden Systems.

4.2.1 Architektur

Als Inspiration für die Grundlage des Simulators wurde der Algorithmus einer diskreten Warteschlangensimulation verwendet, der vom Benutzer *Vince Knight* auf der Plattform GitHub zur freien Verfügung bereitgestellt wurde[4]. Dieser Algorithmus

[1] (IT-Handbuch 2007)
[2] (IT-Handbuch 2007)
[3] (Rouse, Margaret 2012)
[4] (Knight, Vince 2013)

wurde innerhalb dieses Projektes so modifiziert, dass er der Definition der Warteschlange des Hamburger Flughafens entspricht. Das bedeutet, dass sowohl der Ankunftsprozess als auch der Bearbeitungsprozess angepasst wurden.

Der Simulator ist objektorientiert konzipiert und besteht daher aus mehreren Klassen. Die Simulation im eigentlichen Sinne findet in der Klasse *Simulator* statt. Es wird unterschieden zwischen Model-Klassen wie zum Beispiel *Pax* und *Counter* und Logik-Klassen wie zum Beispiel die Klasse *Simulator*. Model-Klassen werden lediglich als Datenspeicher verwendet, die Logik-Klassen dagegen beinhalten die Programmlogik. Das UML-Diagramm in Abbildung 4.1 gibt Aufschluss über die Assoziationen zwischen den einzelnen Klassen.

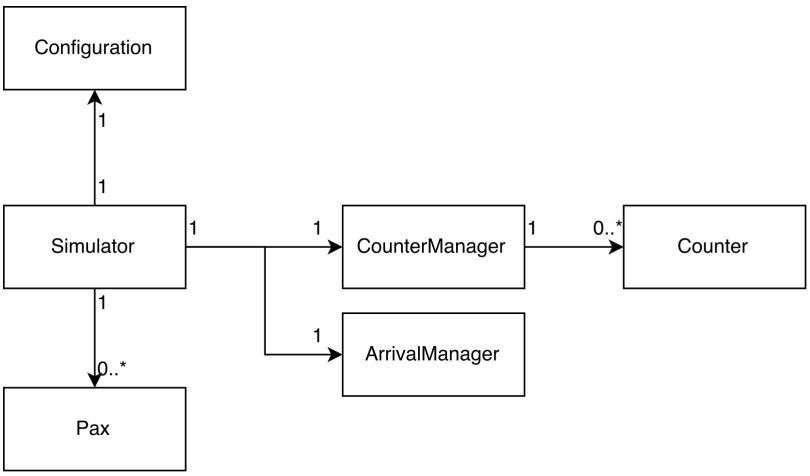

Abbildung 4.1: Assoziationen zwischen den Klassen als UML-Diagramm

Im Folgenden werden die fünf wichtigsten Klassen beschrieben. Da die Klasse *Simulator* die Simulationslogik beinhaltet, wird in dem Abschnitt 4.2.1 (Die Klasse Simulator) der Ablauf der Simulation erläutert.

Die Konfigurationsklasse

Die Konfigurationsklasse (*Configuration*) ist eine Model-Klasse, da sie allein für die Speicherung von Konfigurationsparametern konzipiert ist. Die wichtigsten dieser Parameter sind jene für die Definition des Simulationszeitraumes, für den Zeitplan der Kontrollspuren-Öffnung sowie die für den Zeitplan der Ankunftsraten.

Der Zeitraum der Simulation lässt sich in Form von Millisekunden angeben. Der Startzeitpunkt wird als Unix Zeitstempel angegeben. Dieser beschreibt die vergangene Zeit seit dem 1. Januar 1970 00:00:00 Uhr in Millisekunden. Die Dauer der Simulation wird ebenfalls in Millisekunden angegeben, jedoch nicht als Zeitstempel. Der Startzeitpunkt in Addition mit der Dauer der Simulation ergibt also den Zeit-

stempel für das Ende der Simulation.

Die Kontrollspurenöffnung gibt an, welche und wie viele Kontrollspuren zu einem beliebigen Zeitpunkt der Simulation geöffnet sind. Zu diesem Zweck werden der Zeitpunkt sowie die korrespondierende Anzahl der geöffneten Kontrollspuren in einer Map angegeben. Eine Map ist ähnlich einer Liste, jedoch mit einem Schlüssel, der mit einem beliebigen Datentyp belegt werden kann. Diese Eigenschaft der Map ermöglicht es, den Zeitpunkt vom Typen *Long* als Schlüssel sowie die Anzahl der geöffneten Spuren in Form eines *Integer* als Wert anzugeben. Der Schlüssel ist hier die Zeit seit Beginn der Simulation in Millisekunden. Mithilfe dieser Zeit kann die zugehörige Anzahl geöffneter Kontrollspuren ermittelt werden. In dem Codebeispiel 4.1 ist die Befüllung der Map *timeTable* zu sehen.

Listing 4.1: Beispiel für die definition der Kontrollspurenöffnung

```
timeTable.put(900000L, 1);       //0:00 - 0:15
timeTable.put(1800000L, 13);     //0:15 - 0:30
timeTable.put(3600000L, 18);     //0:30 - 1:00
timeTable.put(9900000L, 23);     //1:00 - 2:45
```

Die Ankunftsraten werden ebenfalls in einer Map definiert. In diesem Fall besitzt der Schlüssel den Datentyp *Range* und der Wert den Datentyp *Long*. Der Schlüssel beschreibt den Zeitraum und der Wert die Ankunftsrate. Eine *Range* definiert einen Zeitraum. Dieser wird als Schlüsselwert verwendet, da pro Simulation jedem Zeitraum nur ein Wert für die Ankunftsrate zugewiesen werden darf. Das Codebeispiel 4.2 veranschaulicht die Befüllung der Map *arivalRates*.

Listing 4.2: Beispiel für die definition der Ankunftsrate

```
arrivalRates.put(Range.closed(0L, 900000L), 1250L);
arrivalRates.put(Range.closed(900000L, 1800000L ), 60000L);
arrivalRates.put(Range.closed(1800000L, 2700000L ), 20000L);
arrivalRates.put(Range.closed(2700000L, 3600000L ), 500L);
```

Diese Klasse wird zum Beginn jeder Simulation definiert und ist unerlässlich für die Funktion der Simulation.

Die Klasse Pax

Die Klasse *Pax* ist eine Model-Klasse und beinhaltet Attribute für die Informationen eines Passagiers. In dieser Klasse werden alle Daten gespeichert, die im Laufe der Simulation generiert werden. Sie besitzt die Attribute *id* (Passagierkennung), *arrivalDate* (erster Zeitstempel), *serviceStartDate* (zweiter Zeitstempel), *serviceTime* (Servicezeit), *counter* (Kennung der Kontrollspur) sowie den Zeitstempel *serviceEndDate*

(Zeitpunkt vom Ende des Services). In der Abbildung 4.2 ist diese Klasse als UML-Diagramm dargestellt, dem nicht nur die Attribute sondern auch deren Datentypen entnommen werden können, die in dieser Klasse enthalten sind.

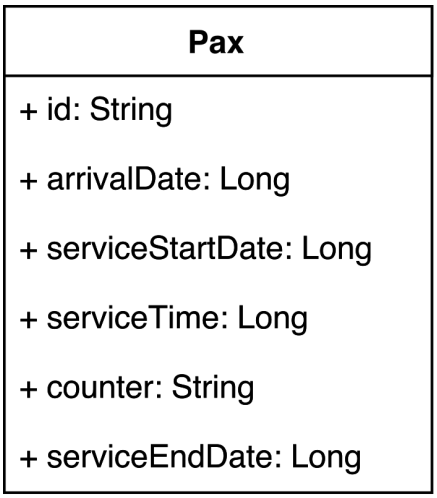

Abbildung 4.2: Die Klasse *Pax*

Die Klasse Counter

Diese Klasse ist ebenfalls eine Model-Klasse und beinhaltet Attribute für die Information der zugehörigen Kontrollspur. In der Abbildung 4.3 ist das UML-Diagramm der Klasse Counter mit den enthaltenen Attributen und Methoden zu sehen. Neben dem Attribut *id* (Kennung der Kontrollspur) besitzt sie noch die Attribute *blocked*, *serviceRate* und *counterClose*.

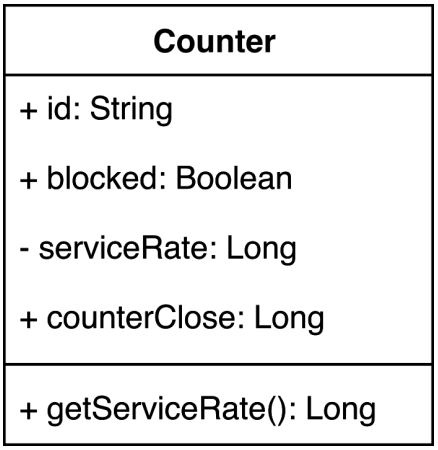

Abbildung 4.3: Die Klasse *Counter*

Das Attribut *blocked* gibt den Zeitpunkt an, bis zu dem das Counter-Objekt geblockt ist. Das Blocken simuliert die Zeit, bis die Kontrollspur einen Passagier abgearbeitet hat und den nächsten übernehmen kann.

Das Attribut *serviceRate* beinhaltet die dem Counter-Objekt zugeteilte Servicerate. Die Servicerate ist die Zeit in Millisekunden, die das Counter-Objekt benötigt, um einen Passagier abzufertigen. Dieses Attribut ist privat und kann nur mittels der Methode *getServiceRate* abgerufen werden, die das Attribut *serviceRate* mit einer konfigurierbaren Abweichung zurückgibt. Diese Abweichung simuliert die Varianz der Bearbeitungsgeschwindigkeit der Kontrollspuren.

Das Attribut *counterClose* ist ein Zeitstempel. Dieser gibt an bis zu welchem Zeitpunkt das Counter-Objekt aktiv ist, also bis zu welchem Zeitpunkt die Kontrollspur geöffnet ist.

Der CounterManager

Die Klasse *CounterManager* ist eine Logik Klasse und generiert, basierend auf dem in der Konfiguration beschrieben Plan der Spurenöffnung, einen sogenannten Schichtplan. Dieser wird von der Klasse *CounterManager* zur Verfügung gestellt. Eine Schicht basiert auf dem zugeteilten Zeitraum sowie der zugehörigen Anzahl der geöffneten Kontrollspuren. Der *CounterManager* stellt für jeden konfigurierten Zeitraum eine Liste mit bereits initialisierten Counter-Objekten zur Verfügung. Diese Counter-Objekte fungieren dann als offene Kontrollspuren. Je nach Zeitraum wird eine der Schichten ausgewählt und beinhaltet die definierte Anzahl an aktiven Counter- Objekten. Aktive Counter-Objekte spiegeln die geöffneten Kontrollspuren wieder.

Die Klasse Simulator

Die Klasse *Simulator* ist ebenfalls eine Logik-Klasse, denn sie beinhaltet die Logik der eigentlichen Simulation. Diese kann nur dann erfolgreich ausgeführt werden, wenn beim Aufruf ein Objekt der Konfigurationsklasse mit den definierten Konfigurationsparametern übermittelt wird. Zunächst wird die übermittelte Konfiguration verarbeitet. Dieses geschieht unter anderem indem die Klassen *CounterManager* und *ArrivalManager* initialisiert werden. Der *CounterManager* ist für die Schichtplanerstellung zuständig. Aus der Liste mit den initialisierten Schichten kann anhand vom Zeitparameter ermittelt werden, welche Counter-Objekte aktiv sind. Die Klasse *ArrivalManager* beinhaltet den vorher konfigurierten Plan der Ankunftsraten und stellt die Ankunftsrate bereit.

Nach der Initialisierung der Konfiguration beginnt die Simulation. Diese wird mithilfe einer Schleife realisiert. Die Klasse Simulator besitzt das Attribut *clock*, welches den aktuellen Zeitparameter definiert, den es in jeder Iteration zu erhöhen gilt. Sobald

dieser den definierten Endzeitpunkt der Simulation erreicht, wird diese beendet. Zu Anfang jeder Iteration wird das Objekt der Klasse *ArrivalManager* über den aktuellen Zeitparameter informiert, um die zur Iteration passende Ankunftsrate bereitzustellen. Die Essenz jeder Iteration ist das Generieren eines vollständigen Pax-Objekts mit folgenden Werten:

- Zeitstempel beim Betreten der Warteschlange (erster Zeitstempel)
- Zeitstempel beim Beginn des Services und somit beim Verlassen der Warteschlange (zweiter Zeitstempel)
- Kennung der benutzten Kontrollspur
- Zeitstempel beim Ende des Services

Ein kommentierter Quellcode dieser Schleife befindet sich im Kapitel A.1 Listing A.1. Aus Gründen der Übersicht wurde der Quellcode an dieser Stelle nicht im Text integriert. Im Folgenden wird zu jeder Aktion die Zeilenzahl des Listings angegeben.

Erster Zeitstempel Dieses Attribut entspricht dem Zeitparameter der aktuellen Iteration in Addition mit der zur aktuellen Iteration gehörenden Ankunftsrate (Zeile 15 und 55). Da die Ankunftsrate den Abstand zwischen dem Eintreffen der einzelnen Passagiere definiert, muss diese in jeder Iteration zum aktuellen Zeitparameter (*clock*) addiert werden. Dies geschieht, indem am Ende jeder Iteration der Zeitparameter gleich dem Wert des ersten Zeitstempels gesetzt wird (Zeile 75).

Zweiter Zeitstempel Der zweite Zeitstempel ist abhängig von dem Counter-Objekt, welchem das Pax-Objekt zugeordnet wird. Die Zuordnung findet aufgrund der Auslastung der Counter-Objekte statt. Es wird jenes ausgewählt, das als nächstes frei wird (Zeile 23). Die Information, für wie lange ein Counter-Objekt blockiert ist, befindet sich in dem Attribut *blocked* des Counter-Objekts. Die Abbildung 4.4 veranschaulicht diesen Auswahl Prozess. In der Zeitleiste mit der Beschriftung *arrival* ist die Ankunftszeit von Passagier 1 (*Pax1*) verzeichnet. Anschließend wird ermittelt, welche der drei Kontrollspuren (Spur 1 - Spur 3) als nächstes frei wird, also nicht mehr blockiert ist. In diesem Fall ist es die Spur 1. Nachdem die Kontrollspur ausgewählt wurde, wird diese für die benötigte Bearbeitungszeit, welche der Servicerate entspricht, geblockt. Im Beispiel entspricht das einer Zeit von 50 Sekunden.

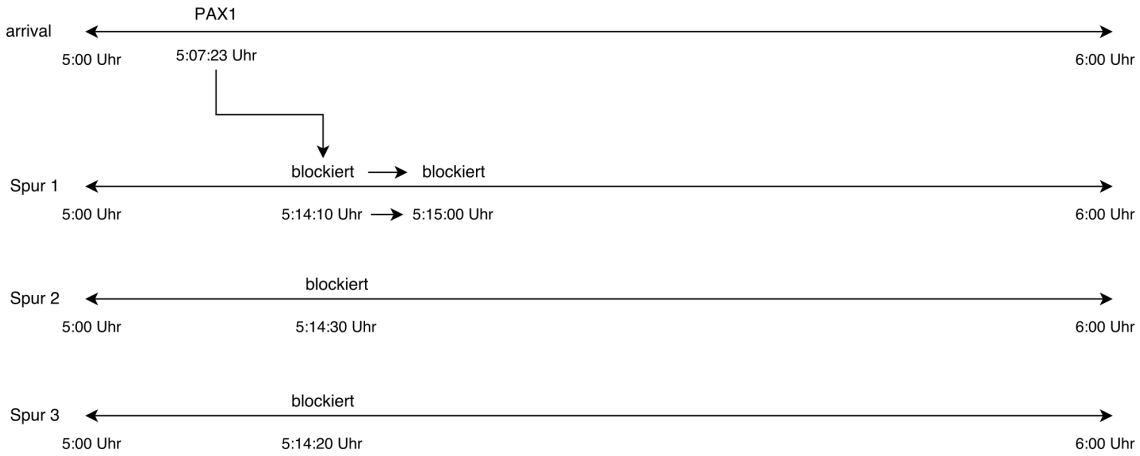

Abbildung 4.4: Auswahl der Kontrollspur

Gleichzeitig wird auch überprüft, ob das ausgewählte Counter-Objekt noch geöffnet ist und zwar anhand des Parameters *closed*. Ist diese Kontrollspur bereits geschlossen, wird aus dem Schichtplan die nächste Schicht ausgewählt (Zeile 25 - 42). Der Zeitüberschuss der restlichen Counter-Objekte wird addiert und auf die neue Schicht übertragen (Zeile 43 - 46). Der Zeitüberschuss setzt sich aus der Differenz zwischen dem *clock* und *blocked* Parameter der einzelnen Kontrollspuren zusammen. Daraufhin wird aus dieser Schicht ein passendes Counter-Objekt ausgewählt (Zeile 48).

