HAW HAMBURG

BACHELORTHESIS
Florian Fleischhauer

Entwurf und prototypische Umsetzung eines Systems zur Vermittlung von Terminen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Florian Fleischhauer

Entwurf und prototypische Umsetzung eines Systems zur Vermittlung von Terminen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Olaf Zukunft

Eingereicht am: 16. Juni 2021

Florian Fleischhauer

Thema der Arbeit

Entwurf und prototypische Umsetzung eines Systems zur Vermittlung von Terminen

Stichworte

Software Engineering, Software-Architektur, Terminvermittlung

Kurzzusammenfassung

Diese Bachelorarbeit beschäftigt sich mit der Entwicklung eines Systems zum Anbieten von Terminen unter der Berücksichtigung von Vergabewünschen. Nach der Analyse von Anforderungen mithilfe von Interviews und nach der Spezifikation erfolgt der Entwurf des Systems mit Verwendung einer 4-Schichten-Architektur und der Aufteilung in ein Frontend- und Backend-Subsystem. Das System wird implementiert und evaluiert. Das Ergebnis der Arbeit ist eine erfolgreiche Umsetzung des Systems mit der Erfüllung von funktionalen Kernanforderungen und der Mehrheit der Qualitätsanforderungen.

Florian Fleischhauer

Title of Thesis

Design and prototypical implementation of a system for the arrangement of appointments

Keywords

Software engineering, software architecture, appointment arrangement

Abstract

This bachelor thesis deals with the development of a system for offering appointments considering allocation desires. After the analysis of requirements using interviews and after the specification, the system is designed using a 4-layer architecture and divided into a frontend and backend subsystem. The outcome is a successful system implementation that fulfills functional core requirements and the majority of quality requirements.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Überall werden Termine verwendet – sei es für Beratungen, oder fürs Einkaufen mit Click & Meet in Corona-Zeiten. Bevor Termine feststehen, müssen sie vorher vereinbart werden. Dabei können Probleme, wie lange Verzögerungen oder ein hoher Aufwand, auftreten.

Verzögerungen entstehen, wenn man bei Terminanfragen nicht im direkten Austausch steht, wofür die E-Mail-Kommunikation ein gutes Beispiel darstellt. Der Aufwand für die Terminanbieter ist hoch, solange sie Termine nicht automatisiert anbieten, weil sie in den eigenen Terminkalender schauen, den besten Termin aussuchen und dabei auf Wünsche der Anfragenden eingehen. Zudem besteht eine geringe Flexibilität hinsichtlich der Berücksichtigung von Vergabewünschen bei vielen Online-Lösungen, wo zum Beispiel feste Zeitfenster angeboten werden müssen.

Diese Problematik kann allgemein auf alle Personen, die Termine anbieten, zutreffen. Denkbar sind viele Bereiche, wie in der Bildung, Wirtschaft oder im Gesundheitswesen, in denen Termine unverzichtbar sind, zum Beispiel eine Studienfachberatung im Bildungsbereich.

Heutzutage wird eine Unterstützung bei der Terminvergabe immer wichtiger, weil die Menschen immer mehr Aufgaben bewältigen müssen. Sie planen ihre Zeit mithilfe von Terminen, und die Vereinbarung von Terminen sollte am besten keine weitere Bürde und kaum Zeitaufwand mit sich bringen.

Diese Probleme sind durch die Informatik lösbar geworden, weil sie das automatisierte Anbieten von Terminen erleichtern kann. Das Internet ist weit verbreitet und ermöglicht die Präsentation, Bereitstellung und Verarbeitung von Informationen sowie Kommunikation für ein breites Publikum.

1.2 Zielsetzung

Das Ziel der Arbeit ist es, ein prototypisches System zu entwerfen und zu entwickeln, das es Personen ermöglicht, Einzeltermine anzubieten. Für die Vergabe sollen schon bestehende Termine und (bei Bedarf mehrere) Vergabewünsche der anbietenden Person berücksichtigt werden. Dabei soll dem Nutzer eine grafische Benutzeroberfläche zur Verfügung stehen.

Das Thema der Terminvergabe wurde für diese Arbeit ausgewählt, weil Professoren der Hochschule für Angewandte Wissenschaften Hamburg (HAW) viel Aufwand in die Terminvergabe per E-Mail stecken und es schwierig ist, den Überblick über alle Konversationen zu behalten. Zudem können dabei Verzögerungen auftreten. Die Professoren möchten bei der Vergabe von Terminen besonders eigene Wünsche berücksichtigen.

Diese Arbeit trägt dazu bei, mit einem prototypischen System eine Basis für Systeme zu bilden, die sich Methoden widmen, um dem Endanwender die Organisation der Terminvermittlung zu erleichtern und um Vergabewünsche des Terminanbieters mit möglichst wenig Einschränkungen zu berücksichtigen.

Das System soll ermöglichen, bestehende Termine zu hinterlegen, Terminangebote mit Vergabewünschen zu erstellen und Terminangebote zu buchen. Die Kernfunktionalität besteht in dem Angebot und der Buchung von Terminen.
Die Arbeit berücksichtigt nicht die Vereinbarung von Gruppenterminen, weil für eine Terminvereinbarung alle Teilnehmer reagieren (bzw. Informationen hinterlegen) müssten, was zu Verzögerungen führen könnte. Die Berücksichtigung von Terminvergabewünschen könnte bei Einzelterminen mehr Nutzen bringen als bei Gruppenterminen, weil für Einzeltermine mehr Terminmöglichkeiten zur Auswahl stehen. Dabei schränken weniger Teilnehmer die freien Zeiträume ein, wenn sie teilweise nicht können.

Ein Terminanbieter würde das System typischerweise zunächst benutzen, um schon bestehende Termine zu hinterlegen. Anschließend würde dieser Terminserien definieren, die Merkmale wiederkehrender Termine abstrahieren. Danach gibt der Anbieter ein Datum für anzubietende Termine an, darunter auch, welche Terminserien innerhalb welcher Zeiträume angeboten werden können, welche Präferenzen es für deren Vergabe gibt und welche Pausen man einplant.
Eine Person, die einen Termin benötigt und buchen möchte, ruft angebotene Terminserien des Anbieters auf, wählt eine passende aus, sucht sich ein passendes Datum und

anschließend die Uhrzeit aus. Nach Eingabe weiterer erforderlicher Daten würde dann ein Termin gebucht werden.

Terminanbieter wären zum Beispiel Professoren, und Terminsuchende wären Studenten oder externe Personen. Auch andere Einsatzgebiete, wie in Unternehmen, bei denen Mitarbeiter Beratungstermine für ihre Kunden anbieten, wären möglich.

1.3 Methodik

Um das Problem der direkten Terminvergabe unter Berücksichtigung der Vergabewünsche zu lösen, wird ein prototypisches System entwickelt. Das System soll eine grafische Benutzeroberfläche anbieten. Es ist eine iterative Vorgehensweise für die Entwicklung geplant.

Zuerst wird eine Anforderungsanalyse durchgeführt. Dafür werden bestehende Lösungen analysiert, Nutzerrollen mit Beispiel-Personen (Personas) festgelegt und Interviews mit geeigneten Personen geführt. Dadurch gewonnene Informationen sollen Aufschluss über die Ausgangs- bzw. Problemlage sowie über fachliche Konzepte liefern und Anforderungen offenlegen. Anhand eines Beispielszenarios wird die Problematik verdeutlicht. Es werden zudem funktionale Anforderungen (User Stories) und Qualitätsanforderungen ermittelt.
In der anschließenden Spezifikation werden Anwendungsfälle für die bedeutendsten funktionalen Anforderungen zusammen mit Wireframes erarbeitet und es wird eine Auswahl von Qualitätsattributsszenarien, welche die Überprüfung von Qualitätsanforderungen unterstützen, definiert. Zudem wird ein fachliches Datenmodell spezifiziert, um einen Überblick über die Domäne der Terminvereinbarung und auftretende Zusammenhänge zu ermöglichen. Ebenfalls erfolgt die Auswahl einer Systemkategorie.

Anschließend folgt der Entwurf. Darin wird die Architektur erst grundlegend strukturiert, zudem erfolgt unter anderem eine Technologieauswahl. Danach wird die Architektur hinsichtlich der Anforderungen unter Berücksichtigung von Entwurfsmustern, falls sinnvoll, weiter verfeinert und Schnittstellen definiert. Die Implementierung erfolgt auf Basis des Entwurfs. Tests werden ebenso angefertigt. In der abschließenden Evaluierung wird überprüft, inwiefern das System die Anforderungen erfüllt.

Grundlegende Architekturen und Technologien, sowie weitere Konzepte, werden vorausgesetzt, sofern diese für den Entwurf bzw. die Umsetzung benötigt werden.

1.4 Gliederung und Aufbau

Die Arbeit besteht aus sieben Kapiteln:

Kapitel 1, Einleitung, zeigt den Nutzen eines Systems für die Terminvermittlung auf und führt an die Zielsetzung sowie Methodik heran. Zudem stellt es die Gliederung und den Aufbau der Arbeit dar.
Kapitel 2, Grundlagen, erklärt Technologien, Architekturen und Konzepte, die für das Verständnis der Arbeit erforderlich sind.
Kapitel 3, Anforderungsanalyse und Spezifikation, untersucht die Ausgangslage und bestehende Lösungen in einer Ist-Analyse. Aus der Analyse wird eine Spezifikation angefertigt, die unter anderem Zielgruppen, funktionale Anforderungen, Qualitätsanforderungen, ein fachliches Datenmodell und Anwendungsfälle umfasst. In der Spezifikation wird zudem die Systemkategorie ausgewählt.
Kapitel 4, Entwurf, legt die grundlegende Systemarchitektur und die Art der Benutzeroberfläche fest, beschreibt die Aufteilung in Subsysteme sowie die zu verwendenden Technologien zusammen mit Referenzarchitekturen. Der Entwurf legt den Aufbau des Systems, dargestellt in unterschiedlichen Sichten, fest, konzipiert eine REST-API und wählt Abbildungsfunktionen für die Bewertung von möglichen Terminen aus.
Kapitel 5, Implementierung, stellt Highlights und Herausforderungen bei der Umsetzung des Entwurfs dar.
Kapitel 6, Evaluation, überprüft die Implementierung des Systems hinsichtlich der funktionalen Anforderungen und Qualitätsanforderungen. Es erfolgt dabei eine Durchführung und Auswertung von spezifizierten Qualitätsattributsszenarien.
Kapitel 7, Fazit und Ausblick, fasst den Inhalt der Arbeit zusammen, bewertet das Ergebnis und gibt einen Ausblick auf zukünftige, sinnvolle Erweiterungsmöglichkeiten des Systems.

2 Grundlagen

In diesem Kapitel werden Technologien, Architekturen und Konzepte vorgestellt, damit der Leser ein besseres Verständnis für die Inhalte der Arbeit gewinnt. Insbesondere die Architekturen sind für die Zielstellung der Arbeit relevant, weil sie den Systemaufbau beeinflussen.

2.1 Technologien

2.1.1 Spring

Spring ist ein Framework für die Programmiersprache Java und ist flexibel erweiterbar, auch mit Drittanbieter-Bibliotheken (vgl. [64]). Es ermöglicht Inversion of Control und Dependency Injection (vgl. [64]).
Mithilfe von Spring Boot lassen sich eigenständige Anwendungen realisieren, die einen eingebetteten Webserver bieten und weitestgehend automatisch konfiguriert würden (vgl. [65]).
Spring Data bietet ein einheitliches, abstrahiertes Modell für den Zugriff auf zu speichernde Daten (vgl. [66]). Es sind unterschiedliche Arten von Technologien bei der Datenspeicherung möglich, wie zum Beispiel relationale oder nicht-relationale Datenbanken (vgl. [66]). Spring Data bietet die Abstraktion von Repositories und der Objekt-Abbildung (vgl. [66]).
Spring Security ermöglicht Sicherheitskonzepte der Authentifizierung und Autorisierung, ebenso den Schutz vor ausgewählten Attacken (vgl. [68]).
Auch für das Testen sei Dependency Injection möglich, Mocking ermögliche Tests einer Implementierung in Isolation (vgl. [55]).

2.1.2 Angular

Angular ist eine Entwicklungsplattform und umfasst ein Webanwendungs-Framework, mit dem Single-Page Applications unter anderem in der Programmiersprache Typescript entwickelt werden können (vgl. [39] und [37]).
Darüber hinaus bietet Angular für eine erweiterte Funktionalität mehrere Bibliotheken, die eingebunden werden können (vgl. [39]). Ein Http-Client ermöglicht Http-Anfragen, um zum Beispiel mit einem Server kommunizieren zu können (vgl. [39]). Mittels Routing kann man zwischen unterschiedlichen Sichten navigieren, das ist in Verbindung mit der Webbrowser-URL möglich (vgl. [36]). Eine weitere Bibliothek bietet Funktionalität für Formulare einschließlich Validierung (vgl. [39]).
Angular bringt Werkzeuge für die Entwicklung mit, womit die Anwendung zum Beispiel gebaut oder getestet werden kann (vgl. [39]).

Hinsichtlich der Architektur ist das Angular-Framework in sogenannte NgModules aufgeteilt (vgl. [37] für diesen und folgende Absätze zu Angular). Diese verwalten zusammenhängende Funktionalität. NgModules können Angular-Components enthalten. Eine Angular-Component definiert eine Sicht und besteht aus einem HTML-Template (auch mit CSS), das die Sicht definiert, sowie aus einer Klasse, die anzuzeigende Daten für das zugehörige HTML-Template bereitstellt und auch Logik beinhaltet.
Es ist bidirektionales Data Binding zwischen den Anwendungsdaten und dem Document Object Model (DOM) möglich, um auf Nutzereingaben zu reagieren oder um veränderte Daten darstellen zu können.
Angular-Components können Hierarchien bilden, um sich zu komplexeren Benutzeroberflächen zusammenzusetzen. Es gibt eine Wurzel-Angular-Component, die die Gesamtsicht repräsentiert und andere Angular-Components einbinden kann. Diese Wurzel-Angular-Component ist mit dem DOM der Webseite verbunden.
Zudem bestehen Angular-Services, die Funktionalität bereitstellen, die nicht mit einer konkreten Sicht assoziiert wird. Diese können von Angular-Components genutzt werden und lassen sich mithilfe von Dependency Injection einbinden.