Wenn der erste Zeitstempel eines Pax-Objekts einen späteren Zeitpunkt beschreibt als der *blocked* Parameter des Counter-Objekts, so ist der erste Zeitstempel in Addition mit einem Parameter für die Wegzeit der zweite Zeitstempel. Das bedeutet, dass keine Wartezeit sondern lediglich die Wegzeit anfällt, wenn ein Passagier die Warteschlange betritt, ohne dass weitere Personen vor ihm anstehen und alle Kontrollspuren frei sind. Wenn der *blocked* Parameter des ausgewählten Counter-Objekts einen späteren Zeitpunkt beschreibt als der erste Zeitstempel, und die Differenz dieser beiden Zeitstempel größer ist als der Wegzeit Parameter, so bestimmt dieser den zweiten Zeitstempel. Das bedeutet dann, dass eine Wartezeit anfällt und der Passagier, der jetzt die Warteschlange betritt, aktiv warten muss (Zeile 58 - 65).

Nachdem diese Attribute bestimmt wurden, wird der *blocked* Parameter des ausgewählten Counter-Objekts mit der konfigurierten Servicerate addiert, sodass dieser jetzt für die benötigte Servicerate blockiert ist (Zeile 73). Das bedeutet, die Kontrollspur ist solange für andere Passagiere gesperrt, bis der Passagier dort abgefertigt wurde.

Kennung der benutzten Kontrollspur Die Kennung des benutzten Counter-Objekts wird aus dem Attribut *id* des ausgewählten Counter-Objekts bestimmt (Zeile 70).

Damit lässt sich jedes Pax-Objekt dem zugehörigen Counter-Objekt zuordnen.

Zeitpunkt der vollständigen Abfertigung Für den Zeitpunkt der vollständigen Abfertigung werden der zweite Zeitstempel und die benötigte Servicezeit addiert (Zeile 68). Dieses Attribut ist für die Berechnung der IST-Wartezeit nicht relevant, wird jedoch generiert, um eine vollständige Information über den generierten Passagier zu erhalten.

4.2.2 Zusammenfassung

Der Simulator erstellt basierend auf der Konfiguration Pax-Objekte, also Informationen über Passagiere. Diese werden in einer Liste abgespeichert und können anschließend für jegliche weitere Operationen verwendet werden. Die Nutzung dieses Simulators bietet viele Vorteile gegenüber der Nutzung von realen Daten. Da dieser für jeden Passagier zwei Zeitstempel (erster und zweiter Zeitstempel) generiert, kann mit dessen Hilfe dieser die tatsächliche Wartezeit für jeden Passagier bestimmt werden, um diese zum Beispiel im Anschluss mit den Vorhersagen zu vergleichen. Des Weiteren bietet diese Eigenschaft die Möglichkeit, die Lösungsvorschläge auf ihr Verhalten bei einer beliebig hohen Datenausbeute zu untersuchen.

4.3 Die Vorhersage

Zu diesem Kapitel zählt sowohl die Vorhersage der aktuellen IST-Wartezeit als auch die Vorhersage der WIRD-Wartezeiten. Diese Aufteilung ist so gewählt, da beide Wartezeiten Vorhersagen sind, die auf analysierten Kennziffern basieren. Ein besonderer Fokus wird hier auf die Ermittlung der IST-Wartezeit gelegt.

4.3.1 Ermittlung der IST-Wartezeit

Diese Arbeit erklärt und vergleicht zwei Lösungswege für die Berechnung der IST-Wartezeit. Bei denen handelt es sich um die *Lösung 1*, welche die höchstmögliche Anzahl an Daten benötigt und um *Lösung 2*, die bereits mit einer geringen Ausbeute an Daten auskommt. Die Höhe der Datenausbeute wird dadurch definiert, wie viele Passagiere den zweiten Zeitstempel besitzen. Obwohl die hohe Datendichte, welche der erste Lösungsvorschlag benötigt, beim Start des Projektes nicht umsetzbar war, wird dieser dennoch näher erläutert, um die Erkenntnis darüber zu vermitteln, welchen Mehrwert der Bezug von mehr Daten hat. Diese Lösung wäre ohne die Nutzung des Simulators nicht möglich, da nur er die hohe Datenausbeute simulieren kann. Wird in dieser Arbeit eine Lösung auf Basis der Anzahl der installierten Ticketscanner entwickelt, so gilt die Voraussetzung, dass diese über die gesamte Zeit geöffnet sind.

Damit die Lösungsvorschläge die Daten von der Simulation erhalten können, muss zunächst eine Schnittstelle definiert werden. Eine Schnittstelle bietet ein einheitliches Format, in dem die Informationen an die Algorithmen weitergegeben werden. Dies bietet den Vorteil, dass die Informationen nicht für jeden Algorithmus individuell angepasst werden müssen.

Die Schnittstelle

Die Schnittstelle definiert eine einheitliche Notation der zu übertragenden Informationen. Diese muss alle Informationen umfassen, welche für die Ermittlung der IST-Wartezeit erforderlich sind und zur Verfügung stehen. Die Informationen beschränken sich auf die zwei Zeitstempel sowie die Passagierkennung und die Kontrollspurkennung. Damit die IST-Wartezeit immer aktuell berechnet werden kann, muss der Algorithmus, unmittelbar nachdem ein Zeitstempel abgegeben wurde, diese Informationen bekommen. Aus diesem Grund bietet es sich an, die Schnittstelle in Form von Ereignissen zu definieren. Ein Ereignis ist in diesem Kontext die Erkenntnis neuer Informationen. Ein Ereignis wird also ausgelöst, wenn ein Ticketscanner benutzt wird.

Das erste Ereignis ist jenes, welches gestartet wird, wenn ein Passagier den ersten Zeitstempel abgibt. Der Name für dieses Ereignis ist daher *checkin*, da die Person sozusagen in die Warteschlange eincheckt. Bei diesem Ereignis werden lediglich die Passagierkennung und der zugehörige Zeitstempel übergeben.

Das zweite Ereignis ist das des zweiten Zeitstempels. Der Name für dieses Ereignis ist *checkout*, da der Passagier die Warteschlange wieder verlässt und somit aus dieser auscheckt. Dieses Ereignis beinhaltet die Passagierkennung, die Kennung der benutzten Kontrollspur und den zugehörigen Zeitstempel.

Um sicherzustellen, dass jeder der entwickelten Algorithmen die grundlegenden Funktionen besitzt, um der Definition der Schnittstelle zu entsprechen, wurde zunächst ein Interface implementiert. Ein Interface definiert die Funktionen, welche eine Klasse besitzen muss. Wenn eine Klasse ein Interface besitzt, beinhaltet diese Klasse alle Funktionen, welche im Interface definiert wurden. In Abbildung 4.5 ist das UML-Diagramm des Interface mit dem Namen *Predictor* zu sehen.

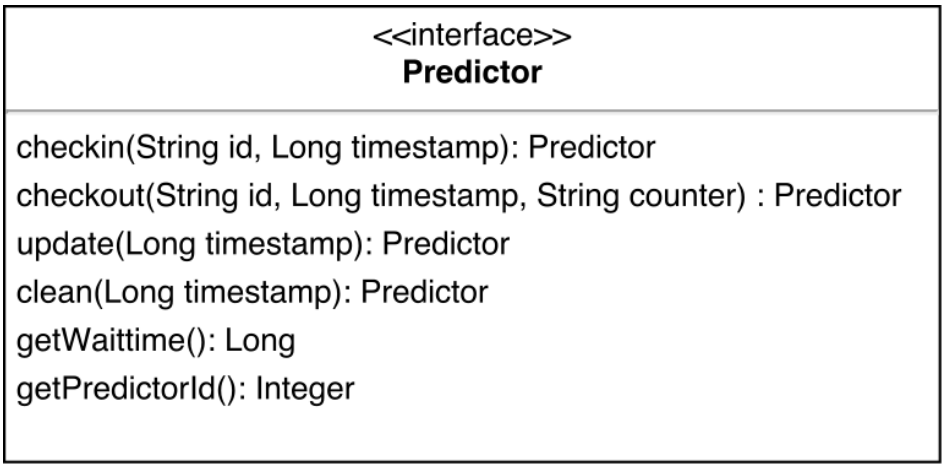

Abbildung 4.5: Das Interface *Predictor*

Die beiden Funktionen *checkin* und *checkout* sind zuständig für die von der Schnittstelle definierten Ereignisse und erwarten die Eingabe der korrespondierenden Informationen.

Die Funktion *update* beinhaltet die Logik der Wartezeitenberechnung. Diese Funktion soll immer dann aufgerufen werden, wenn es neue Informationen für die Prognose gibt. Dies kann von Algorithmus zu Algorithmus unterschiedlich sein.

Die Funktion *clean* ist für das Aufräumen zuständig. Diese entfernt zum Beispiel Pax- oder Counter-Objekte, welche ihre Gültigkeit verloren haben. Eine Definition, unter welchen Umständen ein Pax- oder Counter-Objekt die Gültigkeit verliert, befindet sich in der nachfolgenden Beschreibung der Klassen.

Die Funktion *getWaittime* liefert beim Aufruf die aktuell ermittelte Wartezeit zurück.

Die Klasse Pax

Die Klasse *Pax* (Abb. 4.6) beinhaltet Attribute für alle Informationen, die über einen Passagier in Erfahrung gebracht wurden oder aber speziell für diesen definiert wurden.

Neben den trivialen Attributen wie *id* (Passagierkennung), *arrivalDate* (erster Zeitstempel), *serviceStartDate* (zweiter Zeitstempel) und *counter* (Kennung der Kontrollspur) beinhaltet diese Klasse das Attribut *timeout*. Dieses Attribut legt den Zeitpunkt fest, an dem ein Pax-Objekt die Gültigkeit verliert. Die Ursache für das Verlieren der Gültigkeit ist, wenn der zugehörige Passagier die Warteschlange verlässt, ohne den zweiten Zeitstempel abzugeben. Für die Berechnung der Wartezeit ist es wichtig, dass dieses Pax-Objekt entfernt wird, weil andernfalls die Berechnung auf mindestens ei-

nem Passagier zu viel basieren würde.

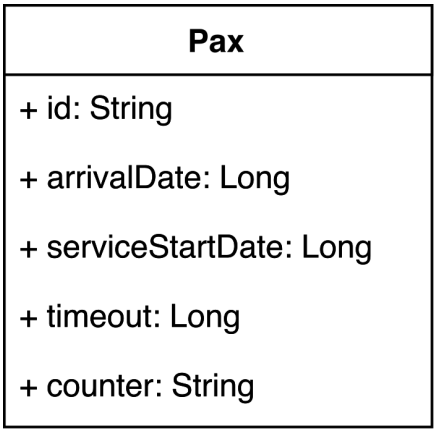

Abbildung 4.6: Die Klasse *Pax*

Die Klasse Counter

Die Klasse *Counter* (Abb. 4.7) ist die zugehörige Klasse der Kontrollspur. Sie beinhaltet Attribute für alle Informationen, die basierend auf den Ereignissen über die Kontrollspur in Erfahrung gebracht werden können.

Diese Klasse beinhaltet ebenfalls das Attribut *id* (Kennung der Kontrollspur). Auch hier ist wieder das Attribut timeout enthalten, welches dieselbe Funktion hat wie das *timeout* Attribut in der Pax-Klasse. Dieses Attribut ist besonders wichtig, weil es kein Ereignis gibt, wenn eine Kontrollspur geschlossen wurde. Darum wird das Counter-Objekt aufgrund des *timeout* Attributes ungültig und kann mithilfe dieses Indikators entfernt werden. Ein Counter-Objekt verliert die Gültigkeit, wenn der Zeitpunkt der letzten Nutzung weiter zurück liegt als der *timeout* Zeitstempel. Die Liste *serviceRates* beinhaltet die Zeitstempel, an denen das Counter-Objekt zuletzt einen Passagier zugeteilt bekommen hat. Das Attribut *bufferLength* gibt an, wie viele der letzten Zeitstempel in der Liste *serviceRates* gespeichert werden.

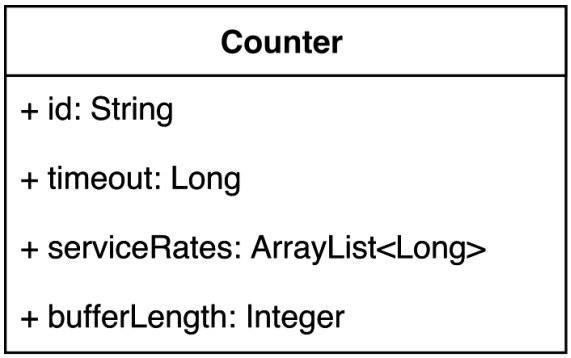

Abbildung 4.7: Die Klasse Counter

Wenn ein Counter-Objekt neu initialisiert wird, wird die Liste *serviceRates* in der angegebenen Länge befüllt (*bufferLength*). Dieses geschieht, indem der Liste generierte Zeitstempel hinzugefügt werden, deren Abstand der durchschnittlichen Bearbeitungsrate entspricht. Dies ist notwendig, damit bereits zu Anfang die Servicerate des Counter-Objekts ermittelt werden kann. Weil die Ermittlung der Servicerate auf den Differenzen der in der Liste *serviceRates* hinterlegten Zeitstempel basiert, muss diese auch Zeitstempel enthalten. Da die Zeitstempel zu Anfang noch nicht gesetzt wurden, werden diese so generiert, dass die resultierende Servicerate der durchschnittlichen Servicerate entspricht. Sobald dieses Counter-Objekt beginnt Pax-Objekte abzufertigen, werden diese generierten Zeitstempel überschrieben und die tatsächliche Servicerate ermittelt.

Lösung 1

Der Grundgedanke dieses Lösungsvorschlags zielt darauf ab, aufgrund der gesammelten Daten sämtliche Parameter zu ermitteln, die sich bei einer Warteschlange für die Berechnung der IST-Wartezeit eignen. Die geeigneten Parameter sind die Ankunftsrate, die Anzahl der sich in der Warteschlange befindenden Passagiere, die Anzahl der geöffneten Kontrollspuren und die zu jeder Kontrollspur gehörige Bearbeitungszeit.

Verwendete Daten Dieser Lösungsweg benötigt für die Funktionsfähigkeit eine nahezu hundertprozentige Erfassung des ersten Zeitstempels in Form des definierten Ereignisses *checkin*. Außerdem ist eine circa hundertprozentige Erfassung des zweiten Zeitstempels in Form des Ereignisses *checkout* erforderlich.

Funktion Die Funktionalität dieser Lösung wird ausschließlich mit der Benutzung der im Interface definierten Methoden realisiert. Das Interface wurde bereits im Kapitel 4.3.1 (Die Schnittstelle) beschrieben.

Die Methode *checkin* initialisiert jenen Passagier, welcher dieses Ereignis ausgelöst hat, in Form eines Pax-Objektes. Dieses Pax-Objekt wird anschließend in einer Liste gespeichert. Diese Liste wird als *queue* bezeichnet und fungiert als Warteschlange. Das hinzugefügte Pax-Objekt beinhaltet zu diesem Zeitpunkt den ersten Zeitstempel, die Passagierkennung (*id*) und das *timeout* Attribut. Das *timeout* Attribut wird aus der zum Zeitpunkt des *checkin* basierenden Wartezeitprognose ermittelt. Diese Prognose wird verdoppelt und anschließend mit dem Zeitstempel des Ereignisses addiert. Das Verdoppeln der Prognose ist notwendig, da das Pax-Objekt ansonsten zu oft als ungültig deklariert wird. Mit der doppelten Prognose gibt es einen gewissen Puffer, der es verhindern soll, dass das Pax-Objekt fälschlicherweise als ungültig deklariert wird. Der so entstandene Zeitstempel definiert den Zeitpunkt, an dem das Pax-Objekt die Gültigkeit verliert.

Die Methode *checkout* ermittelt zunächst das zur mitgelieferten *id* gehörende Pax-Objekt aus der Liste *queue*. Dieses Pax-Objekt wird aus der Liste entfernt, da der Passagier nun abgefertigt wird und nicht mehr Bestandteil der Warteschlange ist. Aufgrund der mitgelieferten Kontrollspurkennung (*counter* in Abb. 4.5) wird ermittelt, ob die Kontrollspur bereits als Counter-Objekt initialisiert ist. Die initialisierten Counter-Objekte befinden sich in einer Liste mit dem Namen *counter*. Existiert dieses bereits, so ist es nicht nötig, es erneut zu initialisierten. Der Liste *serviceRates* des existenten Objekts wird der Zeitstempel dieses Events, also der Zeitstempel beim Verlassen der Warteschlange, beigefügt. Anschließend wird das *timeout* Attribut des existenten Counter-Objekts erneuert, da durch die Benutzung sichergestellt ist, dass die zugehörige Kontrollspur noch aktiv ist. Dieses geschieht, indem der Zeitstempel dieses Events zu der doppelten durchschnittlichen Servicerate addiert wird. Auch hier erfüllt das Verdoppeln wieder den Zweck eines Puffers. Wenn noch kein Counter-Objekt mit der übermittelten Kontrollspurkennung existiert, wird ein neues mit dieser Kennung initialisiert. Für das neu initialisierte Counter-Objekt wird ebenfalls das *timeout* Attribut definiert. Dieses wird nach dem im vorherigen Absatz genannten Schema erstellt.

Mit der Methode *update* wird die aktuelle Wartezeit berechnet. In der Abbildung 4.8 befindet sich ein Beispiel zu dieser Berechnung. Die Einheit dieser Berechnung ist Millisekunde, in dem Beispiel wird jedoch aus Gründen der Übersicht in Sekunden gerechnet. Zunächst wird von jedem initialisierten Counter-Objekt die *serviceRates* Liste abgerufen. In Abbildung 4.8 ist diese bereits mit Beispieldaten befüllt. Von jeder dieser Listen werden nun die Differenzen zwischen den einzelnen Elementen berechnet. In dem Beispiel ergeben sich also aus sechs Elementen fünf Differenzen, die für jedes der Counter-Objekte eine Liste mit den zuletzt benötigten Bearbeitungszeiten ergeben. Für jede dieser Listen wird der Median ermittelt, der den Mittelwert der jeweiligen Liste darstellt. Für jedes der Counter-Objekte wurde so ein Wert für die aktuelle *serviceRate* berechnet. In dem Beispiel resultieren die Zahlen 17, 18, 22 und 17 Sekunden pro Passagier. Diese Werte gilt es noch zu einer gesamten Servicerate

zusammenzufassen, die ebenfalls in Sekunden pro Passagier angegeben wird. Um die einzelnen Serviceraten zu einer zusammenzufassen, wird jede der Serviceraten (s in Formel 4.1) invertiert und addiert. Die Summe aus dieser Addition wird daraufhin erneut invertiert und ergibt so die gesamte Servicerate (siehe Formel 4.1). Mit dieser Methode wird in der Elektrotechnik der Gesamtwiderstand von parallelen Widerständen berechnet[5]. Das Prinzip der Berechnung lässt sich auch für die Ermittlung der gesamten Servicerate anwenden.

$$SR = \frac{1}{\sum_{i=1}^{k} \frac{1}{s_i}} \tag{4.1}$$

$$W = A * SR \tag{4.2}$$

Wenn die Ergebnisse des Beispiels (17s, 18s, 22s, 17s) auf die Formel 4.1 angewendet werden, ergibt sich eine *serviceRate* von 4,57 Sekunden. Die gesamte Servicerate multipliziert mit der Anzahl der Pax-Objekte in der *queue*, also der Anzahl der Passagiere in der Warteschlange (A in Formel 4.2), ergibt die Wartezeit für diesen Moment. Bei dem Beispiel ist definiert, dass sich 50 Passagiere in der Warteschlange befinden. Die hieraus resultierende Wartezeit für jene Person, die sich nun als nächstes anstellt, beträgt dann 237,5 Sekunden (50 Passagiere x 4,57 Sekunden). Diese Methode wird nach jedem Ereignis aufgerufen.

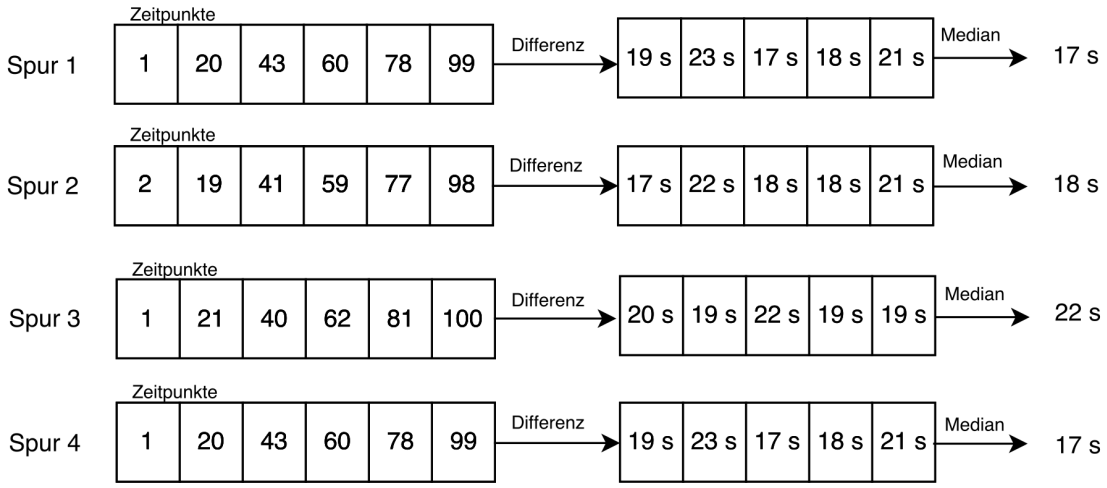

Abbildung 4.8: Skizze zur Ermittlung der Servicerate

Die Methode *clean* entfernt alle ungültigen Pax- und Counter-Objekte, welche sich noch im System befinden und wird nach jedem Ereignis aufgerufen.

[5](Elektronik Kompendium 2014)

Auswertung Die Auswertung der Ergebnisse erfolgt anhand zweier Vorgehensweisen. Dies ist zum einen die mathematische Berechnung und Bewertung der Qualität der Ergebnisse hinsichtlich unterschiedlicher Aspekte und zum anderen aus der visuellen Darstellung der Ergebnisse und der optischen Bewertung. Als Grundlage für die Konfiguration der Simulation dieser Tests wurden die Parameter an die Daten des 16. Oktobers 2015 angepasst. Die Messdaten dieses Tages wurden vom Hamburger Flughafen bereitgestellt (A.3 Abbildung A.1). Das bedeutet, dass der Plan der Kontrollspurenöffnung sowie die Ankunftsraten dieses Tages übernommen wurden.

Bei der mathematischen Auswertung werden zunächst die Aspekte definiert, die zur Bewertung der Ergebnisse herangezogen werden müssen. Hierzu gehört die durchschnittliche Differenz. Diese ergibt sich aus der Addition der Beträge der Differenzen und dessen anschließenden Division durch die Anzahl der betrachteten Zahlen. Bei dieser Art der Auswertung verlieren einzelne hohe Differenzen an Bedeutung. Die Differenz zwischen tatsächlicher und vorhergesagter IST-Wartezeit wird im Folgenden als Abweichung bezeichnet. Aus diesem Grund besteht der zweite Aspekt aus der Betrachtung der Quantität hoher Abweichungen. Neben diesen beiden Aspekten werden außerdem die höchste und die niedrigste Abweichung betrachtet, um den Rahmen der Abweichungen zu definieren. Alle hier ausgewerteten Zahlen basieren auf dem Durchschnitt von zehn Testläufen.

Die Auswertung, basierend auf der durchschnittlichen Differenz, gibt Aufschluss über die Gesamtabweichung im Durchschnitt. Die Formel 4.3 beschreibt diese Berechnung. Die durchschnittliche Differenz (D) ergibt sich aus der Summe des Betrags der Differenzen (d) und der Division durch die Anzahl der Differenzen (n). Die Betrachtung der durchschnittlichen Differenz gibt Aufschluss über die durchschnittliche Abweichung.

$$D = \sum_{i=1}^{n} \frac{|d_i|}{n} \tag{4.3}$$

Die durchschnittliche Differenz der Testläufe bei diesem Lösungsvorschlag beträgt 9.016,24 Millisekunden also ungefähr 9 Sekunden. Das bedeutet, dass die Abweichungen bei den IST-Wartezeiten aller Passagiere im Schnitt 9 Sekunden betragen haben. Zu diesem Zeitpunkt bestehen noch keine Referenzwerte, die aus anderen Lösungsvorschlägen entwickelt wurden, weswegen hier lediglich eine subjektive Bewertung vorgenommen werden kann. Aus dieser subjektiven Sicht ist das Ergebnis als äußerst positiv zu bewerten, da eine Abweichung von 9 Sekunden bei Wartezeiten von bis zu mehreren Minuten nur sehr gering auffällt.

Da sich aus diesem Ergebnis aber nur wenig über die Quantität von hohen Abweichungen ermitteln lässt, wird die Tabelle 4.1 hinzugezogen. Diese Tabelle gibt Aufschluss über die Quantität der Abweichungen.

Abweichung	Anzahl der Passagiere
0 - 1 Minute	96,65 %
1 - 2 Minuten	2,47 %
über 2 Minuten	0,98 %

Tabelle 4.1: Abweichungen in Prozent

Die Tabelle ordnet die Passagiere basierend auf der Abweichung den jeweiligen Zeiträumen zu. Die Anzahl der Passagiere wird in Prozent angegeben und basiert auf einer Gesamtzahl von circa 28.500 Passagieren. Es lässt sich ermitteln, dass 96,65 % der Passagiere eine Abweichung der Vorhersage von bis zu einer Minute erfahren haben. Bei 2,47 % der Passagiere betrug die Abweichung 1 - 2 Minuten, dies entspricht einer Anzahl von ungefähr 704 Passagieren. Bei lediglich 0,98 % der Passagiere lag die Abweichung bei über 2 Minuten, dies entspricht einer Anzahl von ungefähr 279 Passagieren. Weil die Zahlen gerundet wurden, beträgt die Prozentzahl in der Tabelle zusammengerechnet 100,1 %. Bei diesen Ergebnissen betrug die tatsächliche maximale Abweichung 5,5 Minuten (328.168 Millisekunden), die minimale Abweichung betrug dagegen nur 1 Millisekunde. Auch hier kann wieder nur nach subjektiven Gesichtspunkten bewertet werden, da keine Referenzwerte vorliegen. Aus dieser Sicht ist das Ergebnis von 96,65 % der Passagiere, die eine Abweichung von bis zu einer Minute hatten, als positiv zu bewerten. Für eine genauere Bewertung der Ergebnisse muss zunächst Aufschluss über die Ursachen dieser Abweichungen erlangt werden. Dies geschieht anhand der visuellen Auswertung.

Die Auswertung der visuellen Darstellung der Ergebnisse liefert ebenfalls Aufschluss über die maximale Abweichung, die minimale Abweichung sowie die Quantität von hohen Abweichungen. Aufgrund der übersichtlichen Darstellung lassen sich bei dieser Methode die Ursachen für Abweichungen schnell erkennen.

In dem Diagramm in Abbildung 4.9 ist für jeden Passagier, der das Warteschlangensystem durchlaufen hat, die tatsächliche Wartezeit (Rot) und die vorhergesagte Wartezeit (Schwarz) sowie die Anzahl der offenen Kontrollspuren beim Betreten des Systems (Magenta) verzeichnet. Die Y-Achse verzeichnet die Wartezeit in Sekunden und im Falle der geöffneten Kontrollspuren die Anzahl dieser. Die X-Achse gibt den Zeitparameter in Stunden an.

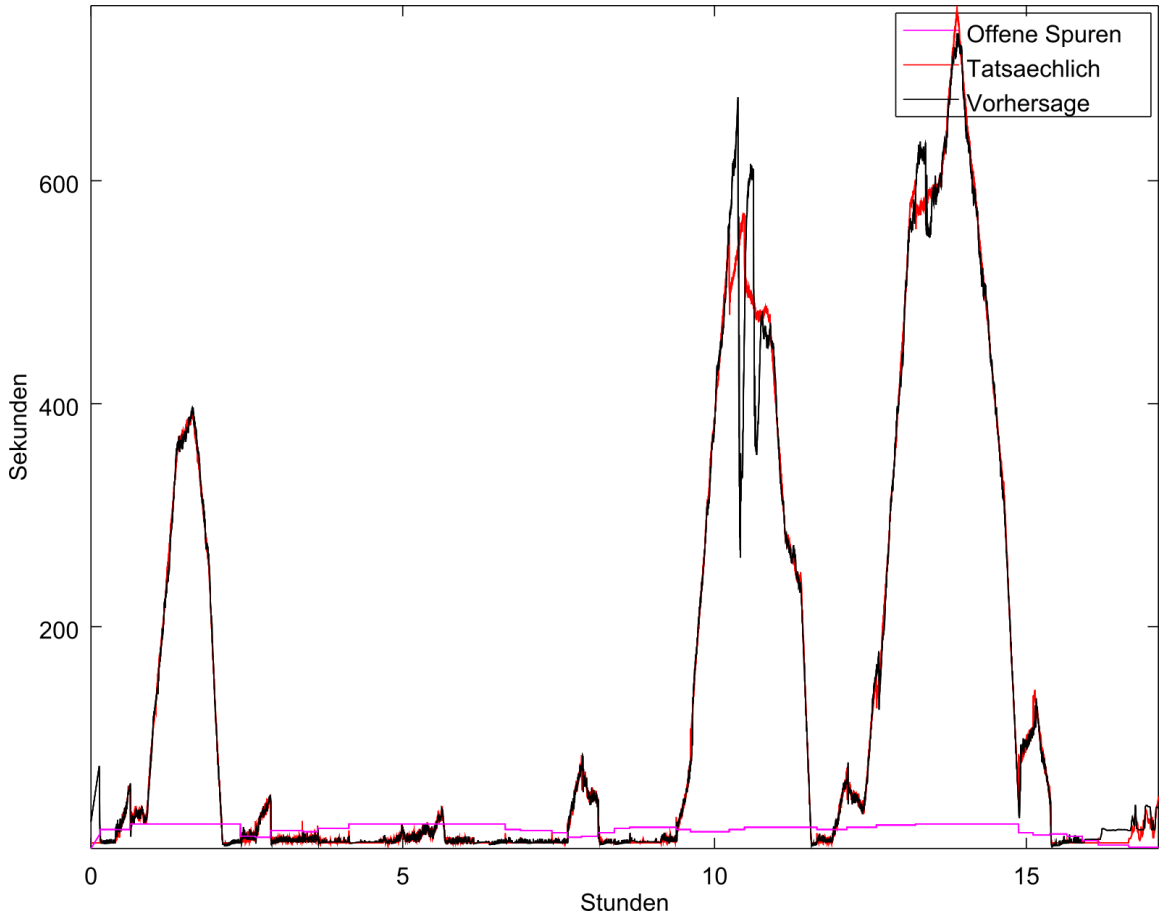

Abbildung 4.9: Ergebnisse der Lösung 1 dargestellt als Diagramm

Es wird schnell ersichtlich, dass immer eine sehr hohe Abweichung entsteht, wenn eine Kontrollspur geschlossen oder geöffnet wird. Diese Abweichung ist hinnehmbar, da keine Informationen über die zukünftige Planung der Öffnung und Schließung von Spuren vorhanden ist. Der Zusammenhang zwischen Öffnung und Schließung der Kontrollspuren und der damit verbundenen Abweichung kann anhand eines Beispiels aufschlussreich erklärt werden.

Ein Passagier betritt die Warteschlange und bekommt zu diesem Zeitpunkt die Prognose, dass die Wartezeit voraussichtlich fünf Minuten beträgt. Diese Annahme basiert auf der Anzahl der zu diesem Zeitpunkt geöffneten Kontrollspuren. Während der Passagier in der Warteschlange steht, werden zwei Kontrollspuren geschlossen und die Wartezeit erhöht sich auf sieben Minuten. Nun beträgt die tatsächliche Wartezeit, die der Passagier warten muss, sieben Minuten, die ihm vorhergesagte Wartezeit lag aber bei fünf Minuten.

In diesem Beispiel entsteht eine Abweichung von zwei Minuten. Dies ist ebenfalls erklärend dafür, wenn Kontrollspuren geöffnet werden und so eine Abweichung entsteht. Diese Abweichungen würden sich mit der Kenntnis über die bevorstehende Öffnung und Schließung von Kontrollspuren beseitigen lassen. Da diese aber nicht vorhanden sind, können die Abweichungen in diesem Projektes nicht verhindert werden.

Abschließend lässt diese Auswertung das Fazit zu, dass die Ergebnisse als positiv zu betrachten sind. Jedoch beruht diese Lösung auf dem Vorhandensein einer zu hohen Datenausbeute und ist aus diesem Grund nicht als durchführbare Option für den Hamburg Flughafen zu bewerten. Dieser Lösungsvorschlag dient als Referenz für die Bewertung des zweiten Lösungsvorschlags und spiegelt außerdem die Ergebnisse wider, die erreicht werden, wenn alle vorhandenen Daten verwendet werden.