2.1.3 React

React ist eine JavaScript-Bibliothek, mit der Benutzeroberflächen entwickelt werden können (vgl. [21]). Es besteht eine Aufteilung in React-Components mit jeweils einem eigenen Zustand (vgl. [21]). Sie können wiederverwendet und in Kombination eingesetzt werden

(vgl. [21]). React-Components können React-Elements enthalten, welche die Benutzeroberfläche definieren (vgl. [22]). Das Document Object Model wird von React entsprechend aktualisiert, sodass es die React-Elements widerspiegelt (vgl. [22]).
Die React-Bibliothek ermöglicht unidirektionales Data Binding, wodurch sich Änderungen des Modells auf die Benutzeroberfläche auswirken (vgl. [24]). React kann, muss aber nicht für eine Single-Page Application genutzt werden (vgl. [20]).
Um eine erweiterte Funktionalität zu bieten, lassen sich externe Bibliotheken und Frameworks einsetzen (vgl. [21]). Tests werden durch Bibliotheken unterstützt (vgl. [23]).

2.2 Architekturen

2.2.1 4-Schichten-Architektur

Die 4-Schichten-Architektur umfasst die Präsentations-, Applikations-, Domänen- und Infrastrukturschicht (vgl. für den gesamten Abschnitt zur 4-Schichten-Architektur [17, S. 68-75]).
Die Präsentationsschicht zeigt Nutzern Informationen und interpretiert deren Befehle. Die Applikationsschicht definiert Aufgaben der Software, sollte dünn sein, keine Geschäftslogik enthalten, Aufgaben koordinieren und deren Probleme lösen, indem es sie an Domänenobjekte der Domänenschicht delegiert. Zudem kann diese sich unter anderem um Sicherheitsaufgaben kümmern.
Die Domänenschicht umfasst Repräsentationen von Geschäftskonzepten, Geschäftsregeln und Informationen über die Geschäftssituation. Sie nutzt und verwaltet Zustände für die Geschäftssituation, für die technische Umsetzung erfolgt eine Delegierung an die Infrastrukturschicht.
Die Infrastrukturschicht ist für grundlegende technische Funktionalitäten zuständig, zum Beispiel die Persistenz oder das Versenden von Nachrichten.

2.2.2 Microservice-Architektur

Die Microservice-Architektur ist ein Architekturstil, bei dem eine Anwendung durch Aufteilung in kleine Services entwickelt wird (vgl. für den gesamten Microservice-Architektur-Abschnitt [30]).

Ein Microservice bietet Funktionalität für eine Geschäftsfunktion und wird in einem eigenen Prozess ausgeführt. Dieser kann unabhängig und automatisiert bereitgestellt werden. Microservices sind nicht übergreifend an eine konkrete Programmiersprache oder Technologie für die Datenspeicherung gebunden.
Eine zentrale Verwaltung wird soweit möglich vermieden, und zwischen den Microservices kann eine leichtgewichtige Kommunikation erfolgen.

2.2.3 Architektur für Angular von Pietrucha

Nach der Architektur für Angular von Pietrucha bestehen die Präsentationsschicht, Abstraktionsschicht und die Kernschicht (übersetzt, vgl. [54]).

Die Kernschicht umfasst die grundlegende Applikationslogik, sie ist unter anderem zuständig für die Datenmanipulation, darunter auch Zustände, um Daten zu speichern, und ist ebenso zuständig für den Zugriff auf externe Schnittstellen, wie HTTP-Schnittstellen (vgl. [54]).
Die Abstraktionsschicht soll die Präsentationsschicht von der Kernschicht mithilfe des Facade Patterns entkoppeln und stellt Interfaces sowie Streams für Daten für die Präsentationsschicht bereit (vgl. [54]).
Die Präsentationsschicht ist nur für die Darstellung und Benutzerinteraktion zuständig und umfasst Angular-Components (vgl. [54]). Einige Angular-Components rufen die Abstraktionsschicht für das Abrufen von Daten für die Darstellung auf, und greifen auf dessen Schnittstelle zu, um Nutzeraktionen zu delegieren (vgl. [54]).
Für Angular-Components besteht bei Pietrucha eine Kategorisierung in „smart components“ und „dumb components“ (vgl. [54]). In „smart components“ können Facades oder Services injiziert werden, damit Daten von der Abstraktionsschicht abgerufen oder Aktionen ausgelöst werden können (vgl. [54]). Sie sind den „dumb components“ übergeordnet und geben Daten an diese weiter oder reagieren auf Events, die von den „dumb components“ im Rahmen von Benutzerinteraktionen ausgelöst werden können (vgl. [54]).

2.3 Weitere Konzepte

2.3.1 Richardson Maturity Model

Das Richardson Maturity Model wurde von Leonard Richardson entwickelt und zeigt unterschiedliche Stufen (Level) der Schnittstellen-Reife für den REST-Ansatz auf (vgl. [28]).
Beim Level 0 wird mithilfe von HTTP als Transportmechanismus interagiert, ohne Webmechanismen zu nutzen (vgl. [28]).
Level 1 führt Ressourcen ein, die individuell im Uniform Resource Identifier adressiert werden (vgl. [28]).
Für das Level 2 kommen zusätzlich HTTP-Verben zum Einsatz, die möglichst gemäß HTTP genutzt werden sollen (vgl. [28]). Für Antworten finden ebenso HTTP-Codes Verwendung, um zum Beispiel einen Erfolg oder Fehler zu signalisieren (vgl. [28]).
Level 3 berücksichtigt darüber hinaus HATEOAS (Hypertext As The Engine Of Application State) mit „hypermedia controls“, die mögliche Aktionen aufzeigen und eine flexible Änderung des Uniform Resource Identifier-Schemas ermöglicht (vgl. [28]).

2.4 Zusammenfassung

In diesem Kapitel wurden Grundlagen für die Technologien Spring, Angular und React, die Architekturen 4-Schichten-Architektur, Microservice-Architektur und die Architektur für Angular nach Pietrucha sowie das Konzept Richardson Maturity Model vorgestellt. Die für den Systementwurf und die Umsetzung relevanten Technologien und Architekturen werden ab dem Entwurfskapitel miteinbezogen, um unter anderem die Technologieauswahl und den Architekturaufbau nachvollziehen zu können. Das Richardson Maturity Model findet insbesondere in den Qualitätsanforderungen (Anforderungsanalyse und Spezifikation) und der REST-Schnittstelle des Entwurfs Verwendung.

3 Anforderungsanalyse und Spezifikation

In diesem Kapitel wird die Ausgangssituation inklusive bestehender Lösungen untersucht. Auf Basis der gewonnenen Informationen wird eine Spezifikation gefertigt. Sie umfasst unter anderem Anforderungen, die das System erfüllen kann bzw. soll, und ein fachliches Datenmodell, das die fachlichen Zusammenhänge aufzeigt.
Mit der Anforderungsanalyse und Spezifikation können fachliche Vorgänge der Terminvereinbarung nachvollzogen werden, um die Nützlichkeit des zu entwickelnden Systems zu verbessern, denn die Anforderungen sollen die realen Bedürfnisse widerspiegeln.

3.1 Ist-Analyse

3.1.1 Ausgangslage

Zur Ermittlung der Ausgangslage wurden Professoren (als interne Mitarbeiter) und Studenten, insgesamt 5 Teilnehmer, interviewt.
Diese Personen verwalten ihre Termine entweder manuell mittels eines Kalender-Notizbuchs oder digital, zum Beispiel mit den Lösungen Apple Kalender, Google Kalender oder Outlook.
Die befragten internen Mitarbeiter würden ihre Termine überwiegend Studenten anbieten, auch externen Personen, die Studieninteressenten sind. Anliegen seien beispielsweise eine Studienfachberatung oder eine Themenfindung für eine Bachelor- oder Masterarbeit. Bei Terminbedarf bezüglich des Studiums würden sich Studenten überwiegend per E-Mail an interne Mitarbeiter wenden. Falls vorhanden, würden auch Online-Dienste dafür genutzt. Ebenfalls seien persönliche (auch telefonische) Gespräche denkbar.

Das Anbieten der Termine erfolge per E-Mail, mündlich oder für mehrere Personen über Online-Dienste wie Doodle oder das Deutsche Forschungsnetz.
Bei einer eingegangenen, individuellen Terminanfrage werde erst die Terminart und das

Anliegen ermittelt. Anschließend müsse man einen passenden Zeitpunkt finden. Dafür kämen viele Kriterien infrage, wie die Berücksichtigung bestehender Termine oder freizuhaltender Blöcke wie Pausen. Zudem sei die Angrenzung an bestehenden Terminen zur Vermeidung von Lücken, die Nähe an Zeitpunkten oder eine weite Entfernung zu ihnen erwünscht. Zeiten, an denen der Terminnehmer nicht könne, würden vermieden. Terminwünsche würden teilweise, solange eine Begründung vorliege, berücksichtigt. Nach der Einigung auf einen Zeitpunkt sei es unter Umständen erforderlich, notwendige Informationen, wie eine Matrikelnummer, anzumerken und nachzufordern.

An der Situation der Vereinbarung per E-Mail sei es für die Terminanbieter vorteilhaft, dass man die volle Kontrolle über die Vergabe habe und Termine flexibel, zum Beispiel mit Priorisierung, vergeben könne. Im Rahmen des Interviews wurden keine Vorteile von den Studenten beschrieben. Die Studenten störe, dass die Kommunikation lange dauere und Verzögerungen, auch durch die teils schlechte Erreichbarkeit anbietender Personen, aufträten. Es bestünde Unübersichtlichkeit aufgrund der Nutzung unterschiedlicher Programme. Sowohl für Terminanbieter als auch Terminnehmer sei ein hoher Aufwand durch das Schreiben der Nachrichten erforderlich, Terminanbieter müssten teilweise Informationen nachfordern. Die Terminanbieter müssten reservierte, noch nicht bestätigte Terminvorschläge freihalten. Es sei schwer, reservierte und anschließend abgelehnte Terminvorschläge wiederzuverwenden.
Mögliche Änderungen an Terminen können beidseitige Absagen, Verschiebungen des Zeitpunkts oder eine Änderung des Themas oder der Teilnehmeranzahl sein.

3.1.2 Bestehende Lösungen

Auf den Markt gibt es für die klassische Terminverwaltung die Lösungen Apple Kalender und Google Kalender. Sie ermöglichen es unter anderem, Termine beispielsweise mit einem Titel, Details, Zeitraum, Ort, Teilnehmern und Dateien zu verwalten (vgl. [4], [43], [44]). Darüber hinaus bieten beide Systeme die Möglichkeit, Personen einzuladen, die einen Vorschlag akzeptieren, ablehnen oder signalisieren können, dass sie unschlüssig sind (vgl. [2], [1], [41]). Apple Kalender kann für die Einladung von Personen auch die Verfügbarkeit mittels eines Kalenderdienstes, z.B. CalDAV, prüfen (vgl. [2]).

Die Erinnerung an Termine bieten sowohl Apple Kalender als auch Google Kalender per E-Mail an (vgl. [3], [42]). Apple Kalender kann für die Benachrichtigung auch eine Nachricht auf dem Bildschirm oder das Öffnen einer Datei einsetzen (vgl. [3]).

Im Folgenden werden Lösungen für das Anbieten von buchbaren Terminen bzw. für die Optimierung von Terminen betrachtet.

Mit Doodle Bookable Calendar lassen sich Zeiträume angeben, in denen sich Termine einer festgelegten Länge buchen lassen (vgl. zu diesem Absatz [14]). Mithilfe einer URL kann auf die Buchungsseite zugegriffen werden, der Zugriff ist pausierbar. Die Buchungsmöglichkeiten lassen sich durch den Zeitraum für eine Buchung im Voraus, eine minimale Vorlaufzeit und einen Puffer zwischen Terminen weiter einschränken. Zusätzliche Formularfragen können angegeben werden.

Calenso bietet die Buchung von Einzelterminen an (vgl. zu diesem Absatz [8]). Dafür lassen sich Mitarbeiter und Ressourcen einplanen. Ebenso werden Pufferzeiten sowie die Vor- und Nachbereitungsdauer von Terminen berücksichtigt. Für Kunden können Fragen hinterlegt werden (wie bei Doodle Bookable Calendar).

Youcanbook.me ermöglicht für eine Terminbuchung auch die Nutzung verschiedener Termintypen, die sich unter anderem im Titel oder der Dauer unterscheiden können (vgl. zu diesem Absatz [70]). Anpassungen am Formular sind ebenso möglich wie für die Bestätigungsseite und ein Passwortschutz. Es ist möglich, eine vorläufige Buchung anzubieten, die vom Anbieter bestätigt werden muss. Buchungen lassen sich exportieren. Den Terminen können Preise zugewiesen werden, die sich während der Buchung bezahlen lassen.

Calendly ermöglicht die Buchung verschiedener Arten von Meetings, wie z.B. Einzelgespräche oder Gruppen-Meetings (vgl. zu diesem Absatz [7]). Zudem wird ein URL-Zugriff auf die Buchungsseite angeboten. Der Terminanbieter kann bevorzugte Zeiten und Standortoptionen festlegen. Neben der Kalender-Synchronisierung und der Möglichkeit von Erinnerungen bietet Calendly die Integration von Tools, wie für eine Analyse oder Zahlung, an.

Die Lösung Clockwise verfolgt den Ansatz, den Kalender zu optimieren (vgl. [10]). Das Ziel ist die Maximierung von ununterbrochener Zeit, dafür werden Meetings, auch in Konflikt stehende, automatisch verschoben (vgl. [10], [9]). Zudem kann die Reisezeit berücksichtigt werden (vgl. [10]). Eine Synchronisation ist sowohl mit dem privaten Kalender als auch mit dem Arbeitskalender möglich (vgl. [10]).

Der Ansatz des zu entwickelnden Systems überschneidet sich mit Konzepten von Lösungen wie Apple Kalender oder Google Kalender hinsichtlich der allgemeinen Terminverwaltung, der Synchronisation zu Kalenderdiensten und der Benachrichtigung per E-Mail. Die Verwaltung von Terminen wird benötigt, um Termine für die Buchung erstellen und

bestehende Termine berücksichtigen zu können. Daher muss die Terminverwaltung im zu entwickelnden System enthalten sein, auf ihr liegt nicht der Fokus.
Die Möglichkeit der Buchung von Terminen mit verschiedenen Terminarten, mit konfigurierbaren Formularfragen und Wünschen wie eine Vor-/Nachbereitungsdauer oder einem Puffer ist ebenso für das System vorgesehen. Man findet sie bereits (teilweise) in Lösungen wie Doodle Bookable Calendar, Calenso, Youcanbook.me oder Calendly.
Das Ziel der Maximierung ununterbrochener Zeit von Clockwise soll vom zu entwickelnden System auch berücksichtigt werden, jedoch nur für die Buchung als Anbieterwunsch für die Vergabe von Terminen. Die Möglichkeit von Calendly, spezielle Zeiten zu bevorzugen, findet sich im zu entwickelnden System ebenfalls wieder.

Das System unterscheidet sich von bestehenden Lösungen hinsichtlich des Geschäftsmodells und der Möglichkeit des Betriebs. Es besteht kein gewinnorientiertes Geschäftsmodell, keine Werbung und zugleich soll das System eigenständig betrieben werden können, um die eigene Kontrolle über die Datenhaltung zu ermöglichen.

Der bedeutendste Unterschied besteht in der Buchung. Flexibel bewegliche Pausen werden berücksichtigt. Es werden nicht nur alle möglichen konfliktfreien Terminmöglichkeiten ermittelt und angeboten, sondern nur diejenigen, die dem Terminanbieter nach selbst konfigurierten Kriterien am Besten passen.
Für diese Kriterien sollte nicht nur berücksichtigt werden können, dass zu vergebende Termine möglichst nah an einem Zeitpunkt oder Zeitraum sind und dass möglichst wenig freie Zwischenräume entstehen würden, sondern auch unter anderem, dass diese Termine möglichst früh bzw. spät sind, möglichst weit von einem Zeitpunkt bzw. -raum entfernt sind, und dass eine möglichst hohe Standardabweichung von Zwischenraumlängen besteht.
Es sticht hervor, dass nicht eine einzige, sondern alle der flexiblen, festgelegten Kriterien gleichzeitig für eine dynamische Bewertung der Terminmöglichkeiten herangezogen werden können.

3.2 Zielgruppen (User Roles)

Im Folgenden werden ermittelte Zielgruppen (User Roles) und teilweise Personas dargestellt, die jeweils eine konkrete fiktive Person zu einer Nutzerrolle repräsentieren. Die Ermittlung der Zielgruppen und Personas erfolgt nach Cohn (vgl. [11, S. 31 ff.]).

Administratorin oder Administrator

Attributbeschreibung	Attributinhalt
Genutzte Funktionsart	Wartungsbezogene Funktionen
Nutzungstiefe	Tief
Kenntnis der Domäne	Gut
Kenntnis des Systems	Sehr gut
Computererfahrung	Hoch (Experte)
Ziele	Wartung mit möglichst wenig Aufwand
Wichtige Systemeigenschaften	Detaillierter Einblick in Ereignisse, Fehler des Systems, gute Verständlichkeit des Systems (Dokumentation)

Tabelle 3.1: Nutzerrolle: Administratorin oder Administrator

Außenstehende Person

Attributbeschreibung	Attributinhalt
Nutzungsfrequenz	Selten
Nutzungstiefe	Oberflächlich
Kenntnis der Domäne	Mittel bis hoch
Kenntnis des Systems	Gering
Genutzte Funktionsart(en)	Terminsuche, -vereinbarung und -absage (Nutzung des Terminangebots von Bediensteten)
Computererfahrung	Gering bis hoch, kommt mit Web zurecht
Ziele	Unkomplizierte Terminvereinbarung, Angabe nur der nötigsten Daten, einfache Bedienung

Tabelle 3.2: Nutzerrolle: Außenstehende Person

Interne Mitarbeiterin oder interner Mitarbeiter der Organisation

Attributbeschreibung	**Attributinhalt**
Nutzungsfrequenz	Selten bis oft
Nutzungstiefe	Oberflächlich bis tief (je öfter genutzt)
Kenntnis der Domäne	Mittel bis hoch
Kenntnis des Systems	Gering bis hoch
Genutzte Funktionsart(en)	Termine anbieten (überwiegend), Terminsuche, Terminvereinbarung und -absage (für Besprechungen)
Computererfahrung	Gering bis hoch, je nach Arbeitsbereich, kommt mit Web zurecht
Ziele	Effiziente und aufwandsarme Verwaltung der Termine, gute Berücksichtigung eigener Präferenzen für neue Termine ohne Kollisionen oder Leerlauf zwischen Terminen, einfache Bedienung

Tabelle 3.3: Nutzerrolle: Interne Mitarbeiterin oder interner Mitarbeiter der Organisation

Name	Manfred
Alter	42
Wohnort	Hamburg-Ost
Beruf	Professor. Hat Informatik studiert und promoviert, wurde vor 5 Jahren zum Professor berufen.
Familienstand	Verheiratet, 2 Kinder (5 und 7 Jahre alt)
Hobbies	Tennis, Modellbau
Charakter	Ausgeglichen, sehr höflich und respektvoll, stellt sich selbst in den Hintergrund.
Ziele	Möchte für das Arbeits- und Privatleben eine bessere Balance finden, um sich mehr Zeit für die Familie nehmen zu können. Möchte sich in seinem Fachbereich auf dem neuesten Stand halten, und Forschungen für aktuelle Themen leiten.

Tabelle 3.4: Persona: Interne Mitarbeiterin oder interner Mitarbeiter der Organisation

Studentin oder Student (Mitglied der Organisation)

Attributbeschreibung	Attributinhalt
Nutzungsfrequenz	Gelegentlich
Nutzungstiefe	Eher oberflächlich
Kenntnis der Domäne	Mittel bis hoch
Kenntnis des Systems	Gering bis mittel
Genutzte Funktionsart(en)	Terminsuche, -vereinbarung und -absage (überwiegend, mit internen Mitarbeitern, vereinzelt mit Kommilitonen), Termine anbieten (selten)
Computererfahrung	mittel bis hoch, kommt mit Web zurecht
Ziele	Terminvereinbarung ohne Kollision mit bestehenden Terminen, keine Registrierung bei seltener Nutzung, einfache Bedienung

Tabelle 3.5: Nutzerrolle: Studentin oder Student

Name	Lena
Alter	24
Wohnort	Studentenwohnheim in Hamburg-Mitte
Beruf	Studentin. Hat einen Job als Aushilfe in einem Elektronikfachhandel
Familienstand	Ledig
Hobbies	Lesen, Fechten
Charakter	Selbstbewusst, zielstrebig, hilfsbereit
Ziele	Möchte in ihrem Studium möglichst viel lernen, um in Zukunft einen guten Job finden zu können, der ihr gefällt. Sucht nach einer festen Wohnung in Hamburg, ist offen für einen Partner.

Tabelle 3.6: Persona: Studentin oder Student

3.3 Beispielszenario

Im Folgenden wird ein Szenario nach der Vorgehensweise von Sommerville (siehe [59, S. 70-76]) für das Anbieten eines Termins beschrieben. Darin wird die Nutzungssituation eines internen Mitarbeiters der Organisation dargestellt, das bestehende Problem erklärt und ein Weg der Lösung beschrieben. Es finden Personas Einsatz.

Manfred ist ein Professor in Hamburg und bietet regelmäßig Vorlesungen und Beratungen für Studenten, interne Personen oder außenstehende Personen an. Er möchte, dass Termine für Beratungen möglichst ohne viel Aufwand für ihn vereinbart werden.

Er hat soeben eine Anfrage von der Studentin Lena per E-Mail erhalten, in der um ein Gespräch gebeten wird, um sich über Prüfungsverfahren zu informieren. Er schätzt, wie lange der Termin dauern würde, und schaut in seinem Kalender nach, um eine freie Zeit zu finden. Diese reserviert er erst einmal für die Studentin und bietet sie ihr an. Es dauert einen Tag, bis Lena antwortet. Zwischenzeitlich hat Manfred einen nachfolgenden Termin mit einer außenstehenden Person vereinbart. Lena kann zu dem reservierten Zeitpunkt nicht und bietet einen anderen Tag an. Manfred schaut noch einmal in seinem Kalender nach, und bietet einen alternativen Termin an, den Lena ihm innerhalb kurzer Zeit bestätigt.

Manfred stört, dass die Kommunikation per E-Mail unübersichtlich ist, dass er jedes Mal die Zeit schätzen, im Kalender nachschauen, einen Termin vorschlagen und reservieren muss. Zudem gibt es eine gewisse Verzögerung. Durch die in diesem Fall nicht passende Reservierung ist ihm eine Lücke im Kalender entstanden, die er gerne vermieden hätte.

Durch das System kann er nach Anmeldung verschiedene Terminserien definieren, die unterschiedliche Längen haben, und von Studenten, internen oder außenstehenden Personen im Rahmen von Terminbuchungen ausgewählt werden können. Manfred legt die Zeiträume fest, an denen er Termine anbieten möchte und gibt an, welche Terminserien er darin anbietet. Zudem kann er verschiedene Präferenzen (wie z.B. Pausen oder Staffelungen zu bestimmten Zeiten) festlegen, damit diese beim Anbieten berücksichtigt werden.
Nun kann Lena für eine Terminserie einen verfügbaren Tag aussuchen, und ihr wird ein Zeitpunkt, der für Manfreds Präferenzen am passendsten wäre und nicht mit bestehenden Terminen in Konflikt steht, vorgeschlagen. Danach gibt sie ihre Kontaktinformationen an, um die Buchung abzuschließen. Nach der Buchung werden sowohl Manfred als auch Lena darüber benachrichtigt, dass der Termin festgelegt wurde.

3.4 Funktionale Anforderungen

Die funktionalen Anforderungen an das zu entwickelnde System werden durch User Stories repräsentiert. Sie wurden unter anderem aus den geführten Interviews ermittelt.
Im Folgenden ist eine Auswahl der User Stories dargestellt. Eine vollständige Auflistung ist im Anhang zu finden. Einige User Stories besitzen zur Referenzierung ein Kürzel, beginnend mit „US-“ (Nummierung unabhängig von der Darstellung), wenn die Referenz im Rahmen der Entwicklung benötigt wurde.

US-001 Als Administrator kann ich als Artefakte konfigurierbare Docker-Images erzeugen, um das System betreiben zu können.

US-002 Als Administrator kann ich Systemereignisse und Fehlermeldungen persistent protokollieren lassen, um getätigte Aktionen im Zweifel nachweisen zu können (bezieht sich nur auf Systemteile, die fachliche Daten verwalten und auch fachliche Funktionalität bieten).

US-003 Als interner Mitarbeiter kann ich einen Einzeltermin erstellen, damit außerhalb des Systems bestehende Termine auch berücksichtigt werden.

US-004 Als Student oder interner Mitarbeiter oder externe Person kann ich Informationen eines Einzeltermins einsehen, um mich auf den Einzeltermin vorbereiten zu können.

US-019 Als Student oder interner Mitarbeiter oder externe Person kann ich einen Einzeltermin absagen, um zu signalisieren, dass der Einzeltermin nicht mehr benötigt wird oder nicht wahrgenommen werden kann.

US-005 Als interner Mitarbeiter kann ich Einzeltermine in einer Wochenansicht einsehen, um den Zeitplan der aktuellen/einer speziellen Woche nachvollziehen zu können.

US-020 Als Student oder externe Person kann ich mithilfe eines Links auf einen von mir gebuchten Einzeltermin zugreifen, um ihn einsehen, ändern oder absagen zu können.

- Als Student oder externe Person oder interner Mitarbeiter kann ich eine einfache Wertangabe für einen Einzeltermin erstellen, um benötigte Informationen zu hinterlegen.
- Als interner Mitarbeiter kann ich eine Angabenanforderung für einen Einzeltermin erstellen, um einen Teilnehmer an benötigte Informationen für den Termin zu erinnern.
- Als Student oder externe Person oder interner Mitarbeiter kann ich einen Kommentar für einen Einzeltermin erstellen, um erforderliche Informationen anzugeben.

US-006 Als interner Mitarbeiter kann ich eine Terminserie erstellen, um diese für zukünftig zu vereinbarende Einzeltermine zu nutzen.

US-009 Als interner Mitarbeiter kann ich einen Terminangebotszeitraum für Terminserien erstellen, damit andere Nutzer einfach Einzeltermine buchen können.

US-012 Als Student oder externe Person kann ich angebotene Terminserien einer Person einsehen, um eine Vorlage für einen Einzeltermin auszusuchen.

US-013 Als Student oder externe Person kann ich buchbare Terminmöglichkeiten einer Terminserie einsehen, um einen für mich passenden Zeitpunkt für einen Einzeltermin zu finden.

US-014 Als Student oder externe Person kann ich buchbare Terminmöglichkeiten unter Berücksichtigung von Zeiträumen, an denen ich (nicht) kann, bei gegebener Terminserie einsehen, um einen Termin zu finden, an dem ich auch kann.

US-015 Als Student oder externe Person kann ich eine Terminmöglichkeit buchen, um einen Einzeltermin festzulegen.

US-016 Als interner Mitarbeiter kann ich mich für einen Account registrieren, um zukünftig Termine für andere Personen anbieten zu können.

US-017 Als interner Mitarbeiter kann ich mich anmelden, um meinen Account zu nutzen.

3.5 Qualitätsanforderungen

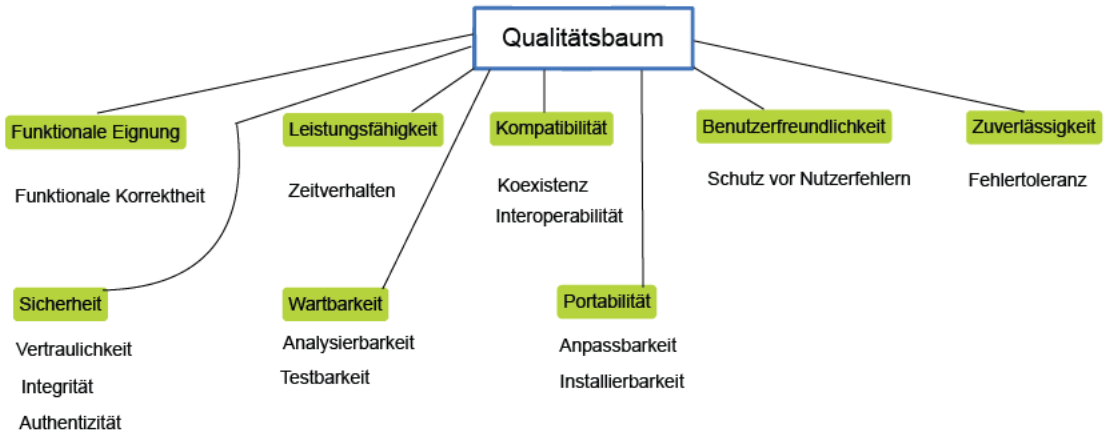

Abbildung 3.1: Qualitätsbaum nach Merkmalen von ISO 25010 (vgl. [49], übersetzt)

Qualitätsmerkmale, die im Rahmen der prototypischen Entwicklung berücksichtigt werden sollen, sind in einem Qualitätsbaum in der Abbildung 3.1 dargestellt. Dafür werden die Qualitätsmerkmale von ISO 25010 (vgl. [49], übersetzt) genutzt.