Lösung 2

Die Motivation hinter diesem Lösungsweg besteht darin, basierend auf wenig Daten eine Prognose der IST-Wartezeit zu erstellen. Für diesen Lösungsvorschlag wird von einer Datenausbeute ausgegangen, wie sie zum Start des Projekts am Hamburger Flughafen definiert wurde. Das bedeutet, dass lediglich zwei Ticketscanner zur Erfassung des zweiten Zeitstempels installiert wurden. Diese beiden Ticketscanner werden nur optional benutzt. In dieser Arbeit nehme ich an, dass diese Ticketscanner von etwa 30 % der Passagiere, welche diesen passieren, benutzt werden. Die Herkunft dieser Annahme wurde bereits im Kapitel 2.2 erläutert.

Verwendete Daten Bei diesem Lösungsvorschlag wird eine nahezu hundertprozentige Ausbeute der ersten Zeitstempel benötigt. Für den zweiten Zeitstempel gibt es keine Vorgabe für die Höhe der Ausbeute, damit der Lösungsvorschlag funktioniert. Selbstverständlich muss die Ausbeute mehr als 0 % betragen. Dennoch wird von einer Ausbeute von 30 % je Scanner ausgegangen.

Funktion Wie bereits bei dem ersten Lösungsvorschlag, wird auch hier die vollständige Funktionalität mit der Benutzung der im Interface definierten Methoden realisiert.

Der Methode *checkin* wird beim Aufruf die *id* sowie der erste Zeitstempel (*arrivalDate*) übermittelt (Abb. 4.5). Diese Daten werden in einem neuen Objekt der Klasse *Pax* in der Liste *data* abgespeichert. In diesem Zusammenhang fungiert die Liste nicht als Warteschlange, so wie es bei dem ersten Lösungsweg der Fall ist, sondern wird lediglich als Speicher für die Pax-Objekte verwendet. In diesem Lösungsvorschlag wird das *timeout* Attribut nicht verwendet, weil davon ausgegangen wird, dass maximal 30 % der Passagiere den zweiten Zeitstempel abgeben. Die Verwendung eines Timeouts würde hier die restlichen 70 % Pax-Objekte fälschlicherweise aus der Liste entfernen.

Der Methode *checkout* werden beim Aufruf die *id*, der zweite Zeitstempel (*serviceStartDate*) sowie die Kontrollspurenkennung (*counter*) übermittelt. Die Methode ermittelt den zu der *id* passenden Eintrag aus der Liste *data* und fügt diesem den zweiten Zeitstempel (*serviceStartDate*) und die Kontrollspurenkennung (*counter*) hinzu.

Die Methode *update* beinhaltet die Logik der Berechnung und wird nur dann ausgeführt, wenn unmittelbar zuvor ein *checkout* Ereignis aktiviert wurde. Zunächst werden zwei Pax-Objekte aus der Liste *data* ausgewählt. In Listing 4.3 wird diese Aktion als Schritt 1 betitelt. Diese Pax-Objekte werden auf Basis eines vordefinierten zeitlichen Abstandes ausgewählt, dieser wird im Folgenden als x (in Listing 4.3 ist dieser *consideredPeriod*) bezeichnet. Der zeitliche Abstand bezieht sich auf den zweiten Zeitstempel (*serviceStartDate*) der Pax-Objekte. Diese müssen den zeitlichen Abstand zueinander haben, um die Schwankung der Ergebnisse zu reduzieren. Andernfalls würde die ermittelte IST-Wartezeit bei einer hohen Datenausbeute großen Schwankungen unterliegen, weil dann immer zwei aufeinander folgende Pax-Objekte als Referenz benutzt werden. Dies wird durch das Einhalten des zeitlichen Abstandes verhindert, da meistens mehrere Pax-Objekte als Referenz benutzt werden. Das erste Objekt, welches ausgewählt wird, ist das neuste vollständige Pax-Objekt und wird im Folgenden als *Objekt A* betitelt. Ein Objekt, welches sowohl den ersten als auch den zweiten Zeitstempel besitzt, wird als vollständiges Objekt bezeichnet. Das zweite Objekt, *Objekt B*, ist das erste vollständige Objekt, dessen zweiter Zeitstempel länger als x Minuten zurückliegt. Wenn es kein vollständiges Objekt gibt, welches länger als x Minuten in der Vergangenheit liegt, wird das älteste Pax-Objekt der Liste entnommen. Zur Berechnung stehen jetzt also das *Objekt A*, das *Objekt B* sowie alle Objekte, die zwischen *Objekt A* und *Objekt B* die Warteschlange betreten haben, zur Verfügung. Da von einem FIFO System ausgegangen wird, nehmen wir an, dass alle Objekte, die zwischen *Objekt A* und *Objekt B* der Warteschlange beigetreten sind, diese auch wieder verlassen haben. Die Servicerate ergibt sich, indem die Anzahl der ermittelten Passagiere (*Objekt A* und alle Objekte dazwischen) durch die Zeitspanne zwischen *Objekt A* und *Objekt B* dividiert wird (Schritt 2 in Listing 4.3). Diese Servicerate wird mit der Anzahl der Passagiere multipliziert, die die Warteschlange nach Objekt A betreten haben, das heißt mit allen Passagieren, die sich noch in der Warteschlange befinden (Schritt 3 in Listing 4.3).

Listing 4.3: Die Methode *update*

```
public Predictor update(Long time) {
/* Schritt 1.
 * Ermittlung der Objekte A und B
 * Die Methode getLastCompleteBefore() sucht den Index für
 * das nächste vollständige Objekt vor dem Index, der beim
 * Aufruf übergeben wird.
 */
```

```
 Integer a = getLastCompleteBefore(data.size()-1);
 Integer b = getLastCompleteBefore(a);
 if(a != -1 && b != -1){
  //Suche so lange Objekt-B bist der zeitliche Abstand
  //(consideredPeriod) gegeben ist
  while(data.get(a).serviceStartDate -
     data.get(b).serviceStartDate < consideredPeriod){
   Integer tmp = getLastCompleteBefore(b);
   if(tmp != -1){
        b = tmp;
   }else{
        break;
   }
  }
  Pax objectA = data.get(a);
  Pax objectB = data.get(b);
 /*Schritt 2.
  *Berechnung der Servicerate
  */
  Double rate = (objectA.serviceStartDate.doubleValue() -
  objectB.serviceStartDate.doubleValue()) / ((double) a -
     (double) b);
 /*Schritt 3.
  *Berechnung der Wartezeit
  */
  wt = ((((double)data.size())-a))*rate;
 }
return this;
}
```

Die Methode *clean* wird in diesem Zusammenhang nicht benötigt, da die Objekte keinen Timeout besitzen.

Auswertung Die Auswertung dieser Lösung wird nach demselben Schema durchgeführt, wie die der ersten Lösung. Das bedeutet, dass die Ergebnisse dieses Lösungungsweges sowohl mathematisch als auch visuell bewertet werden. Die mathematische Bewertung besteht aus der Ermittlung der durchschnittlichen Differenz zwischen der tatsächlichen und der vorhergesagten Wartezeit sowie einer Betrachtung der Quantität dieser Differenzen. Diese Differenz wird im Folgenden als Abweichung beschrieben. Zusätzlich wird die kleinste und die größte Abweichung betrachtet, um den Rahmen der Abweichungen zu definieren.

Die visuelle Bewertung beruht auf der Darstellung der Ergebnisse als Liniendiagramm. In diesem werden Fehlerquellen ermittelt sowie die Quantität der Abweichungen betrachtet. Alle hier ausgewerteten Zahlen basieren auf dem Durchschnitt von zehn Testläufen. Die Konfiguration des Simulators beruht auf den Daten des 16. Oktober 2015 (A.3 Abbildung A.1).

Zunächst wird die durchschnittliche Differenz dieser Lösung begutachtet. Da der Lösungsweg keine fest definierte Informationsausbeute benötigt, werden diese Ergebnisse mit einer Vielzahl an möglichen Fällen getestet. Ein Fall besteht aus einer Anzahl an installierten Ticketscannern sowie die zugehörige Ausbeute dieser.

Der Tabelle 4.2 sind die durchschnittlichen Differenzen einer Auswahl an Fällen zu entnehmen. Im Kapitel A.2 Tabelle A.1 befindet sich eine ausführliche Tabelle, in der für jede mögliche Anzahl an geöffneten Kontrollspuren die durchschnittliche Differenz dargestellt ist. Diese Tabelle bietet Aufschluss, inwiefern sich die Ergebnisse bei einer höheren Datenausbeute verhalten.

In der horizontalen Spalte sind in dieser Tabelle die unterschiedlichen Ausbeuten in Prozent angegeben, in der vertikalen Spalte ist die Anzahl der verwendeten T2-Ticketscanner verzeichnet. Die Abweichungen in dieser Tabelle sind in Millisekunden angegeben.

	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	27.383	19.064	13.739	12.651	10.897	10.168	10.074	9.436	9.185	9.044
2	20.226	12.631	11.802	10.275	9.560	9.635	9.259	9.207	9.108	9.056
3	14.137	10.529	9.130	8.787	8.469	8.407	8.217	8.195	8.116	8.125
4	11.294	9.805	8.967	8.674	8.892	8.498	8.551	8.413	8.359	8.359
7	9.877	8.785	8.518	8.310	8.326	8.208	8.167	8.196	8.206	8.170
12	8.954	8.321	8.287	8.189	8.171	8.190	8.123	8.125	8.116	8.115
16	9.730	9.301	9.190	9.255	9.242	9.195	9.161	9.136	9.147	9.142
20	9.119	8.682	8.707	8.661	8.638	8.644	8.594	8.592	8.581	8.573
24	8.708	8.515	8.421	8.407	8.330	8.358	8.335	8.310	8.315	8.299

Tabelle 4.2: Durchschnittliche Differenzen

So ist zunächst zu erkennen, dass bei einer Ausbeute von 100 % nur geringe Verbesserungen zwischen einem oder 24 installierten Ticketscannern zu verzeichnen sind. Hier bildet sich bereits ein signifikanter Vorteil gegenüber der *Lösung 1* ab. Es lässt sich bereits mit einem Ticketscanner, dessen Nutzung obligatorisch ist, ein akzeptables Ergebnis ermitteln. Dieses ist von vergleichbarer Qualität mit dem Ergebnis der *Lösung 1.*

Ab einer Anzahl von sieben Ticketscannern nähert sich das Ergebnis bereits dem Maximum an Genauigkeit an. Weder mit der Installation zusätzlicher Ticketscanner noch mit der obligatorischen Nutzung der Ticketscanner lassen sich hier die Ergebnisse signifikant verbessern. So lässt sich ab einer Anzahl von sieben Ticketscannern das Ergebnis nur noch um maximal eine Sekunde verbessern. Dies spiegelt ebenfalls einen bedeutsamen Vorteil zur *Lösung 1* wider. Wenn die obligatorische Nutzung der Ticketscanner nicht durchsetzbar ist, lassen sich bereits ab sieben Ticketscannern und ab einer Ausbeute von 10 % Ergebnisse mit einer vergleichbaren Qualität zur *Lösung 1* ermitteln.

Aus Gründen der Übersicht werden in der weiteren Auswertung lediglich drei Fälle begutachtet.

- Der erste Fall spiegelt die Situation wieder, dass lediglich ein T2-Ticketscanner installiert wurde, der eine Ausbeute von 10 % hat. Diese Situation soll den Worstcase widerspiegeln.
- Der zweite Fall bildet die derzeitige Situation am Hamburg Flughafen ab. Das bedeutet, dass zwei T2-Ticketscanner installiert sind, die jeweils eine Ausbeute von 30 % haben.
- Der dritte Fall simuliert den Bestcase, also die Situation mit der größtmöglichen Datenausbeute. In diesem Fall sind 24 T2-Ticketscanner installiert, dessen Nutzung obligatorisch für jeden Passagier ist (100 % Ausbeute).

Zur Bewertung der Quantität von hohen Abweichungen wird auch in dieser Auswertung wieder eine Tabelle (Tabelle 4.3) erstellt, welche die Abweichungen in drei Kategorien einteilt. In der ersten Kategorie beträgt die Abweichung weniger als eine Minute. In der zweiten Kategorie befindet sich die Abweichung zwischen einer und zwei Minuten. In der dritten Kategorie beträgt die Abweichung über zwei Minuten. Die Gesamtanzahl der Passagiere auf der die Prozentangaben basieren, beträgt circa 28.500.

	0-1 Minute	1-2 Minuten	über 2 Minuten
Fall 1 (Worstcase)	84,65 %	10,05 %	5,30 %
Fall 2 (Ist-Zustand)	96,88 %	2,65 %	0,46 %
Fall 3 (Bestcase)	96,93 %	2,80 %	0,27 %

Tabelle 4.3: Darstellung der Quantität der Abweichungen

Aufgrund dieser Tabelle kann die Aussage gemacht werden, dass sich hohe Abweichungen bereits ab dem Einsatz von zwei T2-Ticketscannern und einer Ausbeute von 30 % enorm reduzieren lassen. Eine derartige Verbesserung, wie von Fall 1 zu Fall 2 ,kann jedoch nicht ein weiteres Mal durch die Erhöhung von Ticketscannern und Ausbeute erzielt werden. So lassen sich von Fall 2 zu Fall 3 im Bereich von 0 - 1

Minute Abweichung lediglich 0,05 % Verbesserung erzielen. Das entspricht in diesem Testfall einer Anzahl von etwa 15 Passagieren. Im Bereich von 1 - 2 Minuten Abweichung lässt sich von Fall 2 zu Fall 3 eine Verbesserung von 0,15 % erzielen. Das entspricht einer Anzahl von etwa 43 Passagieren. Im Bereich von über zwei Minuten lässt sich hier jedoch eine Optimierung von 0,19 % erzielen was einer Anzahl von 55 Passagieren entspricht.

Die Quantität von hohen Abweichungen wird bereits ab einer Anzahl von zwei installierten T2-Ticketscannern signifikant verringert und erreicht damit bereits ab Fall 2 eine nur geringfügig schlechtere Qualität im Vergleich mit den Ergebnissen der *Lösung 1*.

Bezogen auf die mathematische Auswertung lässt sich feststellen, dass die *Lösung 2* wesentliche Vorteile gegenüber der *Lösung 1* hat und bereits bei einer geringen Datenausbeute recht genaue Wartezeiten ermittelt. So lassen sich bereits ab einer Anzahl von sieben Ticketscannern und ab einer Ausbeute von 10 % Ergebnisse von vergleichbarer Qualität der *Lösung 1* erzielen.

Neben der mathematischen Auswertung geben die Diagramme für die visuelle Auswertung eine Übersicht über etwaige Fehlerquellen, da sich hier der Kontext leicht überschauen lässt. Dabei werden auch nur die drei oben beschriebenen Fälle betrachtet.

Die X-Achse der Diagramme in den Abbildungen 4.10, 4.11 und 4.12 beschreibt die vorangeschrittene Zeit in Stunden und die Y-Achse beschreibt die Wartezeit in Sekunden. Im Falle der geöffneten Kontrollspuren (Magenta) beschreibt die Y-Achse die Anzahl.

In Abbildung 4.10 ist der erste Fall dargestellt. Es ist zu erkennen, dass die IST-Wartezeit (Schwarz) nur selten aktualisiert wird. Dieses ist der Grund für eine Vielzahl von Abweichungen. Da die IST-Wartezeit nur dann aktualisiert wird, wenn ein Passagier seinen zweiten Zeitstempel abgibt, ist die Lösung für dieses Problem eine höhere Datenausbeute.

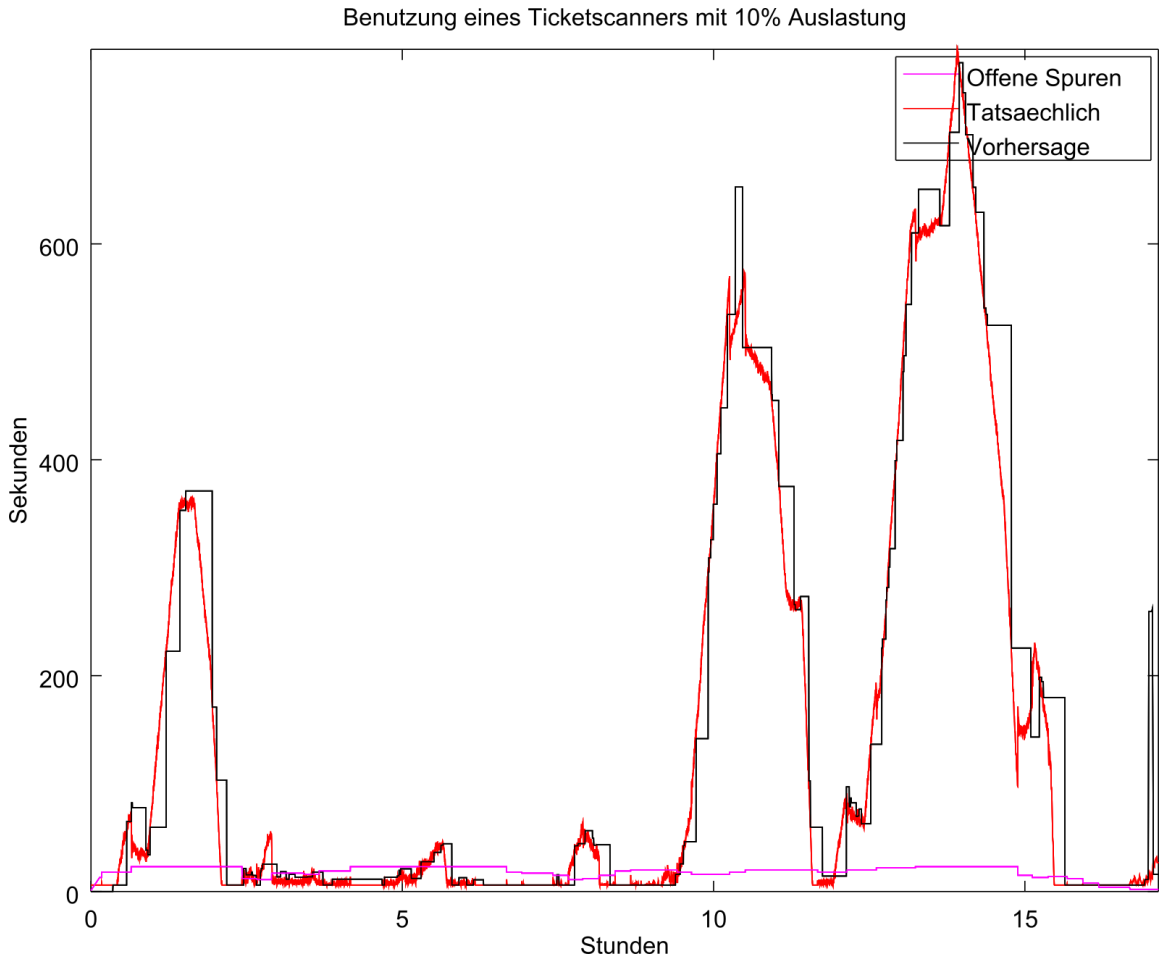

Abbildung 4.10: Fall 1

In dem zweiten Fall, welcher in der Abbildung 4.11 dargestellt ist, lässt sich bereits eine eindeutig detaillierte IST-Wartezeiten Vorhersage erkennen. Die Datenausbeute ist höher als im Fall 1, was in der erheblich häufigeren Aktualisierung der IST-Wartezeiten Prognose (Schwarz) zu erkennen ist. Dieses reduziert die Abweichungen signifikant. Aufgrund der höheren Anzahl an IST-Wartezeit Prognosen kristallisiert sich ein weiteres Problem heraus. Im höheren Bereich der X-Achse bei etwa 10 Stunden und 13 Stunden sind starke Abweichungen gegenüber der tatsächlichen Wartezeit zu verzeichnen. Dasselbe Problem ist bereits bei *Lösung 1* aufgetreten. Die Ursache hierfür wurde dort im Abschnitt 4.3.1 (Auswertug) anhand eines Beispiels beschrieben. Eine Lösung dieses Problems ist mit den zur Verfügung stehenden Daten nicht möglich.

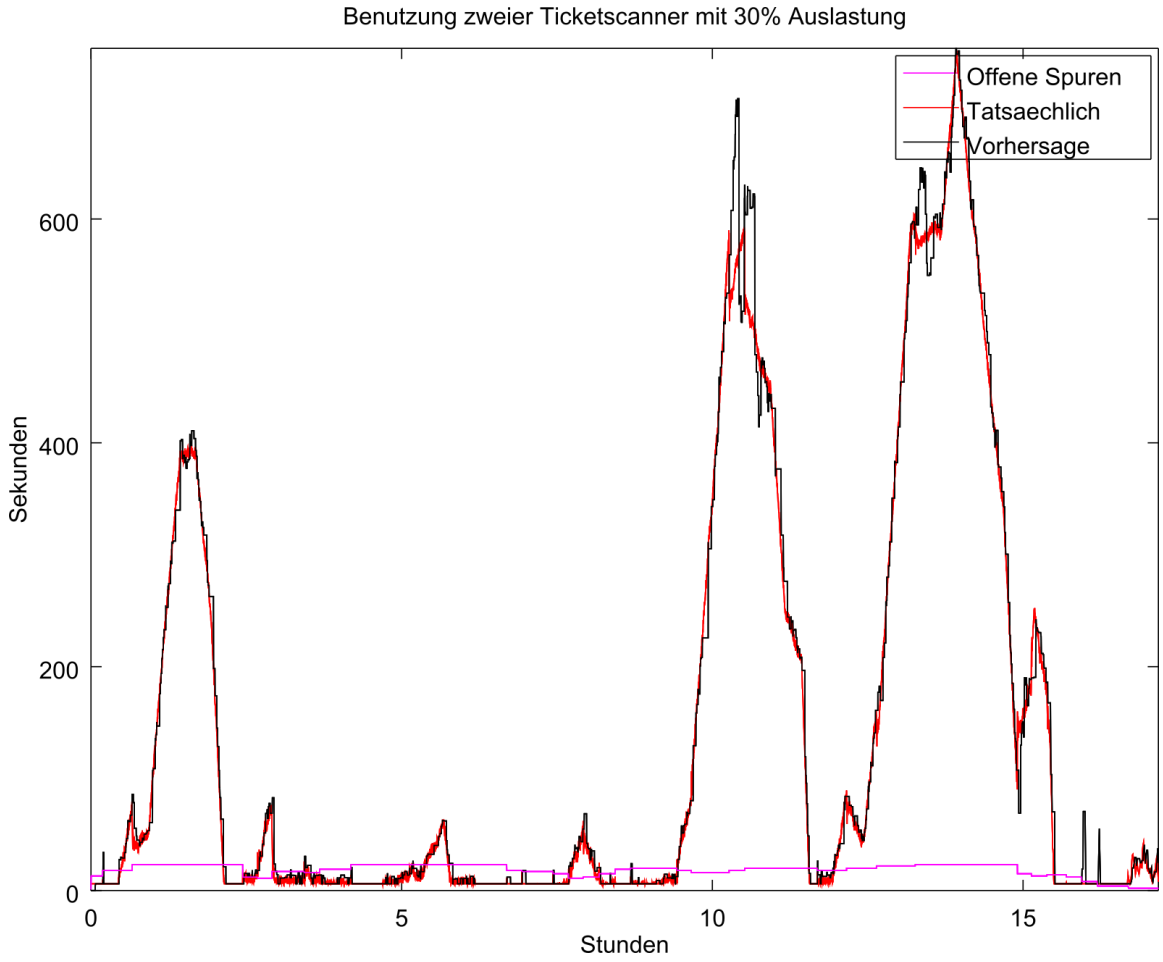

Abbildung 4.11: Fall 2

Der in der Abbildung 4.12 beschriebene dritte Fall unterscheidet sich in der Visualisierung nur geringfügig von dem zweiten Fall. Es ist zu erkennen, dass bei einer hohen Datenausbeute die Anzahl der IST-Wartezeitenprognosen der Anzahl der abgefertigten Passagiere entspricht. Aus diesem Grund wird die IST-Wartezeiten Vorhersage häufiger ausgeführt und hat eine höhere Genauigkeit als im zweiten Fall. Die Vorhersage deckt sich in den meisten Fällen mit der tatsächlichen Wartezeit. Auch hier ist wieder das Problem beim Wechsel der Kontrollspuren-Anzahl zu verzeichnen, für das im Rahmen dieses Projektes keine Lösung entwickelt werden konnte.

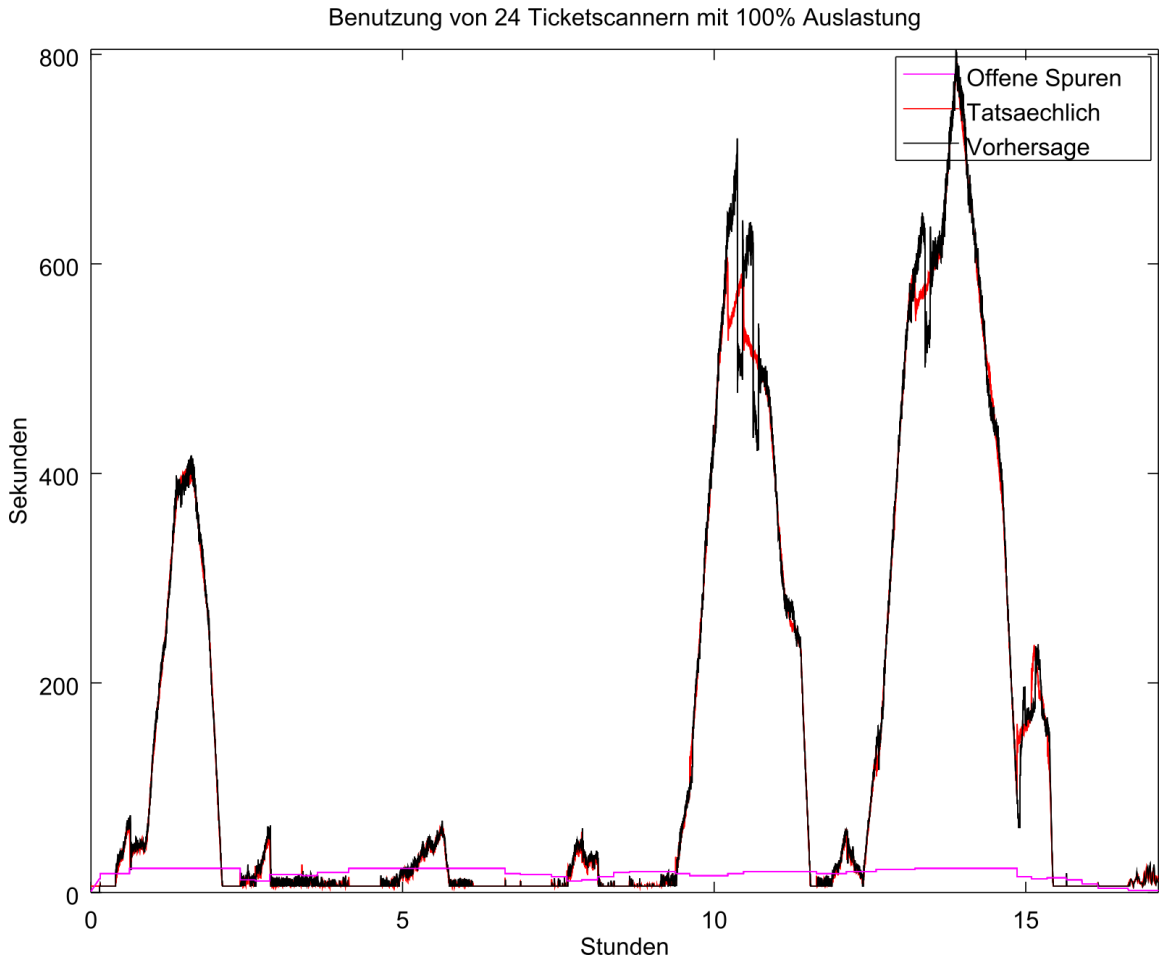

Abbildung 4.12: Fall 3

Mithilfe der visuellen Auswertung konnten die Ursachen der hohen Abweichungen ermittelt werden. Diese sind zum einen dem Wechsel der Anzahl der geöffneten Kontrollspuren und zum anderen einer mangelnden Datenausbeute geschuldet. Jedoch sind auch hier ab einer Ausbeute von 30 % und einer Anzahl von zwei T2-Ticketscannern verwendbare Prognosen zu erreichen.

Abschließend lässt diese Auswertung das Urteil zu, dass *Lösung 2* enorme Vorteile gegenüber *Lösung 1* bietet. So kann diese mit einer beliebigen Anzahl an Ticketscannern verwendet werden und liefert selbst bei einer geringen Datenausbeute verwendbare Ergebnisse. Für die Verbesserung von Ergebnissen muss lediglich die Datenausbeute oder die Anzahl der T2-Ticketscanner erhöht werden, ohne dass der Algorithmus angepasst werden muss. Ein Nachteil gegenüber der *Lösung 1* ist das sensible Verhalten, wenn das *FIFO* Prinzip nicht eingehalten wird. Dieses kann jedoch mit der Wahl einer größeren Zeitspanne für die Berechnung ausgeglichen werden. Die Vorteile überwiegen jedoch und bieten eine Lösung mit einer skalierbaren Ergebnisgenauigkeit für das Problem der IST-Wartezeiten Berechnung.

4.3.2 Ermittlung der WIRD-Wartezeit

Für die Ermittlung der WIRD-Wartezeit gibt es bereits Prognosen, die in Form von Passagieraufkommen pro 15 Minuten bereitgestellt werden.

Damit diese Prognosen in eine Wartezeit umgerechnet werden können, werden noch weitere Werte benötigt. Dies ist zum einen die Servicerate der Kontrollspuren und zum anderen, wie viele Kontrollspuren geöffnet sind. Für diese Daten bestehen allerdings keine Prognosen und müssen daher zunächst ermittelt werden.

Da für die fehlenden Werte keine Prognosen erstellt wurden, müssen diese anhand von Richtwerten ermittelt werden. Der Flughafen Hamburg hat für diese Ermittlungen den Richtwert für die Geschwindigkeit der Kontrollspuren preisgegeben. So wird angenommen, dass eine Kontrollspur 120 Passagiere pro Stunde abfertigen kann. Die Planung, die der Hamburger Flughafen für die Kontrollspurenöffnung erstellt, basiert ebenfalls auf dieser Annahme. Deswegen lassen sich mithilfe dieses Richtwerts Rückschlüsse auf die Anzahl der geöffneten Kontrollspuren ziehen. Die Anzahl der erforderlichen Kontrollspuren kann ermittelt werden, indem man, basierend auf der Prognose und der Annahme, dass eine Kontrollspur 120 Passagiere pro Stunde abfertigen kann, eine Aussage trifft. Die Anzahl der eröffneten Kontrollspuren ergibt sich, indem man die Anzahl der erwarteten Passagiere durch die Anzahl der in diesem Zeitraum abgefertigten dividiert. Wenn also zum Beispiel 700 Passagiere in einem Zeitraum von 15 Minuten erwartet werden und eine Kontrollspur 30 Passagiere (120 Passagiere / 4) in dieser Zeit abfertigen kann, so ergibt sich eine Anzahl von 23,33 benötigten Kontrollspuren (700 Passagiere / 30 Passagiere pro 15 Minuten). Es können maximal 24 geöffnete Kontrollspuren erwartet werden. Die Ermittlung der nötigen geöffneten Kontrollspuren wird nicht mit dieser Präzision durchgeführt, sondern mit einer Toleranz von +/-50 Passagieren. Wenn also 724 Passagiere erwartet werden, wird mit einer gerundeten erwarteten Anzahl von 700 Passagieren gerechnet.

Mit der Anzahl der geöffneten Spuren und dem Richtwert für die Abfertigungs-Geschwindigkeit pro Stunde kann die Servicerate ermittelt werden. Aufgrund des Richtwertes ergibt sich eine Zeit von 30 Sekunden pro Passagier (60*60s / 120 Passagiere). Wird diese Zeit durch die Anzahl der geöffneten Kontrollspuren geteilt, resultiert die gesamte Servicerate in Sekunden pro Passagier. Wenn diese mit der Anzahl der zu erwartenden Passagiere multipliziert wird, ergibt sich der gesamte Zeitaufwand für die erwarteten Passagiere. Zieht man von diesem Zeitaufwand den betrachteten Zeitraum ab, erhält man den Zeitüberschuss, welcher der Wartezeit nach diesem Zeitraum entspricht. Dieser Zeitüberschuss muss auf den erwarteten Zeitüberschuss der darauffolgenden Zeiträume addiert werden.

Diese Berechnung wird für den geforderten Tag in 15 Minuten Zeiträumen getätigt. Das Ergebnis hieraus ist eine Vorhersage über die voraussichtlichen Wartezeiten in 15 Minuten Abständen.

4.3.3 Zusammenfassung

In diesem Kapitel wurde die Ermittlung der IST-Wartezeit beschrieben und erklärt. Die triviale Lösung für die Berechnung der WIRD-Wartezeit wurde hier ebenfalls deutlich gemacht. Auf Basis der Auswertung resultiert die *Lösung 2* als geeignete Option für die Berechnung der IST-Wartezeiten am Hamburger Flughafen. Diese bietet immense Vorteile gegenüber der *Lösung 1*. Diese Vorteile ergeben sich aus der Verwertung von weniger Daten und der damit verbundenen geringeren Fehleranfälligkeit, welche sich in Form von geringeren Abweichungen äußert. Des Weiteren besteht bei der *Lösung 2* nicht die dringliche Notwendigkeit der Installation vieler T2-Ticketscanner. Es ist jedoch möglich, beliebig viele Ticketscanner dem System hinzuzufügen. Für Ergebnisse, die der Qualität von *Lösung 1* ähnlich sind, reicht bereits die derzeitig installierte Hardware (Ticketscanner). Voraussetzung hierfür ist die permanente Öffnung der Kontrollspuren, an denen die Ticketscanner installiert sind sowie die Ausbeute von 30 % der Passagiere.