Im Folgenden werden Qualitätsanforderungen zu den Qualitätsmerkmalen angegeben, die unter anderem aus Informationen von Interviews ermittelt wurden. Die Nummerierung weist Lücken auf (7 und 8).

Funktionale Eignung: Funktionale Korrektheit

QA-1: Die Korrektheit umgesetzter funktionaler Anforderungen wird mittels Tests in einem nach Ansicht des Entwicklers angemessenen Umfang überprüft, und alle Tests der Funktionen sind hinsichtlich funktionaler Korrektheit erfolgreich.

Leistungsfähigkeit: Zeitverhalten

QA-2: Das System reagiert in mindestens 85% der Fälle innerhalb von 4 Sekunden bei geringer Auslastung.

Kompatibilität: Koexistenz

QA-3: Das System ist als (mindestens) ein Docker-Container lauffähig (und kann somit potenziell mit anderen Systemen auf der gleichen Umgebung koexistieren).

Kompatibilität: Interoperabilität

QA-4: Die Funktionalität des Systems ist mithilfe einer HTTP-Schnittstelle verfügbar, die dem REST Level 2 nach dem Richardson Maturity Model entspricht (Ausnahme: Benutzeroberfläche).

Benutzerfreundlichkeit: Schutz vor Nutzerfehlern

QA-5: Das System weist den Nutzer auf Fehleingaben hin, die nicht dem erforderlichen Format oder nicht der fachlichen Eingabemenge entsprechen.

Zuverlässigkeit: Fehlertoleranz

QA-6: Das System ist gegen Fehleingaben abgesichert. Als Fehleingaben gelten hier Eingaben, die im Format oder hinsichtlich fachlicher Regeln abweichen.

Sicherheit: Vertraulichkeit

QA-9: Daten von Nutzern, die ausschließlich für sie selbst bestimmt sind, lassen sich nicht von anderen Personen abrufen.

QA-10: Ein nicht angemeldeter Nutzer kann nicht die Funktionen aufrufen, die ausschließlich für angemeldete Nutzer zugänglich sein sollen.

Sicherheit: Integrität

QA-11: Daten eines Nutzers, die nicht für die Bearbeitung freigegeben wurden, lassen sich nicht von einem anderen Nutzer bearbeiten oder löschen.

Sicherheit: Authentizität

QA-12: Die Identität eines Nutzers kann durch Angabe eines Benutzernamens und Passworts bestätigt werden.

Wartbarkeit: Analysierbarkeit

QA-13: Das Logging erfolgt in einer konsistenten Art und Weise.

QA-14: Ursprünge von auftretenden Fehlern können durch Informationen von Logs eingegrenzt werden.

QA-15: Alle externen Schnittstellen, welche die fachliche Funktionalität bereitstellen, sind dokumentiert (Ausnahme: Schnittstellen für Benutzeroberflächen).

Wartbarkeit: Testbarkeit

QA-16: Das System lässt sich mit ausführbaren Tests testen (mit getrennten Tests der Subsysteme).

QA-17: Automatisierbare (im Rahmen dieser Arbeit entwickelte) Tests des Systems lassen sich automatisiert ausführen.

Portabilität: Anpassbarkeit

QA-18: Build-Artefakte des Systems sind mit Umgebungsvariablen konfigurierbar.

Portabilität: Installierbarkeit

QA-19: Das System ist als (mindestens) ein Docker-Container lauffähig (Ausnahme: erforderliche externe Systeme bzw. Persistenzlösungen).

3.6 Qualitätsattributsszenarien

Für die Qualitätsattributsszenarien wird ein Schema aus dem Buch „Software Architecture in Practice“ eingesetzt (siehe [5, S. 75]). Referenzen auf sie beginnen mit „QAS-“, gefolgt von einer Identifizierungsnummer.
Folgende Qualitätsattributsszenarien stehen im Kontext der Qualitätsanforderung QA-2, welche aussagt, dass das System in mindestens 85 % der Fälle innerhalb von vier Sekunden bei geringer Auslastung reagieren soll.

Attributbeschreibung	**Attributinhalt**
Quelle des Stimulus	Person
Stimulus	Aufruf der Benutzeroberfläche einer erwünschten Funktionalität des Systems (mit URL)
Artefakt	System (Frontend, Backend und Datenbank), Internetbrowser
Umgebung	Normale Umgebung des Systems mit geringer Auslastung
Antwort	Vollständige Darstellung der Benutzeroberfläche
Antwortmaß	Die Dauer, bis die Benutzeroberfläche komplett geladen und angezeigt ist, beträgt in mindestens 85 % der Fälle maximal 4 Sekunden.

Tabelle 3.7: Qualitätsattributsszenario QAS-01: Laufzeit des Ladens und der Anzeige einer aufgerufenen Benutzeroberfläche

Attributbeschreibung	**Attributinhalt**
Quelle des Stimulus	Person
Stimulus	Anstoß eines Vorgangs des Systems nach Ausfüllung bzw. Bereitstellung erforderlicher Daten. Es sind nur Vorgänge zu nutzen, die ab dem Anstoß bis zum Abschluss keine weiteren Benutzereingaben erfordern (z.B. Buchung einer ausgewählten Terminmöglichkeit durch Klick auf „Ausgewählten Termin buchen“).
Artefakt	System (Frontend, Backend und Datenbank), Internetbrowser
Umgebung	Normale Umgebung des Systems mit geringer Auslastung
Antwort	Vollständige Darstellung der Benutzeroberfläche nach Abschluss des Vorgangs (mit Anzeige einer Bestätigung, falls verfügbar)
Antwortmaß	Die Dauer zwischen dem Anstoß des Vorgangs und dem Zeitpunkt, bis die Benutzeroberfläche nach dem Abschluss des Vorgangs inklusive einer Bestätigung komplett geladen bzw. angezeigt ist, beträgt in mindestens 85 % der Fälle maximal 4 Sekunden.

Tabelle 3.8: Qualitätsattributsszenario QAS-02: Laufzeit des Abschluss eines Vorgangs und der anschließenden Anzeige

3.7 Fachliches Datenmodell

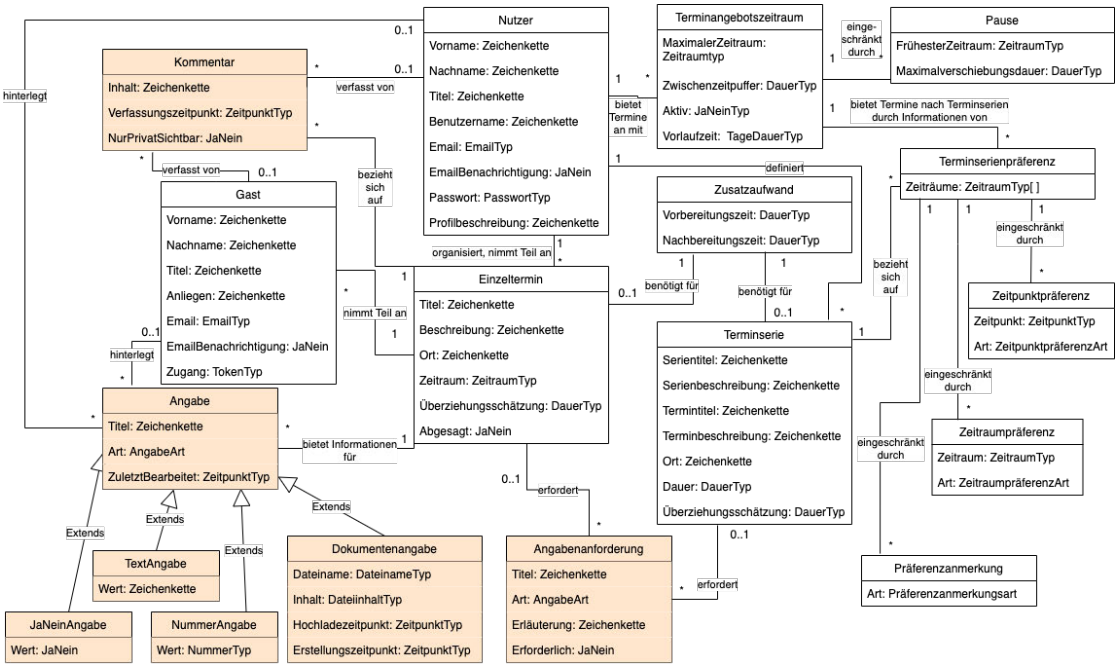

Abbildung 3.2: Fachliches Datenmodell

Um die Daten aus den fachlichen Konzepten mit deren Zusammenhängen zu modellieren, wurde ein fachliches Datenmodell, wie in Abbildung 3.2, erarbeitet. Farblich unterlegte Inhalte des fachlichen Datenmodells wurden im Rahmen der Arbeit nicht umgesetzt.

Der Nutzer steht im Mittelpunkt des fachlichen Datenmodells. Er ist registriert, besitzt also einen Benutzernamen und Passwort, welches aus Sicherheitsgründen nicht im Klartext, sondern mittels einer Einwegfunktion (zum Beispiel eine kryptographische Hashfunktion) maskiert sein sollte.

Ein Nutzer kann beliebig viele Einzeltermine anbieten. Ein Einzeltermin hat unter anderem Informationen über den Titel, die Beschreibung, den Ort und insbesondere einen Startzeitpunkt und die Dauer. Die Kombination aus Startzeitpunkt und einer Dauer anstelle eines Endzeitpunkts vereinfacht die Prüfung des Zeitraums, weil dafür die Dauer benötigt wird, die ansonsten vorher berechnet werden müsste. Beispielsweise muss geprüft werden, ob die Dauer positiv ist und ob sie weitere mögliche Bedingungen erfüllt.

Ein Einzeltermin stellt einen konkreten Termin mit mindestens einer Person dar (dem Nutzer und beliebigen Gästen) und kann auch ohne einem Gast existieren, falls der Nutzer alleine an dem einzutragendem Termin teilnimmt.

Der Gast nimmt an genau einem Einzeltermin teil. Er muss sich nicht registrieren, weil die Registrierung gemäß den Interviews nicht erwünscht ist, und existiert nur in Verbindung mit einem Einzeltermin. Das Anliegen des Gastes ist relevant für den Terminanbieter. Es kann leer sein, falls der Nutzer den Gast selber für einen Einzeltermin eingetragen hat, für eine Buchung durch den Gast ist es aber notwendig. Damit der Gast auf das System bzw. die terminspezifischen Inhalte zugreifen kann, besitzt er einen Zugang, der als Token umgesetzt ist.

Kommentare können für einen Einzeltermin entweder von dem Nutzer, der den Einzeltermin anbietet, oder von dem Gast, der an dem Termin teilnimmt, verfasst werden. Ein Kommentar kann nur existieren, solange der dazugehörige Einzeltermin und Verfasser existiert.

In einer Angabe können Informationen zu einem Einzeltermin hinterlegt werden. Sie kann beliebig oft für einen Einzeltermin angelegt werden. Die Angabe wird entweder vom Gast, der am Termin teilnimmt, oder vom Nutzer, der den Termin anbietet, eingetragen. Sie existiert maximal so lange wie der zugehörige Einzeltermin und der eintragende Gast oder Nutzer.
Neben dem Titel und dem Zeitpunkt der letzten Bearbeitung, um Änderungen einfacher nachvollziehen zu können, ist der Datentyp für die Angabe relevant (als „AngabeArt“ gekennzeichnet). Mögliche Typen sind eine Textform, ein Wahrheitswert, eine Nummer oder ein Dokument. Um den konkreten Wert mit speichern zu können, wird Vererbung bei diesen Datentypen eingesetzt.

Ein Nutzer kann beliebig viele Terminserien definieren. Sie dienen als Vorlage für einen Einzeltermin und weisen keine Eigenschaften auf, die sich auf einen konkreten Zeitpunkt beziehen. Eine Terminserie hat zudem einen Serientitel und eine Serienbeschreibung, damit der Nutzer verschiedene Terminserien einfacher durch unterschiedliche Benennungen auseinanderhalten kann und erweiterte Beschreibungen ermöglicht werden.

Um im Voraus mögliche oder erwartete Angaben für Termine oder Terminserien beschreiben zu können, besteht die Angabenanforderung. Diese wird entweder für einen Einzeltermin oder eine Terminserie eingesetzt, und existiert maximal so lange wie der zugehörige Einzeltermin bzw. die zugehörige Terminserie.

Die Angabenanforderung umfasst einen Titel und einen Datentyp analog zur Angabe. Zusätzlich wird eine Erläuterung und die Information gegeben, ob die Angabe zwingend für den Termin benötigt wird, um den Gast auf noch fehlende Angaben hinweisen zu können. Eine Angabe erfüllt eine Angabenanforderung, wenn sie denselben Titel und Datentyp wie die Angabenanforderung aufweist.

Es kann vorkommen, dass ein Terminanbieter zusätzliche Zeit für die Vorbereitung und Nachbereitung einplanen muss. Genau das umfasst der Zusatzaufwand. Er ist für Einzeltermine und Terminserien erforderlich. Ein Zusatzaufwand bezieht sich entweder auf einen Einzeltermin oder auf eine Terminserie und besteht maximal so lange wie der Einzeltermin bzw. die Terminserie.

Damit der Nutzer Termine für andere Personen anbieten kann, besteht der Terminangebotszeitraum. Dieser bietet Informationen für Angebote an einem konkreten Tag. Das bedeutet, dass dessen maximaler Zeitraum, in dem Termine angeboten werden dürfen, innerhalb eines Tages liegen muss. Zusätzlich bietet der Terminangebotszeitraum Informationen darüber, ob er aktiv ist, wie viel Puffer ein angebotener Termin mindestens zu bestehenden Terminen aufweisen sollte, und wie kurzfristig die Termine angeboten werden (durch eine Angabe von Tagen im Voraus).
Ein Nutzer kann beliebig viele Terminbuchungszeiträume anbieten, aber nur einen pro anzubietendem Tag.

Für einen Terminangebotszeitraum können Pausen berücksichtigt werden. Sie sind verschiebbar. Dafür wird der frühestmögliche Zeitraum angegeben und wie weit dieser maximal nach hinten verschoben werden kann. Eine Bedingung für jede Pause ist, dass ihr Zeitraum einschließlich der möglichen Verschiebung innerhalb des maximalen Zeitraums des Terminangebotszeitraums liegt. Ebenso dürfen sich alle Pausen des Terminangebotszeitraums inklusive der möglichen Verschiebung nicht überschneiden.

Damit für einen Terminangebotszeitraum bekannt ist, für welche Terminserien zu welchen Zeiten Termine angeboten werden sollen und welche Wünsche der anbietende Nutzer dafür hat, wird die Terminserienpräferenz verwendet. Sie gibt an, in welchen Zeiträumen sich eine assoziierte Terminserie buchen lässt. Diese müssen innerhalb des maximalen Zeitraums des Terminangebotszeitraums liegen.
Es darf pro Terminangebotszeitraum mehrere Terminserienpräferenzen geben, solange sie sich nicht auf dieselbe Terminserie beziehen.
Eine Terminserienpräferenz berücksichtigt ebenfalls Vergabewünsche. Diese können sich auf Zeitpunkte oder Zeiträume beziehen oder anmerken, dass eine konkrete Eigenschaft

berücksichtigt werden soll, zum Beispiel der Wunsch nach möglichst wenig Lücken zwischen Terminen. Dabei angegebene Zeitpunkte oder Zeiträume müssen innerhalb des maximalen Zeitraums vom Terminangebotszeitraum liegen, auf den sich die Terminserienpräferenz bezieht.

Es wäre möglich, den Nutzer und Gast zu einer übergeordneten Person zu abstrahieren. Darauf wurde vorerst verzichtet, um die Trennung zwischen der Registrierung als Nutzer und der Angabe von Personeninformationen für eine Buchung (im Interview als Wunsch geäußert (siehe Abschnitt A.1)) zu betonen. Die Abstraktion ist eine zukünftige Verbesserungsmöglichkeit, die insbesondere für einen Kommentar oder eine Angabe nützlich wäre, damit keine Entweder-Oder-Beziehung zum Nutzer oder Gast erforderlich wäre.

3.8 Anwendungsfälle

Im Folgenden wird eine Auswahl von spezifizierten Anwendungsfällen und für die visuelle Begleitung ein Wireframe dargestellt. Weitere Anwendungsfälle sind im Anhang angegeben. Anwendungsfälle werden im Kontext der Identifizierung mit UC abgekürzt.

Erläuterungen und Definitionen

Die feinstgranuläre Einheit für Zeitpunkte und Zeitspannen sollen Minuten sein. Feinere, über die Minutenangabe hinausgehende Zeiteinheiten (Sekunden, Millisekunden...) werden nicht beachtet. Ein Zeitpunkt findet zur vollen Minute statt.

5-Minuten-Granularität: Ein 5-Minuten-granulärer Zeitpunkt muss eine Uhrzeit aufweisen, deren Minutenangabe durch fünf (ohne Rest) teilbar ist. Eine 5-Minuten-granuläre Dauer in Minuten muss analog durch fünf (ohne Rest) teilbar sein.

Vorlaufzeit eines Terminangebotszeitraums: Angabe, wie viele Tage im Voraus ein Termin für einen Terminangebotszeitraum spätestens gebucht werden kann. Beispiel: Termine werden für einen Montag angeboten, die Vorlaufzeit ist auf einen Tag festgelegt. Dann kann ein Termin für Montag bis Samstag, 23:59 Uhr, gebucht werden. Die Buchung wird nicht exakt auf 24 Stunden vor einem möglichen Termin beschränkt, sonst könnten missbräuchlich Termine für eine spätere Uhrzeit durch Abwarten gebucht werden.

Aktiv-Zustand eines Terminangebotszeitraums: Gibt an, ob der Terminangebotszeitraum überhaupt für Buchungen berücksichtigt werden soll. Ist in der Regel aktiv, kann vom Terminanbieter deaktiviert werden.

Puffer: Gibt an, wie viel zusätzliche Zeit zwischen Terminen eingeplant werden muss.

Terminmöglichkeit: Zeitraum eines buchbaren Termins

Allgemeine Fehlerfälle

F-1 Aufruf mit fehlerhaftem Link: System zeigt Fehler an oder navigiert zur Startseite.

F-2 Ein technischer Fehler ist aufgetreten. Anzeige, dass ein Fehler aufgetreten ist.

F-3 Die Eingaben des Nutzers entsprechen nicht den Erwartungen mindestens einer Prüfung, sind nicht vollständig hinsichtlich erforderlicher Felder oder liegen im falschen Format vor: Das System behält die bestehenden Formularangaben bei und weist auf eine Korrektur hin.

Anwendungsfall UC-002: Buchbare Terminmöglichkeiten einer Terminserie einer Person einsehen

Akteur: Student oder externe Person (alias SEP)
Ziel: SEP kennt angebotene Terminmöglichkeiten.
Auslöser: SEP möchte sich informieren, wann eine ausgewählte Terminserie einer Person an einem Tag buchbar ist.
Vorbedingungen: Ein interner Mitarbeiter, der die Terminserien anbieten kann, besitzt einen Account und hat einen gültigen Link für den Zugriff freigegeben. Der interne Mitarbeiter sollte mindestens eine Terminserie in gültigen Terminangebotszeiträumen (siehe Anhang: UC-001: Angebotene Terminserien einsehen) anbieten.
Nachbedingungen: SEP hat sich über Terminmöglichkeiten einer ausgewählten Terminserie einer Person an einem Tag informiert.
Erfolgsszenario:

0 Siehe Schritte des Anwendungsfalls UC-001: Angebotene Terminserien einsehen (Anhang)

1 SEP wählt genau eine Terminserie aus.

2 Das System filtert bestehende Terminangebotszeiträume der anbietenden Person nach Gültigkeit (nicht in der Vergangenheit, genug Vorlaufzeit, aktiv).

3 Das System prüft pro gefiltertem Terminangebotszeitraum (aus Schritt 2), ob ein Termin der Terminserie aus Schritt 1 unter der Berücksichtigung bestehender Termine, Pausen, der Vor-, Nachbereitungszeit und Puffern zeitlich buchbar wäre und sammelt diejenigen mit positivem Ergebnis.

4 Das System zeigt alle Tage der aus Schritt 3 ermittelten Terminangebotszeiträume in chronologischer Reihenfolge zur Auswahl an.

5 SEP wählt einen passenden Tag aus.

6 Das System ermittelt analog zu den Kriterien aus Schritt 3 alle möglichen Terminstarts der ausgewählten Terminserie zu dem ausgewählten Tag (lässt sich zu einem Terminangebotszeitraum zuordnen) von Schritt 5 aus.

7 Das System bewertet alle in Schritt 6 ermittelten Terminstarts anhand bestehender Präferenzen des Anbieters der Termine und ermittelt den/die Terminstart(s) mit der besten Bewertung und wandelt sie zu Terminmöglichkeiten um.

8 Das System zeigt die Terminmöglichkeit(en) mit der besten Bewertung aus Schritt 7 an.

Fehlerfälle:

1-a) Es gibt keine angebotene Terminserie. Keine Auswahl möglich.

5-a) An keinem Tag können Termine gebucht werden. Keine Auswahl möglich.

2,3,6,7-a) Siehe allgemeiner Fehlerfall F-2.

Erweiterungen:

8-a) SEP passt keine der angegebenen Terminbuchungsmöglichkeiten.

8-a-1 Ausführung des Anwendungsfalls UC-003: Einsehen buchbarer Terminmöglichkeiten einer Terminserie einer Person mit einschränkenden Zeiträumen (Anhang)

8-b) SEP möchte eine Terminmöglichkeit buchen

8-b-1 Ausführung des Anwendungsfalls UC-004: Buchung eines Termins

Zugehörige Anforderung(en): US-013
Zugehörige(r) Anwendungsfall/-fälle: UC-001 (Anhang), UC-003 (Anhang)

Anwendungsfall UC-004: Termin buchen

Akteur: Student oder externe Person (alias SEP)
Ziel: Festlegen eines Einzeltermins.
Auslöser: SEP benötigt einen Termin und möchte eine Terminmöglichkeit buchen.
Vorbedingungen: Erfolgreich ausgeführter Anwendungsfall UC-002 bzw. UC-003 (Anhang). SEP hat sich für eine Terminserie, Datum und eine angebotene Terminmöglichkeit entschieden.
Nachbedingungen: Für SEP ist beim Terminanbieter ein Einzeltermin gebucht.
Erfolgsszenario:

0 Siehe Schritte von Anwendungsfall UC-002 oder UC-003 (Anhang)

1 SEP wählt eine Terminmöglichkeit zur Buchung aus.

2 Das System zeigt ein Formular für die Buchung an.

3 SEP füllt das Formular mit den folgenden Daten: Vor-, Nachname, Titel(optional), E-Mail, ob Benachrichtigungen für die Email erlaubt sind, Anliegen(optional), ggf. weitere Daten.

4 SEP wählt die Funktion „Ausgewählten Termin buchen“ aus.

5 Das System prüft die Formularangaben: Sind alle nicht-optionalen Felder angegeben? Stimmen die Formate überein?

6 Das System ermittelt nochmals anhand der anbietenden Person, der Terminserie, dem Datum (und ggf. nicht passenden Zeiträumen des Terminnehmers), passende Terminmöglichkeiten und stellt sicher, dass die zu buchende Terminmöglichkeit darin enthalten ist.

7 Das System bucht den Termin und sendet eine E-Mail-Benachrichtigung, falls erwünscht, an SEP und den Terminanbieter.

8 Das System zeigt SEP eine Bestätigung der Buchung mit Informationen für einen zukünftigen Zugriff an.

Fehlerfälle:

1-a) Keine Terminmöglichkeit steht zur Auswahl bereit. Kein Fortfahren möglich.

5-a) Siehe allgemeiner Fehlerfall F-3.

6-a) Die passendste Terminbuchungsmöglichkeit stimmt nicht mehr mit der Buchungsanfrage überein: Das System weist SEP darauf hin, dass die Terminbuchungsmöglichkeit nicht mehr verfügbar ist.

7-a) Eine E-Mail Benachrichtigung konnte nicht verschickt werden. Fortfahren.

7-b) Siehe allgemeiner Fehlerfall F-2.

Zugehörige Anforderung(en): US-015, US-020, US-021
Zugehörige(r) Anwendungsfall/-fälle: UC-002, UC-003 (Anhang)

Wireframe

Das Wireframe der Abbildung 3.3 zeigt eine vorläufige Benutzeroberfläche zum Zeitpunkt der erfolgreichen Ausführung des Schritts 2 aus dem Erfolgsszenario des Anwendungsszenarios für die Buchung eines Termins (UC-004). Der Student oder die externe Person hat bis zu diesem Zeitpunkt eine Terminserie, ein Datum sowie eine Terminmöglichkeit zur Buchung ausgewählt und das System zeigt ein Buchungsformular an.
Da der Anwendungsfall der Buchung (UC-004) unter anderem auf den Anwendungsfällen zur Einsicht buchbarer Terminserien (UC-001 im Anhang) und Terminmöglichkeiten (UC-002) aufbaut, sind diese mit im Wireframe dargestellt. Vor der Buchung einer Terminmöglichkeit können auch Zeiten angegeben werden, an denen der Terminnehmer nicht kann (UC-003 im Anhang). Das wird in der Darstellung nicht mit abgedeckt, nur durch eine rote Schaltfläche angedeutet.

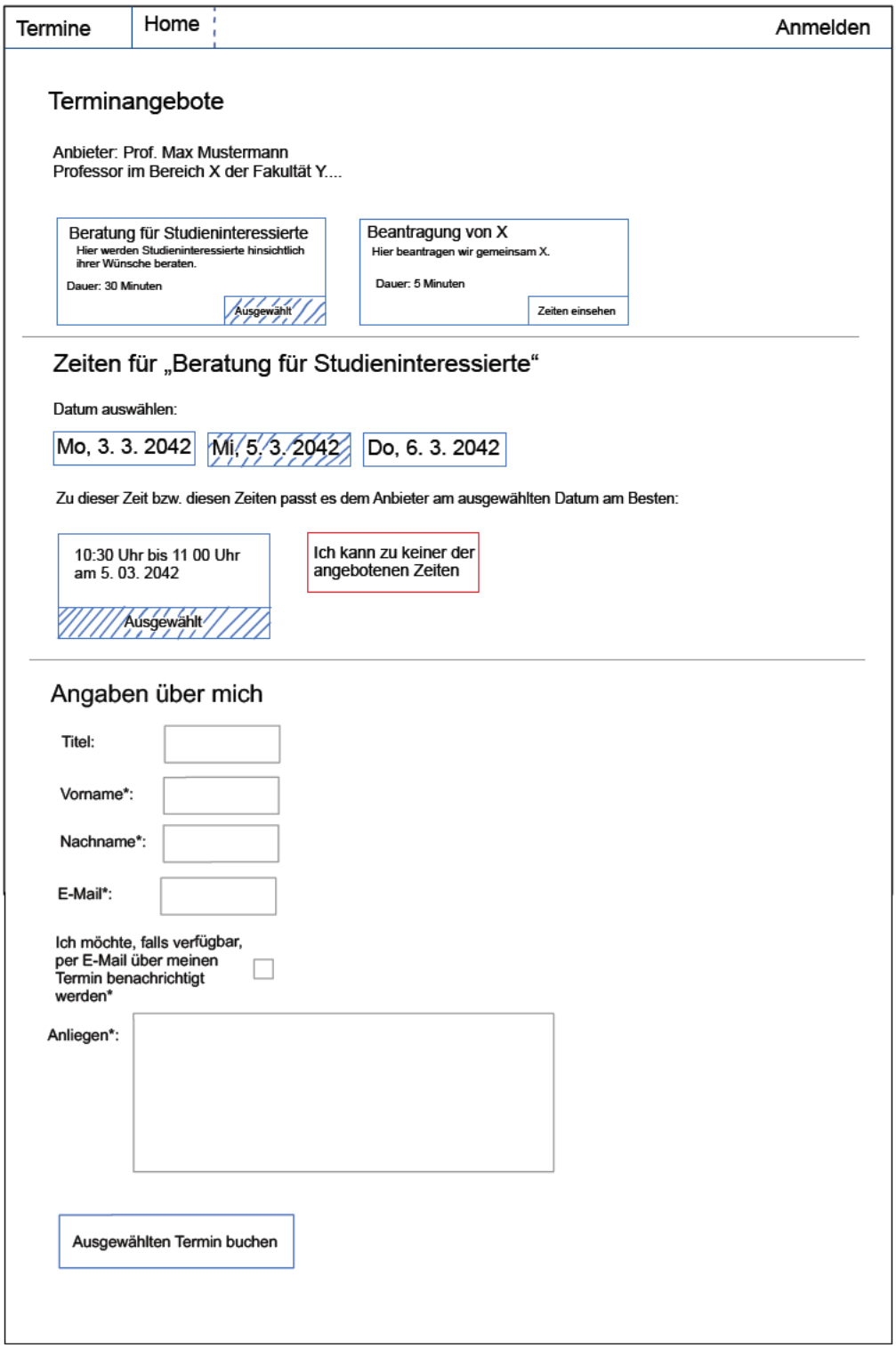

Abbildung 3.3: Wireframe für die Einsicht buchbarer Terminserien, Terminmöglichkeiten und für die Buchung eines Termins

3.9 Systemkategorie

Es werden Systemkategorien nach Starke für die Festlegung einer Systemkategorie genutzt (vgl. [60, Kap. 3.2.2] für alle Informationen zu Systemkategorien dieses Abschnitts).