4.4 Der Server

Der auf dem Server installierte Anwendungsserver ist notwendig für die Ausführung der entwickelten Algorithmen. Der Begriff Anwendungsserver[6] beschreibt eine Serverumgebung auf der Programme ausgeführt werden können. Des Weiteren bietet die Verwendung eines Anwendungsservers die Möglichkeit, die ermittelten Daten direkt via REST-Schnittstelle bereit zu stellen. Da der Fokus dieses Projektes auf der Entwicklung des Simulators und die Berechnung der IST-Wartezeiten liegt, wird dieses Kapitel nicht in dem Detailgrad der vorherigen Kapitel beschrieben.

[6](Barry, Douglas K. 2016)

4.4.1 Technologie

Für die Realisierung der Serverumgebung wurde die JBoss Enterprise Appllication[7] Plattform verwendet. Mithilfe dieser Software besteht die Möglichkeit, Java Anwendungen auf einem Server auszuführen. Die Wahl der Software ist hier so ausgefallen, weil diese bereits auf den Serversystemen des Hamburger Flughafens eingerichtet ist. Der Vorteil des Anwendungsservers ist, dass die Berechnungen zentral ausgeführt werden und ohne weitere Umwege an die REST-Schnittstelle überliefert werden. Zur Realisierung der REST-Schnittstelle wird das Plug-In RESTEasy[8] benutzt. Mithilfe von RESTEasy können die Schnittstellen definiert und bereitgestellt werden. Dieses Plug-In ist für den JBoss-Server entwickelt und funktioniert daher einwandfrei mit diesem.

4.4.2 REST-Schnittstelle

REST ist die Kurzform für Representational State Transfer und beschreibt eine Methode der Datenübermittlung. Anstatt Daten mittels SQL-Befehlen oder sonstigen Protokollen zu ermitteln, werden hier die Daten anhand fester URLs in Formaten wie JSON oder XML bereitgestellt. Jede Ressource muss über eine fest definierte URL erreicht werden, diese Eigenschaft wird als Adressierbarkeit betitelt. Um Daten von einer REST-Schnittstelle zu erhalten, muss keine Sitzung oder ein anderer vergleichbarer Aufwand gestartet werden. Es muss lediglich eine Anfrage geschickt werden. Diese Eigenschaft wird als Zustandslosigkeit betitelt. Auf die Schnittstelle muss mittels der http-Methoden GET, POST, PUT oder DELETE zugegriffen werden [9].

Die Struktur der REST-Schnittstelle ist das wichtigste Element. Zunächst wird definiert, welche Ressourcen bereitgestellt werden müssen, damit im Anschluss jeder Ressource eine eindeutige Adresse zugewiesen werden kann. Diese sind zum einen die IST-Wartezeit und zum anderen die WIRD-Wartezeit.

Die Anwendung soll immer die IST-Wartezeit anzeigen. Neben diesem Wert werden außerdem die IST-Wartezeiten der vergangenen zwei Stunden angezeigt. Auf Basis dieser Anforderungen wird eine Schnittstelle definiert, die eine Liste der Wartezeiten der vergangenen zwei Stunden, basierend auf dem bei der Anfrage angegebenen Zeitparameter, ausgibt.

Für die Angabe der WIRD-Wartezeiten werden immer, mit Ausnahme des aktuellen Tages, alle Werte eines Tages benötigt. Aus diesem Grund stellt die Schnittstelle eine Liste mit WIRD-Wartezeiten zur Verfügung, die alle WIRD-Wartezeiten des zum angegebenen Zeitparameter passenden Tages liefert.

[7](JBoss 2016)
[8](RESTEasy 2015)
[9](Rodriguez, Alex 2015)

4.4.3 Umsetzung

Genau wie bei der Simulation muss auch auf dem Server zur Berechnung der IST-Wartezeit ein Ereignis übermittelt werden. Zu diesem Zweck wird jede Sekunde eine Datenbankabfrage gestartet. Das Ergebnis wird auf Änderungen zur vorherigen Abfrage überprüft. Alle Änderungen werden analysiert und in die gegebenen Ereignisse umgewandelt. Falls in dieser Sekunde mehrere Events erkannt wurden, werden diese chronologisch geordnet und anschließend an den Algorithmus zur Berechnung weitergegeben. Die ermittelten Zahlen werden in einer Liste gespeichert und chronologisch geordnet, sodass jederzeit auf die aktuelle IST-Wartezeit sowie die vergangenen Wartezeiten zugegriffen werden kann. Dieser Vorgang wird dauerhaft auf dem Anwendungsserver ausgeführt und kann nur manuell gestoppt werden.

Die Berechnung der WIRD-Wartezeit wird nicht dauerhaft ausgeführt. Diese wird für jede Anfrage separat ausgeführt. Hier wird die Prognose des Passagieraufkommens aus der Datenbank gelesen und anschließend umgerechnet. Diese Berechnung wird immer für den bei der Anfrage mitgelieferten Zeitstempel erstellt und anschließend der REST-Schnittstelle übermittelt.

Die hier ermittelten Daten werden in der im Abschnitt 4.4.2 definierten Struktur im JSON Format ausgegeben.

Der so eingerichtete Anwendungsserver führt permanent den Algorithmus zur Berechnung der IST-Wartezeit aus. Die Ergebnisse dieser Berechnung werden dann via REST-Schnittstelle zur Verfügung gestellt. Die WIRD-Wartezeit wird immer dann ermittelt und bereitgestellt, wenn diese via REST-Schnittstelle abgefragt wird. Die bereitgestellten Daten können dann unter anderem von der mobilen Anwendung abgerufen werden. Aber auch weitere Systeme haben die Möglichkeit, die berechneten Zahlen über die REST-Schnittstelle abzurufen.

4.5 Die Anwendung

Die Anwendung fungiert als Anzeige der ermittelten IST- und WIRD-Wartezeiten und soll eine übersichtliche Darstellung dieser Daten bereitstellen. Der Benutzer soll schnell und in wenigen Schritten die gewünschte WIRD-Wartezeit angezeigt bekommen. Die IST-Wartezeit soll in dieser Anwendung permanent angezeigt werden. Die Anwendung soll immer aktuell sein und zu keinem Zeitpunkt veraltete Werte anzeigen.

Als Zielgruppe der Anwendung wurden die Mitarbeiter des Hamburger Flughafens festgelegt. Aus diesem Grund wird auf eine Bedienungsanleitung innerhalb der Anwendung verzichtet und Grundkenntnisse in der Bedienung werden vorausgesetzt.

In diesem Kapitel wird die Technologiewahl bezogen auf die Anwendung erörtert. Des Weiteren wird dessen Konzeption und Implementation kurz beschrieben. Auch dieses Kapitel wird nicht in dem Detailgrad wie die Berechnung der IST-Wartezeit oder die Erstellung des Simulators beschrieben.

4.5.1 Technologie

Die Auswahl der benutzten Software basiert auf den Vorgaben des Hamburger Flughafen. Aus diesem Grund wird hier auf eine detaillierte Abwegung von Alternativen verzichtet.

Aufgrund der Forderung, dass die Anwendung plattformunabhängig sein soll, wurde als Framework *Cordova* ausgewählt. *Cordova*[10] ist ein Webframework zur Erstellung von hybriden Anwendungen für mobile Endgeräte. Dieses Framework gewährt es, Anwendungen, die mit Hilfe von JavaScript erstellt wurden, auf mobilen Endgeräten abzuspielen, sodass diese das Gefühl einer nativen Anwendung vermitteln.

Für die Struktur der Anwendung wurde das Framework *AngularJS*[11] verwendet. AngularJS kann als Model View Controller (MVC) Framework verwendet werden. Mittels des MVC Musters werden Logik, Darstellung und Daten getrennt behandelt. Dies bietet einen übersichtlichen Code und eine strukturierte Aufteilung der einzelnen Elemente und bietet damit weiteren Entwicklern einen einfachen Einstieg in die Programmstrukturen.

Für die Elemente der Benutzeroberfläche fiel die Wahl auf das UI-Framework Twitter *Bootstrap*[12]. Hier gibt es eine Vielzahl an vorgefertigten Elementen für die Benutzeroberfläche. Twitter Bootstrap wurde verwendet, damit der Fokus auf die Funktionalität der Anwendung gelegt werden konnte.
Mithilfe der *Flot*[13] Bibliothek werden die Diagramme realisiert. Flot stellt Funktionen zur Verfügung, mithilfe derer Daten als Diagramm dargestellt werden können. Sie ermöglicht ebenfalls Interaktionen mit dem Diagramm zu realisieren. Die Eigenschaft der Interaktion ist notwendig, um auf einfache Art und Weise Werte aus dem Diagramm zu entnehmen.

Die *JQuery*[14] Bibliothek, welche auf JavaScript basiert, stellt Funktionen zur Manipulation und Modifizierung von Elementen der Benutzeroberfläche zur Verfügung. Außerdem lassen sich mit ihr vereinfacht AJAX Befehle umsetzen. Dieses ermög-

[10](Apache 2015)
[11](AngularJS 2016)
[12](Bootstrap 2016)
[13](Flotcharts 2016)
[14](jQuery 2015)

licht zum Beispiel eine vereinfachte Syntax zur Kommunikation mit einer REST-Schnittstelle.

4.5.2 Umsetzung

Die Anforderungen an die Anwendung wurden zum Projektstart mittels User Stories und Mockup-Bildern definiert. User Stories geben die Anforderungen in Alltagssprache definierten kurzen Sätzen wieder [15]. Dieses ermöglicht eine verständliche Definition der Anforderungen sowohl für Entwickler als auch für Kunden. Des Weiteren kann anhand von User Stories der Arbeitsaufwand sehr präzise definiert werden. Im Kapitel A.5 befinden alle definierten User Stories. Mockup-Bilder sind Skizzierungen der Benutzeroberfläche. Im Kapitel A.4 in der Abbildung A.2 befinden sich die erstellten Mockups.

Die Struktur der Anwendung wird mithilfe von AngularJS anhand des Model View Controller Musters umgesetzt. Dieses teilt den Quellcode in unterschiedliche Aufgabengebiete ein. Die Abbildung 4.13 verleiht einen Überblick über die Funktionsweise eines Model View Controller Systems.

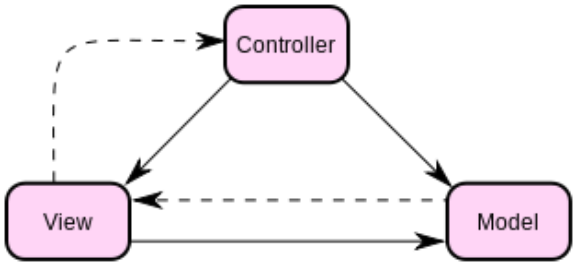

Abbildung 4.13: Model View Controller Prinzip (diese Grafik stammt von der Internetseite Wikipedia[16])

Ein Model ist in diesem Fall eine Klasse, welche die Parameter für die anzuzeigenden Daten enthält. Es ist also ein Datenspeicher, der ebenfalls Funktionen zur Modifizierung dieser Daten anbietet. In diesem Falle gibt es nur zwei Models. Ein Model beinhaltet die von der REST-Schnittstelle bereitgestellte IST-Wartezeit. Das zweite Model beinhaltet die von der REST-Schnittstelle bereitgestellte WIRD-Wartezeit.

Ein View (Ansicht) ist hier eine HTML-Datei, welche enthaltene Daten ausgibt und die Interaktionen des Benutzers an den Controller weitergibt. Es ist lediglich das Gerüst, dass die Daten wiedergibt, die es von dem Controller erhält oder aber die

[15](Meindl, Claudia 2016)
[16](Wikipedia 2016)

Eingabe des Benutzers an den Controller weiterleitet. Die Views enthalten keine Programm Logik. In dieser Anwendung gibt es nur eine View, da diese ausschließlich aus einer Seite besteht. Auf dieser Seite werden sowohl die IST- als auch die WIRD-Wartezeiten angezeigt.

Der Controller wird vom Betrachter angesprochen und ermittelt die angeforderten Daten aus den Models und gibt diese an die View zum Anzeigen weiter. Wenn der Benutzer eine Aktion ausführt, verarbeitet der Controller diese Information und führt die Logik der korrespondierenden Aktion aus. Außerdem stellt dieser die regelmäßigen Anfragen an die REST-Schnittstelle und speichert die ermittelten Daten in der zugehörigen Model-Klasse ab.

Die Benutzeroberfläche wurde mithilfe von *Bootstrap* und *Flot* nach den Vorgaben der Mockups und User Stories gestaltet. Bilder der entwickelten Anwendung befinden sich im Kapitel A.6.

Die so erstellte Anwendung entspricht den definierten Anforderungen. Sie fordert in einem festgelegten zeitlichen Abstand die Daten von der REST-Schnittstelle an, um diese anschließend in einem View anzeigen zu lassen. Da diese Anwendung plattforunabhängig ist, kann sie auf unterschiedlichen Betriebssystemen ausgeführt werden. Der Benutzer hat die Möglichkeit, einen Zeitpunkt von bis zu einer Woche in der Zukunft auszuwählen, um sich so die Wartezeit für den ausgewählten Tag anzeigen zu lassen. Außerdem ist die aktuelle Wartezeit permanent zu sehen.

5 Fazit

Dieses Projekt hatte zum Ziel, eine Lösung für die Problematik der Wartezeitenberechnung zu konzipieren und implementieren. Zu dieser Lösung gehörte die Ermittlung der IST- und der WIRD-Wartezeit. Die Lösung sollte möglichst mit den Daten des bereits installierten Setups am Hamburg Flughafen funktionieren. Weiterhin zählte die Entwicklung einer mobilen Anwendung zum Projektumfang. Dessen Aufgabe besteht in der Anzeige der ermittelten Wartezeiten.

Am vorläufigen Ende dieses Projektes liegen ein Konzept und die Implementation von Lösungswegen für die Berechnung der IST- und WIRD-Wartezeiten vor. Die Installation dieser Lösungswege auf den Servern des Hamburger Flughafens und die damit verbundene Nutzung von realen Daten war bis zum Ende des Projekts aus bürokratischen Gründen nicht möglich. Aus diesem Grund ist die REST-Schnittstelle auf dem Server bis zum vorläufigen Ende des Projekts ebenfalls nicht implementiert. Die Implementation des Lösungsweges bezieht sich lediglich auf die Daten des Simulators. Dies ist ebenfalls der Grund dafür, dass die entwickelte Anwendung nicht auf die tatsächlich berechneten Zahlen zugreifen kann, sondern einzig auf manuell bereitgestellte Testdaten zugreift. Es wurden also Lösungswege entwickelt und anhand von simulierten Daten getestet. Des Weiteren wurde die Anwendung entwickelt, die vorerst nur auf eine Beispieldatei zugreift. Das Entwickeln eines Konzepts für die REST-Schnittstelle und die Installation der Algorithmen auf dem Server konnte ebenfalls erstellt, jedoch aus den genannten Gründen nicht umgesetzt werden.

Mit dem entwickelten Algorithmus für die IST-Wartezeit (*Lösung 2*) ist eine skalierbare Lösung für die Wartezeiten-Berechnung entwickelt worden. Skalierbar ist diese Lösung deshalb, weil sie mit beliebigen Variationen von Ticketscannern und deren Ausbeuten verwendbare Ergebnisse liefert. Die Anzahl der benutzten Ticketscanner sowie dessen Ausbeute kann beliebig variiert werden, ohne dass der Algorithmus angepasst werden muss. Da dieser Lösung nur eine geringe Anzahl an Daten zur Verfügung steht, treten noch einige Fehler auf. So kann diese Lösung zum Beispiel verbessert werden, indem Informationen über die bevorstehende Öffnung und Schließung von Kontrollspuren übermittelt werden. Auf dessen Basis können voraussichtliche Wartezeitenänderungen erkannt und in die Berechnung mit einbezogen werden. Die Erkenntnisse basieren nur auf der Verwendung des Simulators und konnten noch innerhalb keines realen Anwendungsfalls getestet werden. Weil der Simulator ein nahezu fehlerfreies Warteschlangesystem abbildet, muss die Tauglichkeit der Lösung erst noch mit den realen Daten bewiesen werden. Dabei müsste geklärt werden, was

beispielsweise passiert, wenn sich viele Passagiere in einer Warteschlange vordrängen, da so kein perfektes FIFO-System mehr vorhanden wäre.