Zur Auswahl stehen die Systemkategorien „Interaktives Online-System“, „Mobiles System“ „Hintergrundsystem“, „Entscheidungsunterstützungssystem“, „Eingebettetes System “ und „Echtzeitsystem“.

Ein Echtzeitsystem kann ausgeschlossen werden, weil keine strengen Anforderungen hinsichtlich der Dauer der Laufzeit erforderlich sind. Die Performance muss nicht unter allen Umständen garantiert werden.
Die Qualitätsanforderungen der Lauffähigkeit in Docker-Containern (siehe QA-3 und QA-19) sprechen gegen ein eingebettetes System, weil es hardwarenah ist. Docker ermöglicht die virtualisierte Ausführung auf unterschiedlichen Plattformen, somit ist es unwahrscheinlich, dass das zu entwickelnde System hardwarenah ist. Daher wird ein eingebettetes System als Systemkategorie ausgeschlossen.
Ein Entscheidungsunterstützungssystem lässt sich auch ausschließen, weil die Analyse zur Unterstützung von Entscheidungen keine primäre Aufgabe des Systems ist und keine lesenden Zugriffe auf Kopien von Daten stattfinden. Es wird mit aktuellen Daten und Transaktionen gearbeitet (zum Beispiel für die Buchung).
Ein Hintergrundsystem passt als Systemkategorie weniger, weil sich das zu entwickelnde System im Gegensatz zur Systemkategorie nicht hauptsächlich auf die Datenmanipulation bezieht. Laut Starke sei es bei Hintergrundsystemen verbreitet, dass bestehende Datenbestände vor- und nachverarbeitet würden (vgl. [60, Kap. 3.2.2]). Das ist bei dem zu entwickelnden System nicht der Fall.

Zur Auswahl stehen noch das interaktive Online-System und das mobile System.
Das interaktive Online-System bietet eine Benutzeroberfläche und die Verarbeitung von Transaktionen, die im Rahmen der funktionalen Anforderungen erforderlich sind. Eine Benutzeroberfläche ist in der Zielstellung vorgesehen und Transaktionen, zum Beispiel für die Buchung eines Termins (US-015), sind möglich. Zudem wird mit aktuellen Daten gearbeitet und deren Bearbeitung sollte zeitnah erfolgen. Das passt zur Qualitätsanforderung QA-2, die eine zeitnahe Bearbeitung durch eine Maximalreaktionszeit von vier Sekunden in 85% der Fälle voraussetzt.
Die Kategorie des mobilen Systems bietet Eigenschaften analog zum interaktiven Online-System. Zusätzlich werden dort mobile Zugriffe ermöglicht und Backend-Systeme genutzt.

Hierbei wird sich auf die mobilen Systeme wie Smartphones oder Tablets konzentriert. Das zu entwickelnde System ist nicht, wie das mobile System, nur auf Smartphones und Tablets fokussiert, sondern berücksichtigt auch Desktop-Systeme.
Ein mobiles System liegt den Fokus ebenfalls stark auf die Benutzbarkeit und Benutzerfreundlichkeit. Die im Rahmen dieser Arbeit definierten Qualitätsanforderungen berücksichtigen die Benutzbarkeit und Benutzerfreundlichkeit kaum, nur eine Anforderung zum Schutz vor Nutzerfehlern (QA-5).

Als Systemkategorie des zu entwickelnden Systems wurde das interaktive Online-System ausgewählt, weil im Vergleich mit dem mobilen System kein Fokus auf mobilen Systemen sowie auf der Benutzbarkeit und Benutzerfreundlichkeit liegt. Die restlichen Systemkategorien sind nicht geeignet und wurden ausgeschlossen.

3.10 Zusammenfassung

Im Kapitel der Anforderungsanalyse und Spezifikation wurde durch Interviews die Ausgangslage analysiert, wonach die Vereinbarung unter anderem per E-Mail geschehe, dabei störe die hohe Kommunikationsdauer und Verzögerungen, zudem müssten Terminvorschläge freigehalten werden.
Das Konzept des umzusetzenden Systems unterscheidet sich besonders von bestehenden Lösungen hinsichtlich der Terminangebote mit flexiblen Pausen und mehreren, konfigurierbaren Kriterien zur Terminvergabe.
Zielgruppen sind Administratoren, außenstehende Personen, interne Mitarbeiter und Studenten, von den beiden letzteren besteht jeweils ein Persona, die in einem Beispielszenario vorkommen.
Es wurden funktionale Anforderungen und Qualitätsanforderungen sowie Qualitätsattributsszenarien für das System spezifiziert.
Das fachliche Datenmodell umfasst Zusammenhänge unter anderem zwischen einem Nutzer, Einzeltermin mit Gast, einer Terminserie oder einem Terminangebotszeitraum.
Anwendungsfälle und ein Wireframe für funktionale Anforderungen bestehen zum Beispiel für die Buchung eines Termins.
Die ausgewählte Systemkategorie ist ein interaktives Online-System.

Die Anforderungsanalyse und Spezifikation bildet eine Grundlage für das kommende Kapitel, insbesondere den Entwurf und die Evaluation, weil sie Anforderungen und fachliche Zusammenhänge, die für das System zu berücksichtigen sind, beschreibt.

4 Entwurf

Im Kapitel des Entwurfs wird die Systemarchitektur entwickelt. Dafür erfolgt neben der Ermittlung der Benutzeroberflächenart die Festlegung der grundlegenden Architekturstruktur mit einer Aufteilung in Subsysteme. Zudem werden passende Technologien ausgewählt, dafür Referenzarchitekturen bestimmt und die Systemarchitektur in Sichten dargestellt. Die Beschreibung der Sichten erfolgt nach der Vorlage arc42 (vgl. [48], mit Kontextsicht alias Kontextabgrenzung). Darüber hinaus wird die REST-API des Systems für Anfragen beschrieben, und es werden Abbildungsfunktionen für die Bewertungsermittlung von Terminmöglichkeiten ausgewählt.

Der Entwurf berücksichtigt die Spezifikation des vorigen Kapitels, unter anderem funktionale Anforderungen, Qualitätsanforderungen, und das fachlichen Datenmodell. Für die Zielstellung ist der Entwurf zentral, weil er den Aufbau des Systems für die Umsetzung festlegt.

4.1 Grundlegende Struktur der Architektur

Festgelegt wurde, dass das System ein interaktives Online-System ist. Nun ist eine grundlegende Struktur der Architektur zu erarbeiten. Es wird eine vertikale und horizontale Architekturaufteilung ermittelt.

4.1.1 Vertikale Architekturstruktur

Die Architektur soll unter anderem die Wartbarkeit unterstützen. Allgemein sind eine geringe Kopplung und hohe Kohäsion erstrebenswert.

Um die Architektur vertikal zu strukturieren, wird eine Schichtenarchitektur eingesetzt. Die Schichtenaufteilung verbessert die Kohäsion, weil jede Schicht als „zusammenhängende Einheit“[31, S. 17 (übersetzt)] eine eigene, getrennte Zuständigkeit besitzt.

Darüber hinaus erhöht die Schichtenarchitektur die Analysierbarkeit, weil die Suche nach bestimmten Quelltextpassagen anhand der Zuordnung nach Zuständigkeit erleichtert wird.
Ein weiterer Vorteil liegt in der geringeren Kopplung, die Abhängigkeiten zwischen Schichten würden verkleinert (vgl. [31, S. 17]), zum Beispiel mithilfe einer Schnittstelle. Das erhöht die Kapselung, auch weil eine Schicht der meisten Schichtenarchitekturen überwiegend die eigene Nutzung darunterliegender Schichten verstecke (vgl. [31, S. 17]). Es verbessert sich die Austauschbarkeit (und Änderbarkeit), weil die Implementierung einer Schicht durch eine Alternative (hinsichtlich grundlegender Services) ausgetauscht werden könne (vgl. [31, S. 17]).
Darüber hinaus erleichtert die Trennung von Zuständigkeiten der Schichten die Analysierbarkeit, weil sich Teile des Quelltexts anhand der Schichtenzuständigkeit einfacher lokalisieren lassen.

Ein Nachteil bei einer Schichtenarchitektur sei die Verminderung der Performance, weil zwischen mehreren Schichten durchgereichte Inhalte in unterschiedliche Repräsentationen umgewandelt werden müssten (vgl. [31, S. 18]). Zwar besteht eine Qualitätsanforderung für das Zeitverhalten (siehe QA-2), sie ist mit einer Maximaldauer von vier Sekunden in 80 Prozent der Fälle nicht derart restriktiv, sodass der beschriebene Nachteil wenig Gewicht besitzt.
Die Aufteilung in Schichten beeinträchtigt zudem die Änderbarkeit, weil schichtenübergreifende Inhalte bei Änderungen Kaskadierungen mit sich brächten (vgl. [31, S. 18]).

Auswahl einer Schichtenarchitektur

Nun wird eine konkrete Schichtenarchitektur ausgewählt. Dafür stehen hier die 2-Schichten-Architektur sowie die 4-Schichten-Architektur zur Auswahl. Wichtig ist die Berücksichtigung der Trennung von Zuständigkeiten zwischen den Schichten, um die Vorteile der Schichtenarchitekturen bestmöglich auszuschöpfen.

Bei der 2-Schichten-Architektur wird die Benutzeroberfläche der Client-Schicht und die Datenhaltung der Server-Schicht zugeordnet (vgl. [31, S. 18]). Die erforderliche Domänenlogik gehöre entweder zu der Client- oder Server-Schicht (vgl. [31, S. 18]). Die Integration der Domänenlogik innerhalb einer der beiden Schichten vermischt die Zuständigkeiten und führt zu einer verminderten Kohäsion und größeren Kopplung, weil zum Beispiel Code für die Domänenlogik zusammen mit Code für die Benutzeroberfläche gemischt

verwendet werden kann.
Die Änderbarkeit werde verringert, weil die Vermischung der Zuständigkeiten von der Benutzeroberfläche und Domänenlogik die Code-Duplikation begünstige und Änderungen an mehreren Stellen notwendig seien (vgl. [31, S. 18]).
Vorteilhaft ist eine mögliche erhöhte Performance, weil der Datenfluss durch weniger Schichten weniger Repräsentationsumwandlungen erfordert.

Die 4-Schichten-Architektur bietet im Vergleich zur 2-Schichten-Architektur eine stärkere Trennung der Zuständigkeiten. Es findet sich neben der Benutzeroberfläche und Datenhaltung auch die Geschäftslogik in einer eigenen Schicht wieder. Zusätzlich sorgt die Applikationsschicht unter anderem für eine Delegation an die Domänenschicht für die Bearbeitung von Anfragen (vgl. [17, S. 70]). In der Applikationsschicht kann zum Beispiel eine REST-API untergebracht werden.
Aufgrund der höheren Anzahl von Schichten besteht bei der 4-Schichten-Architektur ein Nachteil der verminderten Performance, weil für den Datenfluss zwischen den Schichten mehr Umwandlungen in unterschiedliche Repräsentationen notwendig sind.
Von Vorteil ist die größere Kohäsion innerhalb der vier Schichten im Vergleich zur Client- und Server-Schicht, weil insbesondere die Geschäftslogik von der Domänenschicht nicht mit anderen Zuständigkeiten vermischt wird. Somit können die anderen bereits beschriebenen Vorteile der Schichtenarchitektur, wie die geringe Kopplung, besser ausgeschöpft werden.

Letztlich soll eine 4-Schichten-Architektur und keine 2-Schichten-Architektur genutzt werden, weil die Zuständigkeiten der Schichten damit feingranularer getrennt sind und sich deshalb die Vorteile von Schichtenarchitekturen besser ausschöpfen lassen. Mögliche Unterschiede in der Performance sind weniger relevant als die Wartbarkeit.
Auf eine 3-Schichten-Architektur wurde in der Auswahl aufgrund der fehlenden Applikationsschicht verzichtet, weil der Applikationsschicht insbesondere eine REST-Schnittstelle zugeordnet werden kann.

4.1.2 Horizontale Architekturstruktur

Die Architektur ist vertikal in eine 4-Schichten-Architektur aufgeteilt. Zwar sind die Zuständigkeiten der Schichten jeweils kohäsiv, jedoch besteht keine fachliche Kohäsion auf der horizontalen Ebene. In diesem Abschnitt wird die horizontale Aufteilung der Architektur festgelegt.

Ziel ist es, die fachliche Kohäsion zu vergrößern, die Kopplung zu verringern und die Wartbarkeit zu vergrößern.

Durch eine modulare Aufteilung des Systems mit in sich zusammenhängenden fachlichen Einheiten wird die Kohäsion verbessert. Wie von Haywood beschrieben, würden Module der Kapselung der Funktionalität dienen und die Interaktion restriktieren (vgl. [46]). Das wirke Seiteneffekten bei Änderungen entgegen, die ohne derartige Restriktionen der Interaktion auftreten könnten (vgl. [46]), also wird die Änderbarkeit verbessert. Die modulare Aufteilung verbessere das Verständnis der Zuständigkeiten und der Funktionalität (vgl. [46]). Dieses steigert die Analysierbarkeit.

Es gibt verschiedene Möglichkeiten, wie sich die modulare Aufteilung in der Architektur widerspiegeln kann. Es werden zwei davon betrachtet, darunter eine monolithische Architektur mit zusätzlicher modularer Aufteilung und die Microservice-Architektur.

Ein zentraler Vorteil der Microservice-Architektur bestehe in dem individuellem Deployment und der unabhängigen Skalierbarkeit von den einzelnen Services (vgl. [30]). Man könne Microservices, zum Beispiel bei Änderungen, einzeln neu bereitstellen und deren Skalierung der Auslastung anpassen (vgl. [30]). Eine monolithische Architektur benötige im Vergleich ein ganzheitliches Deployment und müsse als Einheit skaliert werden (vgl. [30]).

Microservices weisen im Vergleich zu Monolithen eine geringere Kopplung auf, weil die Microservices einzeln und unabhängig voneinander (teilweise mit eigener Datenhaltung) bereitstellbar sind und dabei eine prozessübergreifende Kommunikation, zum Beispiel über Webservice-Anfragen, erfolge (vgl. [30]). Während Microservices durch die unabhängige Bereitstellung und die entsprechende Kommunikation eine Schnittstelle, die die Kapselung verbessert, erfordern (oder dazu anregen), sei man bei Monolithen auf die Disziplin der Entwickler angewiesen, die Kapselung aufrecht zu erhalten, (vgl. [30]). Zum Beispiel lässt sich beim Monolithen mit der Ausführung im selben Prozess direkt eine Methode einer konkreten Klasse aufrufen, anstelle eine explizite Schnittstelle zu nutzen.

Laut Fowler sei das Anfangen mit einer monolithische Architektur sinnvoller, weil Flexibilität für die modulare Aufteilung bestehe (vgl. [29]). Für eine Veränderung der modularen Aufteilung von Funktionalität sei das Refactoring mit Monolithen einfacher als mit Microservices (vgl. [29]), also ist die Änderbarkeit erhöht.
Darüber hinaus kann ein Monolith die Entwicklung vereinfachen, weil Monolithen einen stärkeren Konsistenzgrad als Microservices ermöglichen würden (vgl. [30]). So würden

Microservices über eine dezentrale Datenverwaltung verfügen und Transaktionslosigkeit betonen, wobei Eventual Consistency möglich sei (vgl. [30]).
Der Verwaltungsaufwand sei bei einer monolithischen Anwendung geringer als bei Microservices, weil bei Microservices zusätzliche Verwaltungsaufgaben, z.B. Service-Discovery, erforderlich seien (vgl. [46]).

Sowohl die monolithische Architektur als auch die Microservice-Architektur bieten eine Aufteilung, um die fachliche Kohäsion zu vergrößern.
Aufgrund der prototypischen Neuentwicklung des Systems ist die modulare Aufteilung nicht genügend erprobt, weshalb diese im Rahmen der weiteren Entwicklung verändert werden könnte. Daher ist der Vorteil der Änderbarkeit des Monolithen für die Architekturauswahl besonders relevant.
Das individuelle Deployment und die Skalierbarkeit von Microservices können für den Betrieb mit vielen Nutzern zwar hilfreich sein, darauf liegt im Rahmen der prototypischen Entwicklung jedoch kein Fokus. Dem Vorteil der geringeren Kopplung stehen Nachteile des erhöhten Verwaltungsaufwands und der erschwerten Entwicklung aufgrund des geringeren Konsistenzgrads entgegen. Dieser Zusatzaufwand besteht bei einer monolithischen Architektur nicht, weshalb dort potenziell mehr funktionale Anforderungen umgesetzt werden können.
Letztlich überwogen bei der Auswahl die Vorteile der monolithischen Architektur diejenigen der Microservice-Architektur für die prototypische Entwicklung, somit wird eine monolithische Architektur (mit zusätzlicher modularer Aufteilung) verwendet.

4.2 Art der Benutzeroberfläche

Bisher steht fest, dass das System der Kategorie „Interaktives Online-System“ angehört. Dafür ist auch eine Benutzeroberfläche vorgesehen. Dieser Abschnitt klärt, welche Art der Benutzeroberfläche für das System eingesetzt werden soll.

Es werden zwei Möglichkeiten betrachtet: eine native Anwendung mit einer Benutzeroberfläche auf dem System des Endnutzers, zum Beispiel Desktop- oder Smartphoneanwendungen, oder eine Webanwendung, die mittels eines Webbrowsers aufrufbar ist. Eine mögliche Entscheidung für eine native Anwendung bezieht sich hier ausschließlich auf die Benutzeroberfläche (Präsentationsschicht). Das heißt, dass andere Teile des Systems, die nicht zur Benutzeroberfläche gehören, dann nicht zwingend der nativen Anwendung

zugehören. Für den Vergleich wurde eine Quelle verwendet ([16]), die sich auf den Vergleich zwischen Web- und Desktopanwendungen konzentriert. Eigenschaften von Desktopanwendungen werden in der folgenden Beschreibung abweichend von der Quelle auch allgemein auf native Anwendungen bezogen.

Für die native Anwendung spricht, dass sie mehr Zugriff auf systemspezifische Informationen, wie einen Kalender oder Adressbücher, ermöglichen kann (vgl. [16]), die für die Terminvermittlung hilfreich sind. Bei einer Webanwendung sei der Zugriff auf solche Information über die Hardware nicht möglich (vgl. [16]).
Eine native Anwendung bietet den Vorteil des weniger aufwendigen Hostings, weil eine native Anwendung lokal installiert werde und im Vergleich Webanwendungen Hosting benötigen würden (vgl. [16]). Zum Beispiel müssen dafür Webserver betrieben werden.

Die Wartung der Webanwendung wird erleichtert, weil Aktualisierungen serverseitig vollzogen werden könnten, bei nativen Anwendungen muss die lokale Installation aktualisiert werden, das geschehe meistens bei Desktop-Anwendungen durch zusätzliche Installationen einer manuellen Bestätigung (vgl. [16]).
Eine Webanwendung sei sofort und ohne Installation verfügbar, während diese bei einigen nativen Anwendungen wie Desktopanwendungen erforderlich sei (vgl. [16]). Zum Beispiel müsste die Webanwendung nur mithilfe einer URL aufgerufen werden, während Apps für Smartphones installiert werden müssen.
Darüber hinaus sei die Webanwendung im Vergleich unabhängiger vom Betriebssystem bzw. von den Endgeräten einsetzbar, weil nur ein Webbrowser mit einer Internetverbindung benötigt werde, während native Anwendungen an das jeweilige Betriebssystem angepasst werden müssten (vgl. bezogen auf die Desktopanwendung [16]).

Das Aufwand-Nutzen-Verhältnis der Entwicklung (und Wartung) ist für Webanwendungen größer, weil eine einzige entwickelte Webanwendung alle Endgeräte mit einem Webbrowser abdecke (vgl. [16]). Desktopanwendungen würden eine betriebssystemspezifische Entwicklung erfordern (vgl. [16]), analog für Smartphone-Apps.

Letztlich wurde als Art der Benutzeroberfläche die Webanwendung ausgewählt, weil insbesondere für die prototypische Entwicklung ein hohes Aufwand-Nutzen-Verhältnis vorteilhaft ist, um möglichst viele funktionale Anforderungen zu entwickeln. Der Zugriff auf Informationen bei nativen Anwendungen durch die Hardware wäre hilfreich, um Informationen zu bestehenden Termine zu erhalten, jedoch können abhängig vom Betriebssystem unterschiedliche Schnittstellen dafür bestehen. Der Aufwand, mit allen Betriebssystemen kompatibel zu sein, wäre hoch. Hosting-Nachteile der Webanwendungen können aufgrund

der sofortigen, verbreiteten Nutzung und erleichterten Aktualisierung vernachlässigt werden.

4.3 Aufteilung in Subsysteme

Dieser Abschnitt legt fest, wie die Aufteilung in Subsysteme bezüglich der Schichten aussehen soll. Es werden im Folgenden die Möglichkeiten keiner weiteren Aufteilung oder einer Aufteilung zwischen der Präsentationsschicht und der Applikationsschicht, wie in Abbildung 4.1 dargestellt, betrachtet.
Eine Aufteilung zwischen der Präsentationsschicht und der Applikationsschicht war im Vergleich zur Aufteilung zwischen anderen Schichten besonders attraktiv, weil durch die Applikationsschicht eine HTTP-REST-API realisiert werden kann, die die Entkopplung erleichtert.

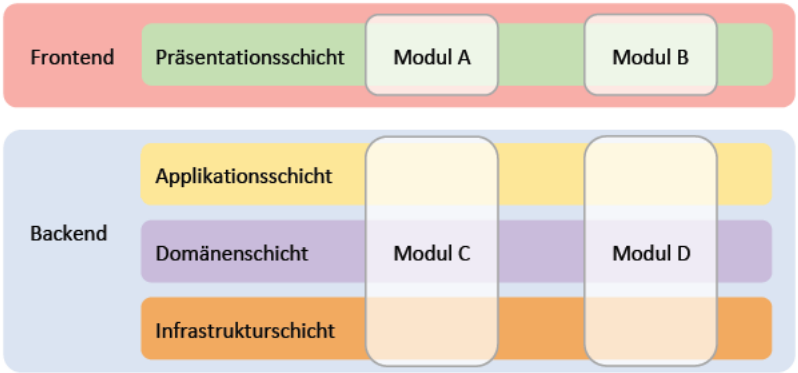

Abbildung 4.1: Architektur nach einer Systemaufteilung zwischen der Präsentations- und Applikationsschicht

Im Folgenden werden im Falle einer Systemaufteilung die Subsysteme Frontend (Präsentationsschicht) und Backend (restliche, darunterliegende Schichten) genannt, wie in Abbildung 4.1 dargestellt. Zudem wird angenommen, dass dabei eine Schnittstelle vom Backend zum Frontend erforderlich ist, zum Beispiel eine REST-Schnittstelle.

Gegen eine Aufteilung würde sprechen, dass der Betriebs- und Installationsaufwand mit einer Aufteilung in Subsysteme größer wäre, weil mehrere Subsysteme anstelle eines einzelnen Systems bestünden.
Zudem können Daten ohne eine Aufteilung in Subsysteme schneller in die Benutzeroberfläche integriert werden, weil sie nicht über die Grenzen der Subsysteme hinaus trans-

portiert werden müssen, sondern zum Beispiel durch einen einfachen Methodenaufruf innerhalb eines Prozesses übergeben werden können. Ein Beispiel wäre eine Webseite, deren Inhalte serverseitig zusammengestellt werden, darunter auch fachliche Inhalte wie angebotene Termine.

Die Aufteilung in zwei Subsysteme verstärkt die Trennung der Verantwortlichkeiten (Separation of Concerns), die bereits durch die 4-Schichten-Architektur angestrebt wird, weil die Zuständigkeit der Benutzeroberfläche (Präsentationsschicht) in das Frontend ausgelagert wird und die Systemgrenzen die Trennung verstärken.
Mit der Trennung der Verantwortlichkeiten geht die Erhöhung der Kohäsion innerhalb der Subsysteme einher. Es bestünde dann kein einzelnes System, das sowohl für die Benutzeroberfläche als auch für die Fachlogik zuständig ist, sondern das Frontend ist für die Benutzeroberfläche und das Backend (zusammengefasst) für die Bereitstellung der fachlichen Funktionalität zuständig.

Die Kopplung zwischen dem Frontend und dem Backend ist geringer, weil die Subsysteme getrennt sind und eine (klar definierte) Schnittstelle, wie eine REST-API, erforderlich ist.
Mit der geringen Kopplung wird auch der Einsatz unabhängiger Technologien in den Subsystemen ermöglicht. Die einzige Voraussetzung ist die Einhaltung der Schnittstellen zwischen den Subsystemen.
Die Wartbarkeit verbessert sich ebenfalls durch eine Aufteilung, weil sich beide Subsysteme unabhängig voneinander entwickeln lassen. Zum Beispiel müsste für eine Verbesserung der Benutzeroberfläche nur das Frontend aktualisiert werden, nicht aber das Backend.
Darüber hinaus steigt die Flexibilität von Benutzeroberflächen und die Wiederverwendbarkeit des Backends, denn es lassen sich potenziell weitere Frontend-Systeme parallel betreiben, die auf dasselbe Backend-Subsystem zugreifen.

Letztlich ist die Aufteilung zwischen der Präsentationsschicht und der Applikationsschicht in zwei Subsysteme (Frontend und Backend) sinnvoll, weil die Kopplung verringert und die Trennung der Zuständigkeiten sowie die Kohäsion verbessert wird. Insbesondere die Möglichkeit der Verwendung unabhängiger Technologien sorgt für Flexibilität in der Entwicklung.
Die Vorteile einer Trennung überwiegen. Die Argumente für den Verzicht auf eine Aufteilung lassen sich entkräften. Der höhere Betriebs- und Installationsaufwand ist verkraftbar. Die Möglichkeit des schnelleren Einsetzens von Daten in die Benutzeroberfläche kann auch Nachteile für die Performance bringen, weil es (abhängig von der Verteilung) sein

kann, dass die (serverseitig) zusammengestellte Webseite bei Anfragen komplett geladen werden müsste, anstelle nur wenige benötigte Daten bei Nutzung des Systems nachzuladen bzw. anzufragen, wenn z.B. ein Frontend-Subsystem im Webbrowser ausgeführt würde und angebotene Termine ermittelt werden sollen.

Für den Fall, dass für das Frontend Client-seitige Ausführungen im Webbrowser erfolgen, zum Beispiel mithilfe von JavaScript, muss die Einschränkung, dass das Frontend ausschließlich für die Präsentationsschicht zuständig ist, aufgeweicht werden. Es können Vorgänge im Frontend erforderlich sein, die sich nicht eindeutig der Präsentationsschicht zuordnen lassen, wie zum Beispiel die lokale, teils fachliche Validierung von Nutzereingaben.

4.4 Technologieauswahl

Zuletzt wurde festgelegt, dass das System in zwei Subsysteme, das Frontend und Backend, aufgeteilt wird.
Für die Kommunikation zwischen Frontend und Backend soll eine REST-API (Level 2) nach dem Richardson Maturity Model zum Einsatz kommen, damit die Schnittstelle verständlicher und analysierbarer wird. Der Einsatz dieser Schnittstelle findet sich zudem in der Qualitätsanforderung QA-4 wieder.

Frontend

Das Frontend soll unter anderem für die Benutzeroberfläche zuständig sein. Daher ist es wichtig, dass die eingesetzte Technologie eine Benutzeroberfläche bietet. Es sollten HTTP- bzw. REST-Anfragen an das Backend gestellt werden können.

Zur Auswahl stehen die Technologien React und Angular.

Sowohl die React-Bibliothek als auch die Angular-Entwicklungsplattform (auch Framework) bieten eine Benutzeroberfläche (vgl. [21] und [39]). HTTP-Anfragen können mit Angular-zugehörigen Bibliotheken durchgeführt werden, bei React sind fremde Bibliotheken erforderlich (vgl. [39] und [18]). Da beide Technologien sowohl eine Benutzeroberfläche als auch HTTP-Anfragen ermöglichen, kommen sie infrage.