Die Lösung für die Berechnung der WIRD-Wartezeit ermöglicht es, basierend auf dem voraussichtlichen Passagieraufkommen, eine Wartezeit zu ermitteln. Mit dieser Lösung steht ein zuverlässiges Werkzeug bereit. Aber auch bei dieser Berechnung könnte mit der Kenntnis über bevorstehende Öffnungen und Schließungen der Kontrollspuren eine weitaus präzisere Wartezeitenprognose bereitgestellt werden.

Mit dem Simulator wurde in diesem Projekt ein weiteres Tool entwickelt, das für weitaus mehr Zwecke benutzt werden kann, als lediglich zum Generieren von Testdaten. So ist es möglich eine Vielzahl an Warteschlangenszenarien zu simulieren. Mithilfe dieser könnte zum Beispiel eine Bedarfsanalyse bezogen auf die Anzahl der zu öffnenden Kontrollspuren durchgeführt werden. Ein weiteres denkbares Einsatzgebiet wäre die Planung für den Ausbau des Warteschlangensystems. Hier können zum Beispiel Erkenntnisse gewonnen werden, inwiefern sich das Erhöhen von geöffneten Kontrollspuren auf die Wartezeit auswirkt.

Die entwickelte mobile Anwendung erfüllt alle Anforderungen, die zu Beginn des Projektes gestellt wurden. Sie bietet eine übersichtliche Anzeige der ermittelten Daten. Doch auch diese konnte bis zum vorläufigen Ende des Projekts nicht in Betrieb genommen werden, da der Zugriff auf die Server und das damit verbundene Einrichten der REST-Schnittstelle sowie das Installieren der Lösungsalgorithmen nicht möglich war.

Mit den in dieser Arbeit entwickelten Lösungswegen, der mobilen Anwendung und dem Konzept für die Serverumgebung, steht dem Flughafen Hamburg eine interessante und kostengünstige Lösung für das Ermitteln und Anzeigen von Wartezeiten zur Verfügung. Sie ist deshalb kostengünstig, weil zunächst keine weitere Hardware installiert werden muss.

Auf eine fehlerfreie Zusammenarbeit dieser vielen kleinen, aufeinander aufbauenden Systeme wurde in der Implemtierungsphase ein großes Augenmerk gelegt. Da Probleme nie vollkommen ausgeschlossen werden können, wird es bei der Inbetriebnahme vermutlich noch zu kleinen Fehlern kommen. Diese gilt es dann zu beheben. Generell ist mit den hier vorgestellten Systemen aber die Grundlage für eine baldige Inbetriebnahme gelegt. Dafür müssen aber seitens des Flughafens, die in der Arbeit angesprochenen Hindernisse aus dem Weg geräumt werden.

A Material

A.1 Quellcode Ausschnitte

Listing A.1: Der Ablauf der Simulation

```
1  public Simulation simulate(Configuration config) {
2   //Die Configuration einbinden
3   this.config = config;
4   //Der Zeitparameter wird zum Start auf die Anfangszeit
       der Simulation gesetzt
5   this.clock = config.simulationStartTime;
6   //Der Endzeitpunkt der Simulation wird festgelegt
7   this.simulationEndDate = clock +
       this.config.simulationTime;
8   //Initialisiere CounterManager and ArrivalManager
9   CounterManager counterManager = new
       CounterManager(config, clock);
10  ArrivalManager arrivalManager = new
       ArrivalManager(config);
11  Integer counterPlanIndex = 0;
12  //Aktuellen Schichtplan holen
13  ArrayList<Counter> counters =
       counterManager.counterPlan.get(counterPlanIndex);
14  //Setze Initialwerte für arrivalDate und addititionalTime
15  Long arrivalDate = clock;
16  Long additionalTime = 0L;
17  //Start der Simulation
18  while (clock < simulationEndDate) {
19   Long serviceStartDate;
20   //ArrivalManager den aktuellen Zeitparameter übergeben
21   arrivalManager.update(clock);
22   //Der Schalter wird ausgewählt. Die Funtkion
        nextFreeCounter() such den, der als nächstes frei
        wird
23   Counter nextFreeCounter = nextFreeCounter(counters);
24   //Wenn der Counter geschlossen ist wird die nächste
        Schicht gestartet
```

```
25  if (nextFreeCounter.isClosed()) {
26   additionalTime = 0L;
27   //Der Zeitüberschuss wird berechnet
28   for (Counter counter : counters) {
29    if (counter.blocked - clock > 0L) {
30     additionalTime += counter.blocked - clock;
31    }
32   }
33   //Wähle den nächsten Schichtplan aus
34   counterPlanIndex += 1;
35   //Wenn es keinen Schichtplan mehr gibt wird die
       Simulation abgebrochen
36   if (counterManager.counterPlan.size() >
       counterPlanIndex) {
37    //wähle neuen Schichtplan
38    counters =
        counterManager.counterPlan.get(counterPlanIndex);
39   } else {
40    //Beende die Simulation
41    break;
42   }
43   //Verteile den Zeitüberschuss auf die neue Schicht
44   for (Counter counter : counters) {
45    counter.blocked = clock + (additionalTime /
       counters.size());
46   }
47   //Wähle aus dem neuen Schichtplan ein Kontrollspur aus
48   nextFreeCounter = nextFreeCounter(counters);
49  }
50  //Ermittle die Servicerate der Kontrollspur
51  Long serviceTime = nextFreeCounter.getServiceRate();
52  //Ermittle die aktuelle Ankunftsrate
53  Long arrivalRate = arrivalManager.getArrivalRate();
54  //Berechne den Ankunftszeitpunk (erster Zeitstempel)
55  arrivalDate += arrivalRate;
56  //Berechne den Start des Services (zweiter Zeitstempel)
57  //Berechnet die Wegzeit von 6 Sekunden mit ein
58  if (arrivalDate > nextFreeCounter.blocked) {
59   serviceStartDate = arrivalDate + 6000L;
60  } else {
61   if((nextFreeCounter.blocked-arrivalDate) < 6000L){
62    serviceStartDate = nextFreeCounter.blocked + 6000L;
63   }else{
```

```
64      serviceStartDate = nextFreeCounter.blocked;
65     }
66    }
67    //Berechne den Ende des Services
68    serviceEndDate = serviceStartDate + serviceTime;
69    //Erstelle neues Pax-Objekt und füge es der Liste hinzu
70    Pax pax = new Pax(arrivalDate, serviceStartDate,
        serviceTime, nextFreeCounter.id, serviceEndDate);
71    paxs.add(pax);
72    //Blockiere die Benutze Kontrollspur
73    counters.get(counters.indexOf(nextFreeCounter)).blocked
        = pax.serviceEndDate;
74    //Erhöhe den aktuellen Zeitparameter
75    clock = arrivalDate;
76   }
77  }
```

A.2 Tabellen

	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	27.383	19.064	13.739	12.651	10.897	10.168	10.074	9.436	9.185	9.044
2	20.226	12.631	11.802	10.275	9.560	9.635	9.259	9.207	9.108	9.056
3	14.137	10.529	9.130	8.787	8.469	8.407	8.217	8.195	8.116	8.125
4	11.294	9.805	8.967	8.674	8.892	8.498	8.551	8.413	8.359	8.359
5	11.739	9.225	8.709	8.471	8.419	8.301	8.338	8.330	8.249	8.239
6	11.542	9.526	9.068	8.829	8.893	8.847	8.756	8.789	8.717	8.713
7	9.877	8.785	8.518	8.310	8.326	8.208	8.167	8.196	8.206	8.170
8	9.480	8.736	8.589	8.336	8.306	8.291	8.212	8.273	8.236	8.219
9	9.728	8.302	8.266	8.228	8.138	8.081	8.053	8.074	8.062	8.036
10	9.630	8.772	8.655	8.481	8.371	8.445	8.414	8.381	8.385	8.376
11	9.232	8.977	8.613	8.509	8.542	8.495	8.497	8.448	8.429	8.413
12	8.954	8.321	8.287	8.189	8.171	8.190	8.123	8.125	8.116	8.115
13	8.988	8.521	8.411	8.362	8.321	8.309	8.277	8.284	8.258	8.260
14	8.518	8.232	7.944	7.898	7.849	7.827	7.849	7.830	7.812	7.795
15	8.603	8.059	7.871	7.829	7.827	7.818	7.819	7.807	7.806	7.805
16	9.730	9.301	9.190	9.255	9.242	9.195	9.161	9.136	9.147	9.142
17	9.416	9.027	8.879	8.888	8.885	8.826	8.845	8.879	8.853	8.851
18	9.069	8.526	8.526	8.466	8.453	8.439	8.365	8.405	8.402	8.387
19	8.786	8.389	8.278	8.291	8.278	8.297	8.261	8.272	8.233	8.249
20	9.119	8.682	8.707	8.661	8.638	8.644	8.594	8.592	8.581	8.573
21	9.381	9.059	8.978	8.867	8.896	8.867	8.901	8.863	8.874	8.864
22	9.212	8.983	8.955	8.944	8.946	8.936	8.954	8.935	8.926	8.920
23	8.776	8.462	8.343	8.345	8.331	8.313	8.313	8.287	8.292	8.281
24	8.708	8.515	8.421	8.407	8.330	8.358	8.335	8.310	8.315	8.299

Tabelle A.1: Durchschnittliche Differenzen

A.3 Messdaten

16.10.2015 Passagieraufkommen und Spurenöffnung

Von	Bis	PAX	K
04:00:00	04:15:00	0	0
04:15:00	04:30:00	222	13
04:30:00	04:45:00	377	18
04:45:00	05:00:00	567	18
05:00:00	05:15:00	644	23
05:15:00	05:30:00	725	23
05:30:00	05:45:00	731	23
05:45:00	06:00:00	570	23
06:00:00	06:15:00	548	23
06:15:00	06:30:00	377	23
06:30:00	06:45:00	250	23
06:45:00	07:00:00	297	12
07:00:00	07:15:00	298	11
07:15:00	07:30:00	398	17
07:30:00	07:45:00	372	17
07:45:00	08:00:00	417	16
08:00:00	08:15:00	435	19
08:15:00	08:30:00	416	19
08:30:00	08:45:00	435	23
08:45:00	09:00:00	550	23
09:00:00	09:15:00	536	23
09:15:00	09:30:00	595	23
09:30:00	09:45:00	650	23
09:45:00	10:00:00	575	23
10:00:00	10:15:00	534	23
10:15:00	10:30:00	539	23
10:30:00	10:45:00	380	23
10:45:00	11:00:00	311	23
11:00:00	11:15:00	364	18
11:15:00	11:30:00	278	17
11:30:00	11:45:00	278	17
11:45:00	12:00:00	295	15
12:00:00	12:15:00	275	11
12:15:00	12:30:00	291	12
12:30:00	12:45:00	347	15
12:45:00	13:00:00	333	19
13:00:00	13:15:00	391	20
13:15:00	13:30:00	468	20
13:30:00	13:45:00	484	20
13:45:00	14:00:00	511	18
14:00:00	14:15:00	512	16
14:15:00	14:30:00	488	16
14:30:00	14:45:00	517	16
14:45:00	15:00:00	479	18
15:00:00	15:15:00	486	20
15:15:00	15:30:00	425	20
15:30:00	15:45:00	431	20
15:45:00	16:00:00	337	20
16:00:00	16:15:00	437	18
16:15:00	16:30:00	526	18
16:30:00	16:45:00	569	20
16:45:00	17:00:00	617	22
17:00:00	17:15:00	681	22
17:15:00	17:30:00	644	22
17:30:00	17:45:00	549	22
17:45:00	18:00:00	623	23
18:00:00	18:15:00	666	23
18:15:00	18:30:00	503	23
18:30:00	18:45:00	543	23
18:45:00	19:00:00	421	23
19:00:00	19:15:00	378	15
19:15:00	19:30:00	358	15
19:30:00	19:45:00	316	13
19:45:00	20:00:00	248	14
20:00:00	20:15:00	154	12
20:15:00	20:30:00	95	8
20:30:00	20:45:00	52	4
20:45:00	21:00:00	35	4
21:00:00	21:15:00	54	2
21:15:00	21:30:00	59	2

PAX = Anzahl der Passagiere
K = Anzahl geöffneter Kontrollspuren

Abbildung A.1: Messdaten vom 16. Oktober 2015

A.4 Mockups

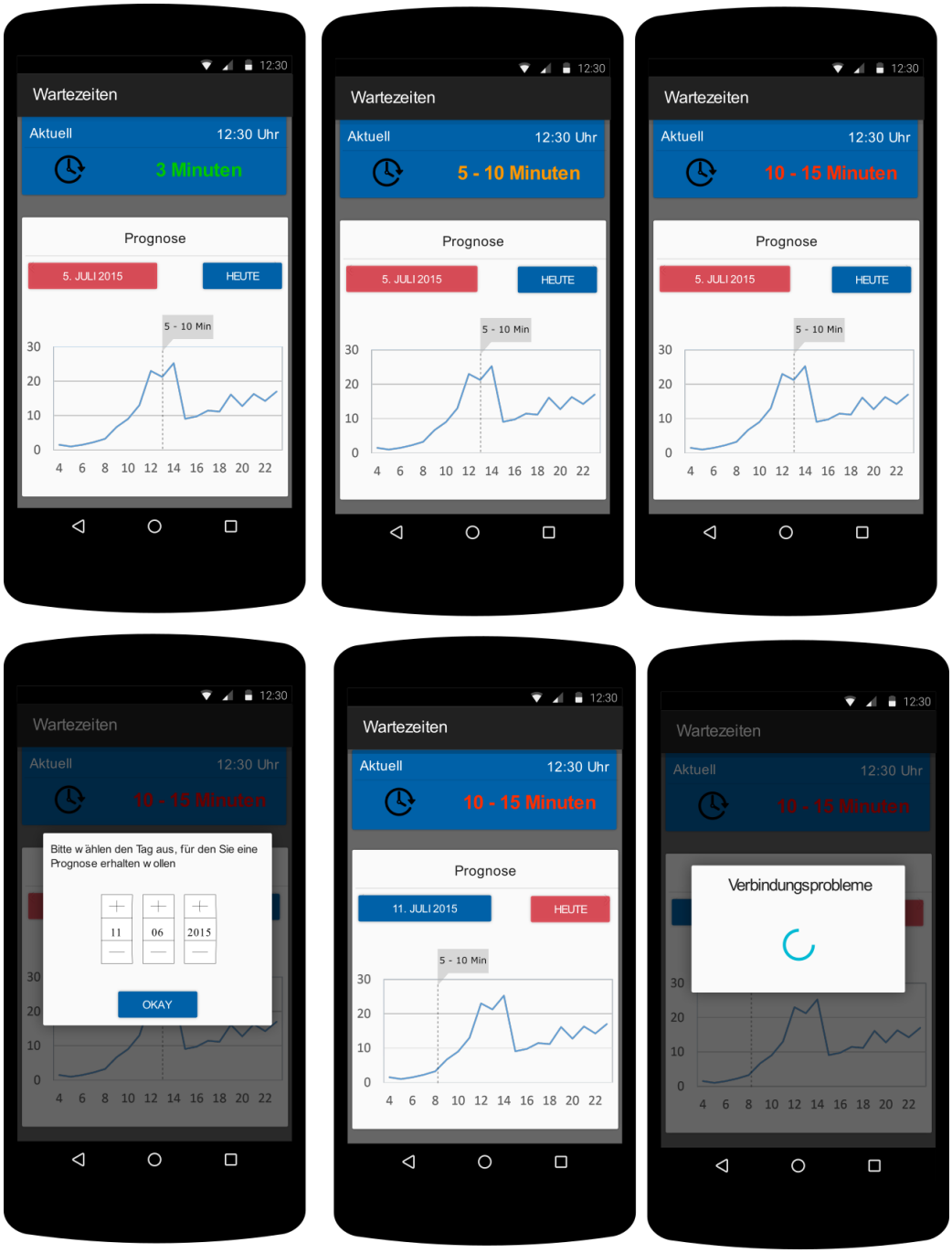

Abbildung A.2: Mockups der mobilen Anwendung

A.5 User Stories

1. Als Benutzer möchte, ich dass mir nach dem Start der Anwendung die aktuelle Wartezeit angezeigt wird, damit ich diese schnell ermitteln kann.
2. Als Benutzer möchte, ich dass mir nach dem Start der Anwendung die prognostizierte Wartezeit für den aktuellen Tag angezeigt wird, damit ich schnell ermitteln kann, wie die Wartezeit am aktuellen Tag verlaufen wird.
3. Als Benutzer möchte ich in wenigen Schritten auf die prognostizierten Wartezeiten zugreifen, damit ich diese schnell ermitteln kann.
4. Als Benutzer möchte ich auf die Wartezeiten Prognose von bis zu einer Woche in die Zukunft zugreifen, damit ich weiß, wie die Wartezeit an meinem Abflugtag wird.
5. Als Benutzer möchte ich darauf hingewiesen werden, wenn ein Verbindungsproblem besteht, damit ich weiß, ob meine Werte aktualisiert werden können.
6. Als Benutzer möchte ich in wenigen Schritten auf die aktuelle Wartezeit zugreifen können, damit ich diese schnell ermitteln kann, nachdem ich mir eine Prognose ausgegeben habe.
7. Als Benutzer möchte ich, dass die Anzeige jede Minute aktualisiert wird, damit ich immer aktuelle Werte habe.
8. Als Benutzer möchte ich in dem Diagramm per Touch-Geste die Uhrzeit der anzuzeigenden Wartezeit einstellen, damit ich auf genaue Wartezeiten schnell zugreifen kann.
9. Als Benutzer möchte ich in dem Diagramm zoomen können, damit ich eine detailliertere Ansicht bekomme.
10. Als Benutzer möchte ich, dass mir die aktuelle Wartezeit in Textform angezeigt wird, damit ich diese leicht ermitteln kann.
11. Als Benutzer möchte ich, dass mir die Wartezeiten unter fünf Minuten genau angezeigt werden, damit ich einen besseren Überblick habe.
12. Als Benutzer möchte ich, dass mir die Wartezeiten ab fünf Minuten in Fünferblöcken angezeigt wird, damit ich einen besseren Überblick habe.
13. Als Benutzer möchte ich, dass mir die prognostizierte Wartezeit als Liniendiagramm angezeigt wird, damit ich mir einen schnellen Überblick für den angezeigten Tag verschaffen kann.
14. Als Benutzer möchte ich anhand von Ampelfarben auf die Dauer der aktuellen Wartezeit hingewiesen werden, damit ich ein schnelles Feedback habe ob die Wartezeit lang oder kurz ist.
15. Als Benutzer möchte ich das Datum für die Prognose anhand eines *date-pickers* auswählen, damit ich dieses nicht manuell schreiben muss.

16. Als Benutzer möchte ich eine Markierung bei der aktuellen Uhrzeit im Diagramm, damit ich weiß, an welchem Zeitpunkt ich mich im Diagramm befinde.

A.6 Mobile Anwendung

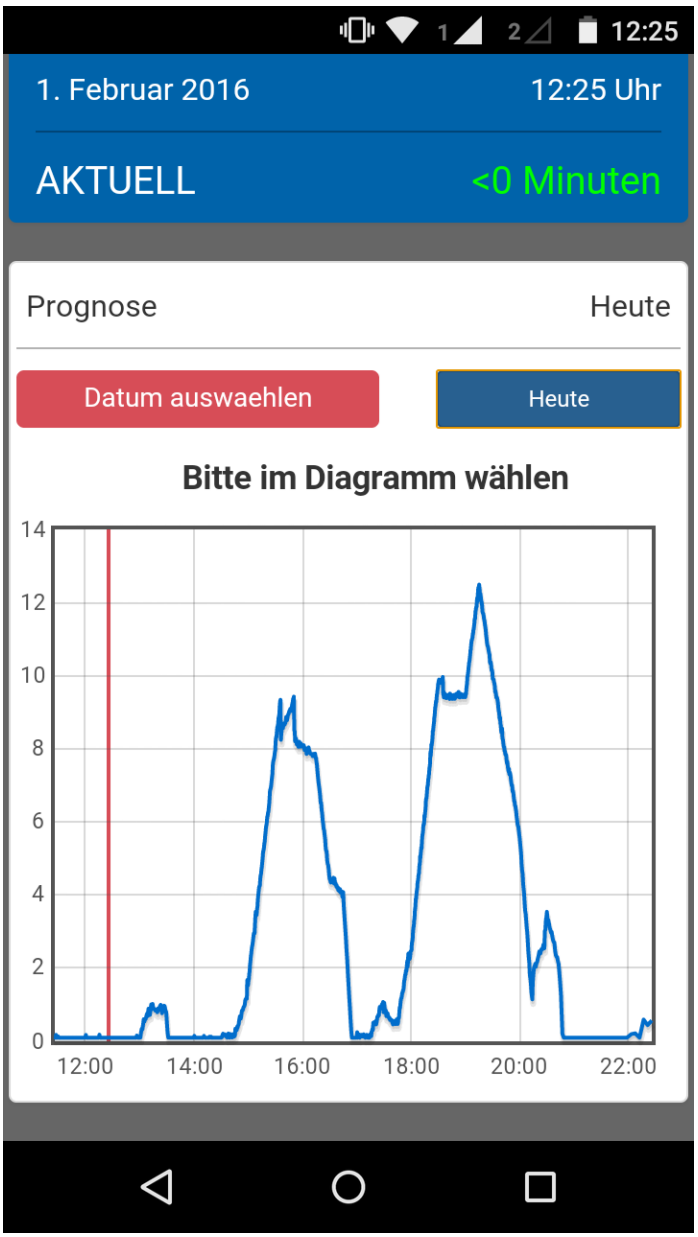

Abbildung A.3: Ausgangslage der Anwendung

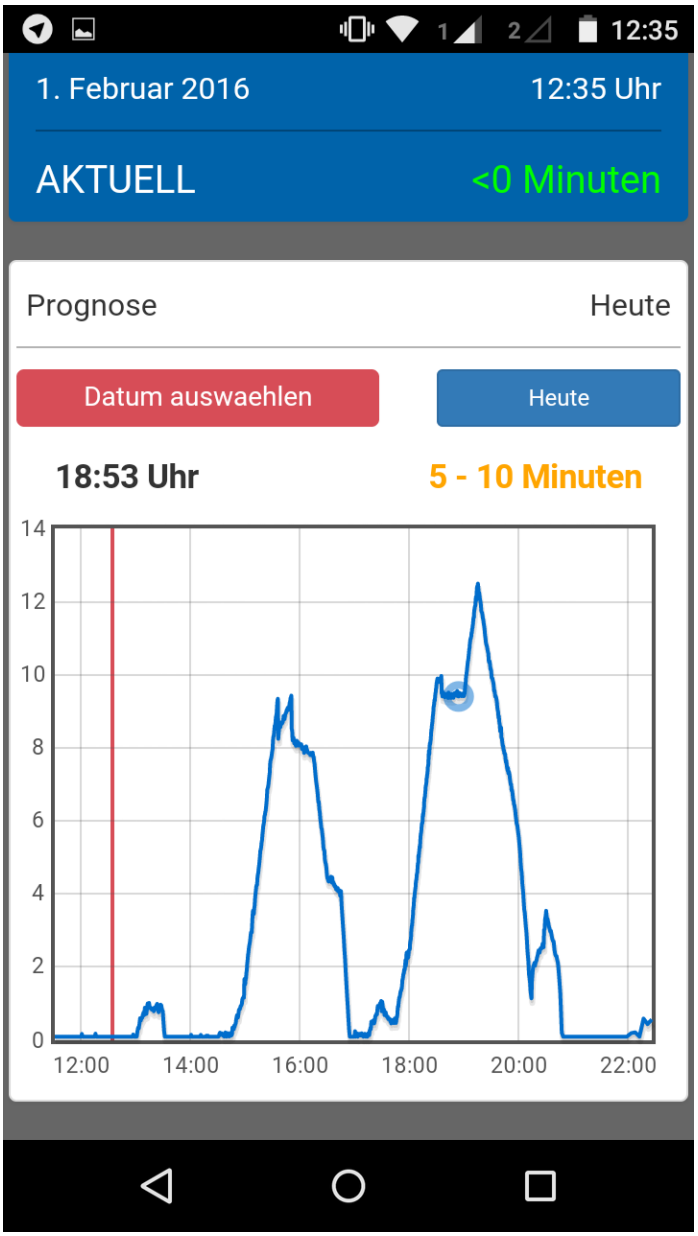

Abbildung A.4: Anzeige der WIRD-Wartezeit. 18:53 Uhr des aktuellen Tags

Abbildung A.5: Auswahl eines Datums von bis zu einer Woche in der Zukunft

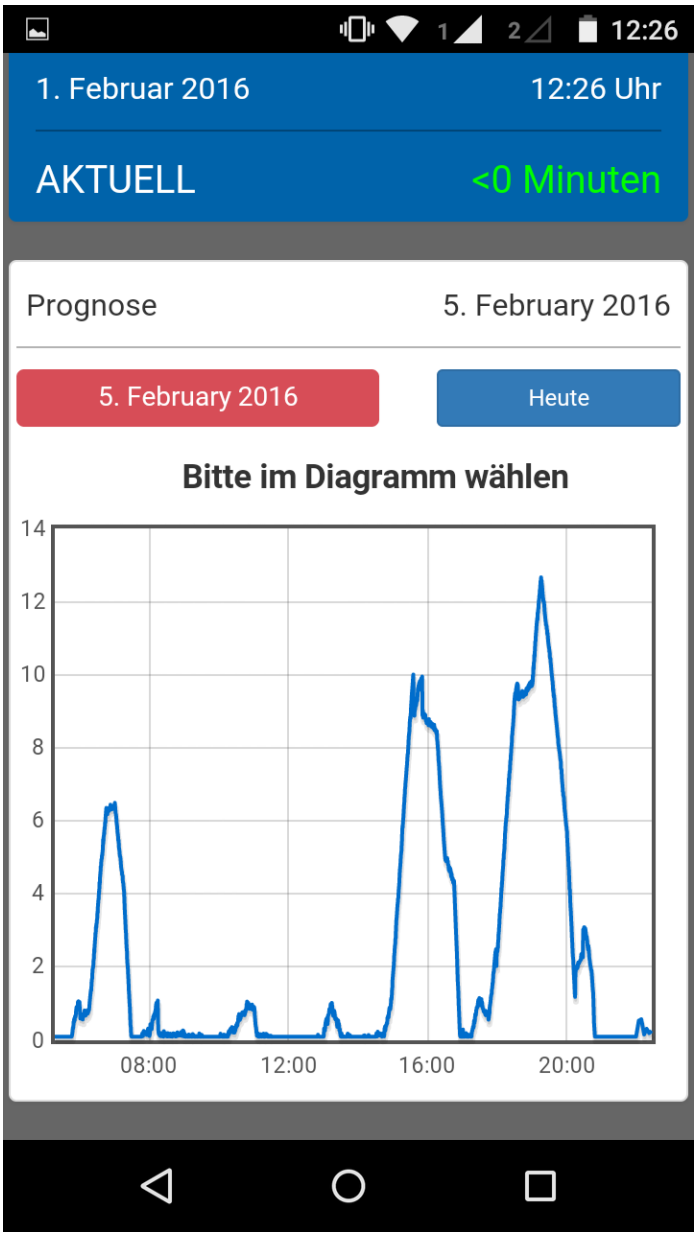

Abbildung A.6: Anzeige der WIRD-Wartezeiten für den 5. Februar

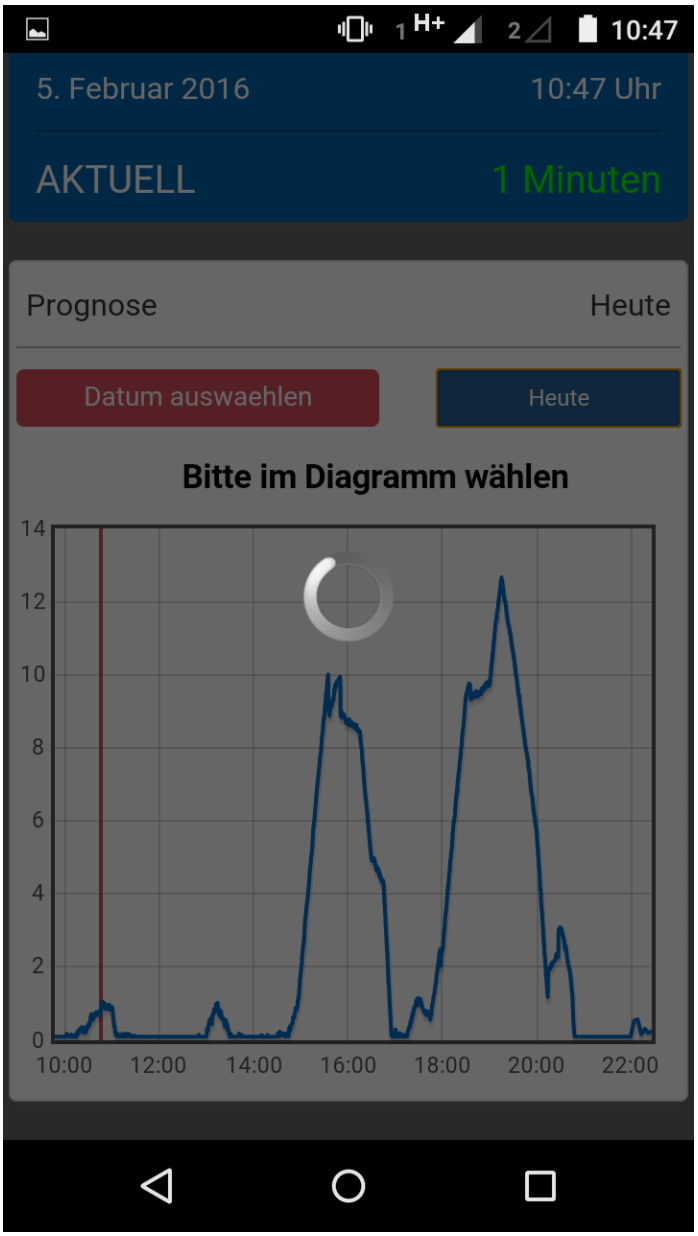

Abbildung A.7: Ladeanzeige bei Verbindung mit dem REST-Service

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

Xovis AG *Airport Customers* http://xovis.com/en/customers/airport-customers/, 2016, (besucht am 22.01.2016)

Barry, Douglas K. *Service Architecture - Application Server Definition* http://www.service-architecture.com/articles/application-servers/application_server_definition.html, 2016, (besucht am 29.01.2016)

Rodriguez, Alex *IBM - RESTful Web services: The basics* http://www.ibm.com/developerworks/webservices/library/ws-restful/, 2015, (besucht am 29.01.2016)

E-teaching *Plattformunabhängigkeit* https://www.e-teaching.org/projekt/nachhaltigkeit/plattform, 2015, (besucht am 29.01.2016)

Rouse, Margaret *WhatIs - framework* http://whatis.techtarget.com/definition/framework, 2015, (besucht am 29.01.2016)

JBoss *Red Hat JBoss Enterprise Application Platform* http://www.jboss.org/products/eap/overview/, 2016, (besucht am 29.01.2016)

Praxisbuch Objektorientierung *Die Basis der Objektorientierung* http://openbook.rheinwerk-verlag.de/oo/oo_02_basisderobjektorientierung_000.htm#Xxx999144, 2006, (besucht am 29.01.2016)

IT-Handbuch *Objektorientierte Ansatz* 5. Auflage Braunschweig: Westermann Verlag ISBN: 978-3-14-225042-7, 2007

Bootstrap *Bootstrap Homepage* http://getbootstrap.com/, 2016, (besucht am 29.01.2016)

Flotcharts *Flotcharts Homepage* http://www.flotcharts.org, 2016, (besucht am 29.01.2016)

IT-Handbuch *Definition Simulation* 5. Auflage Braunschweig: Westermann Verlag ISBN: 978-3-14-225042-7, 2007

Rouse, Margaret *WhatIs - discrete event simulation (DES)* http://whatis.techtarget.com/definition/discrete-event-simulation-DES, 2012, (besucht am 29.01.2016)

Knight, Vince *GitHub - Simulating Queues* `https://github.com/drvinceknight/Simulating_Queues`, 2013, (besucht am 29.01.2016)

Apache *Cordova - Overrview* `https://cordova.apache.org/docs/en/latest/guide/overview/`, 2015, (besucht am 29.01.2016)

AngularJS *AngularJS - What Is Angular?* `https://docs.angularjs.org/guide/introduction`, 2016, (besucht am 29.01.2016)

jQuery *About jQuery* `https://learn.jquery.com/about-jquery/`, 2015, (besucht am 29.01.2016)

Meindl, Claudia *User Stories in Scrum* `https://alphanodes.com/de/user-stories-scrum`, 2016, (besucht am 29.01.2016)

Elektronik Kompendium *Parallelschaltung von Widerständen* `http://www.elektronik-kompendium.de/sites/slt/0110192.htm`, 2014, (besucht am 29.01.2016)

RESTEasy *RESTEasy - Overview* `http://docs.jboss.org/resteasy/docs/3.0.13.Final/userguide/html_single/index.html#Overview`, 2015, (besucht am 29.01.2016)

Wikipedia *Model View Controller* `https://de.wikipedia.org/w/index.php?title=Model_View_Controller&stable=1`, 2016, (besucht am 29.01.2016)

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht.

Ort, Datum Florian Schädler