Im Vergleich zum Angular-Framework bietet die React-Bibliothek den Vorteil, dass sie nicht auf eine Single-Page Application beschränkt ist und auch als Teil einer Webseite einsetzbar ist, die Koexistenz mit anderen Lösungen ist dadurch besser (vgl. [20]).
Darüber hinaus besteht bei React das Prinzip, die Bibliothek kompakt zu halten und auf Funktionalität zu verzichten, die vom Nutzer selbst implementiert werden kann (vgl. [19]). Als Erweiterungen lassen sich externe Bibliotheken einbinden (vgl. [21]). Das hat den Vorteil, dass React keine überflüssige Funktionalität mit sich bringt und man sie bei Bedarf erweitern kann.

Angular bietet bidirektionales Data Binding, während es bei React unidirektional ist (vgl. [37] und [24]). Das bidirektionale Data Binding ermöglicht die Bindung von Anwendungsdaten an Eigenschaften von HTML-Elementen sowie das Reagieren auf Events für diese Elemente (vgl. [38]).
Generell ist es hilfreich, dass Angular Werkzeuge für den Build und das Testen bereitstellt, weil diese auch für die Entwicklung und Auslieferung benötigt werden (vgl. [39]).
Ein Vorteil von Angular liegt darin, dass es für die Entwicklung eine Sammlung von Angular-eigenen Bibliotheken bereitstellt, zum Beispiel für HTTP-Anfragen oder für Formulare, und somit kaum weitere externe Bibliotheken von Drittanbietern erforderlich sind (vgl. [39]). Das führt zu einer Zeitersparnis im Vergleich zu React, weil man für entsprechende benötigte Funktionalität direkt Lösungen von Angular nutzen kann, während man bei React auf externe Bibliotheken angewiesen ist, die ausgewählt werden müssen.
Da bereits Erfahrung mit Angular besteht, verringert sich die Einarbeitungszeit im Vergleich zu React.

Letztlich überwiegen die Vorteile für Angular. Die größere Auswahl an zu Angular gehörenden Bibliotheken, die Entwicklungswerkzeuge für Builds und Tests sowie die bestehende Erfahrung sind ausschlaggebend. Da das Frontend neu entwickelt wird und nicht bereits als Webseite besteht, ist die Koexistenz von React mit anderen Lösungen bzw. die Einbindemöglichkeit als Teil einer Webseite nicht nötig.

Backend

Für das Backend ist es wichtig, dass es eine REST-Schnittstelle bereitstellt und die Speicherung von Daten ermöglicht bzw. zumindest auf eine Schnittstelle einer Persistenzlösung zugreifen kann.

Als Technologie für das Backend wurde das Spring Framework ausgewählt.
Vorteilhaft für das Backend sind umfassende Erweiterungsmöglichkeiten (vgl. [64]), um flexibel auf Änderungswünsche der Anwender reagieren zu können.
Zudem vereinfacht Dependency Injection von Spring (vgl. [64]) die Entwicklung, weil sich der Anwendungscode nicht aktiv um das Einsetzen der Abhängigkeiten kümmern muss. Durch Dependency Injection können auch bei Tests Abhängigkeiten speziell konfiguriert werden (vgl. [55]).
Es ist außerdem weniger Einarbeitungsaufwand erforderlich, weil bereits eigene Erfahrung mit dem Framework besteht.
Ein integrierter Webserver bietet die Basis für eine erforderliche REST-Schnittstelle (vgl. [65]). Anforderungen des Systems hinsichtlich der Anmeldung von Nutzern werden schon durch Sicherheitsfunktionalität des Frameworks unterstützt (vgl. [68]).
Für die Infrastrukturschicht vereinfacht Spring die Entwicklung, weil Objektabbildungen möglich sind und Repositories abstrahiert sind (vgl. [66]). Aus Methodennamen können Anfragen dynamisch abgeleitet werden (vgl. [66]). Außerdem wird die benötigte Datenspeicherung durch eine Anbindung von Datenspeicherungstechnologien, wie relationale Datenbanken, ermöglicht (vgl. [66]).

Ausschlaggebend für die Entscheidung, Spring Framework zu nutzen, sind die Möglichkeiten des Angebots einer REST-Schnittstelle, der Anbindung an eine Datenspeicherungstechnologie und die geringere Einarbeitungszeit.

Als Datenspeicherungstechnologie wurde die relationale Datenbank ausgewählt, weil die Daten eine festgelegte Struktur aufweisen, die aus dem fachlichen Datenmodell abgeleitet werden kann. Zudem sind die für relationale Datenbanken bestehenden ACID-Eigenschaften wichtig, insbesondere die Atomarität, weil Transaktionen, wie die Buchung eines Termins, nur vollzogen werden sollen, wenn keine Fehler auftreten. Relationale Datenbanken werden auch von Spring unterstützt (vgl. [66]).

Für die lokale Entwicklung und Tests soll eine H2-Datenbank als relationale Datenbank zum Einsatz kommen, weil sie quelloffen ist, eingebettet werden kann und neben einer Speicherung als Datei auch die In-Memory-Nutzung, zum Beispiel für die Ausführung von Tests, ermöglicht (vgl. [45]). Insbesondere die In-Memory-Nutzung sollte nur für Testzwecke eingesetzt werden, weil sie nicht ausreichend gegen einen Hardwareausfall abgesichert ist.
Für die Bereitstellung wird als relationale Datenbank eine externe PostgreSQL-Datenbank eingesetzt, weil sie quelloffen und kostenlos ist (vgl. [63]). Zudem sei sie konform zu den

ACID-Eigenschaften (vgl. [63]).
Die Datenbank soll nicht im Backend eingebunden, sondern extern angebunden werden, um sie getrennt betreiben, verwalten und warten zu können. So können beispielsweise für die Datenbank besondere, hochverfügbare Umgebungen eingesetzt werden, die für das Backend alleine nicht erforderlich wären. Darüber hinaus ist die Konfigurierbarkeit verbessert, weil je nach Konfiguration eine andere PostgreSQL-Datenbank angebunden werden kann und diese dann kein fester Teil des Backends sein muss.

4.5 Referenzarchitekturen

Frontend

Die Referenzarchitektur im Frontend orientiert sich an einer von Pietrucha vorgestellten Schichtenarchitektur für Angular (vgl. [54], siehe Grundlagen) und greift das Konzept der Page-Pattern-Umsetzung für Angular von Parlakov (vgl. [53]) auf.

Die zuvor dem Frontend zugeordnete Präsentationsschicht spiegelt sich im Einsatz von Angular-Components wider, die die Benutzeroberfläche bereitstellen.
Es werden hier die von Pietrucha kategorisierten „smart components“ und „dumb components“ eingesetzt (vgl. [54]), mit dem Unterschied, dass bei „dumb components“ Benutzeraktionen mittels Callback-Aufrufen statt Events signalisiert werden und dass sich diese „dumb components“ einander hierarchisch einsetzen können. Als „dumb components“ werden im Frontend auch Formulare eingesetzt, die für die Validitätsprüfung von Eingaben oder deren Umwandlung in Klasseninstanzen autonom agieren dürfen. Nach wie vor muss das Formular bei Bestätigung und validen Eingaben die Eingaben mithilfe eines Callback-Aufrufs an die übergeordnete Angular-Component übermitteln.

Die Benutzeroberfläche ist in Seiten nach der Page-Pattern-Umsetzung von Parlakov strukturiert, weil dem Endnutzer verschiedene aufrufbare Seiten für unterschiedliche Funktionalitäten präsentiert werden müssen, zum Beispiel für die Erstellung eines Einzeltermins (vgl. [53]).
Eine konkrete Seite wird innerhalb der Präsentationsschicht durch eine Angular-Component als „smart component“ repräsentiert. Sie ist dafür zuständig, die Benutzeroberfläche für eine Seite bereitzustellen. Dafür können zusätzlich „dumb components“ eingebunden (und wiederverwendet) werden.

Damit die „smart component“ einer Seite Daten für die Anzeige erhält und Benutzeraktionen delegieren kann, wird ihr ein für die Seite zuständiger Angular-Service bereitgestellt. Der Seiten-Service setzt gemäß der Architektur von Pietrucha eine Fassade um, indem er den Zugriff auf Daten für die Seite mithilfe von Observables (bzw. Streams) an Klassen delegiert, die die Zustände kapseln (vgl. [54]). Zudem delegiert er Benutzeraktionen unter anderem an die REST-API des Backends weiter und sorgt dafür, dass die Zustände durch Informationen aus der Antwort geändert werden (vgl. [54]).
Da jede Instanz eines Seiten-Services einen individuellen Zustand einer Seite enthält, wie geladene Terminangebote, sind die Seiten-Services zustandsbehaftet und entsprechen deshalb nicht der Service-Definition von Evans (vgl. [17, S. 105-106]). Die Benennung „Seiten-Service“ wird im Kontext des Frontends beibehalten, weil die Seiten-Services Angular-Services sein sollen.

Der Seiten-Service entspricht einer „Page“ gemäß der Page-Pattern-Umsetzung von Parlakov, weil er für die Seite (ohne Benutzeroberfläche), auch für die Bereitstellung der anzuzeigenden Daten, zuständig ist und an ihn Nutzeraktionen und -eingaben delegiert werden können (vgl. [53]). Abweichend von dem Konzept der Abstraktionsschicht von Pietrucha und dem Facade Pattern, soll zusätzlich Geschäftslogik im Seiten-Service enthalten sein, die in Zusammenhang mit Nutzerinteraktionen der Seite benötigt wird (vgl. [53]). Das ist zum Beispiel die Entscheidung, ob weitere Aktionen nach einer erfolgten Aktion (zum Beispiel das erneute Abrufen angebotener Termine nach einer Buchung) erfolgen sollen.

Im Frontend ist die Kernschicht nach Pietrucha integriert (vgl. [54]). Darin sind State-Klassen, die hier einen Zustand für Seiten-Services kapseln, enthalten (vgl. [54]). In der Kernschicht des Frontends befinden sich außerdem Services, die den Zugriff auf externe Schnittstellen ermöglichen (vgl. [54]). Ein Beispiel für eine externe Schnittstelle ist die REST-API des Backends. Die State-Klassen und Services für den Schnittstellenzugriff in der Kernschicht werden von Seiten-Services der Abstraktionsschicht genutzt.

Es ist noch darauf hinzuweisen, dass zusätzlich im Frontend eine Validierung der Nutzereingaben erfolgt, ohne dafür auf das Backend zugreifen zu müssen. Gründe dafür sind ein verbessertes Zeitverhalten und das frühzeitige Hinweisen auf Fehleingaben (für QA-5). Dafür erforderliche Fachlogik, die streng genommen der Domänenschicht des Backends zuzuordnen ist, soll als Ausnahme auch im Frontend für die Validitätsprüfungen verwendet werden können.

Zusammenfassend wird das Frontend in die Präsentationsschicht, die Abstraktionsschicht und die Kernschicht aufgeteilt, wobei die Strukturierung nach Seiten mitberücksichtigt wird.

Backend

Die Architektur des Backends soll sich an den Schichten (Applikationsschicht, Domänenschicht und Infrastrukturschicht) orientieren.

In der Infrastrukturschicht werden für den Zugriff auf Persistenz abstrahierende Repository-Interfaces eingesetzt. Damit wird das Repository Pattern umgesetzt. Neben der Definition des Interfaces mit Methoden ist keine konkrete Implementierung erforderlich. Sie wird von Spring Data bereitgestellt, aus den Namen der Methoden werden die Operationen automatisch abgeleitet (vgl. [66] und [67]).
Im Backend wird die Domäne innerhalb der Domänenschicht durch Entitäten und zugehörige Wertobjekte (Value Objects) repräsentiert. Zusätzlich bestehen in der Domänenschicht Services. Laut Evans würden gute Services unter anderem zustandslose Operationen ermöglichen, die zum Domänenkonzept, aber zu keiner Entität und zu keinem Wertobjekt gehören (vgl. [17, S. 105]). Darauf soll bei der Entwicklung Rücksicht genommen werden, hier sollen auch mehrere Operationen von einem Service angeboten werden können. Eine derartige Operation kann zum Beispiel der Buchungsvorgang sein, für den überprüft werden muss, ob der zu buchende Termin noch angeboten wird und bei dem anschließend ein Termin erstellt wird.
Für die Durchführung einer Operation kann ein Service mit Entitäten und Wertobjekten interagieren und Aufgaben an andere Services derselben Schicht delegieren. Um die Entitäten zu persistieren bzw. persistierte Entitäten abzurufen, sollen die Services auch auf Schnittstellen der Infrastrukturschicht (Repository-Interfaces) zugreifen können. Ein Service soll Transaktionsverwaltung nutzen können, um Vorgänge als Einheit durchzuführen und bei auftretenden (technischen oder fachlichen) Fehlern die Transaktion rückgängig zu machen.

Für die Referenzarchitektur war es schwierig, festzulegen, inwiefern diese beschriebenen Services, die hier der Domänenschicht zugeordnet wurden, der Domänen- oder Applikationsschicht angehören. Diese Services würden auch Aufgaben der Software definieren und für deren Bearbeitung auf Domänenobjekte zurückgreifen, zum Beispiel bei der Erstellung eines Termins, was laut Evans zu der Applikationsschicht gehöre (vgl. [17, S. 70])).

Jedoch können die Services auch Geschäftslogik berücksichtigen und Zustände (der Geschäftssituation) durch Delegierung an die Infrastrukturschicht verwalten, das gehöre zur Domänenschicht (vgl. [17, S. 70]). Als Beispiel für Geschäftslogik wird für die Buchung die Geschäftsregel berücksichtigt, dass der zu buchende Termin noch angeboten wird.
Obwohl diesen Services einige Zuständigkeiten der Applikationsschicht zugeordnet werden können, wurden sie insgesamt der Domänenschicht zugeordnet, weil insbesondere weitere Geschäftslogik für die Bearbeitung der Aufgaben erforderlich sein kann und Zustände mithilfe der Delegierung an die Infrastrukturschicht verwaltet werden können. Eine zukünftige Verbesserungsmöglichkeit besteht darin, diese Services stärker anhand der Zuständigkeiten zwischen der Applikations- und Domänenschicht in separate Services bzw. Klassen aufzutrennen.

Damit das Backend für das Frontend eine HTTP-REST-Schnittstelle anbieten kann, bestehen in der Applikationsschicht RestController-Klassen, die REST-Schnittstellen als Fassaden (nach dem Facade Pattern) anbieten. Dadurch kapseln sie die innere Struktur des Backends nach außen und sorgen so für eine geringere Kopplung. Sie definieren analog zu den Services in der Domänenschicht Aufgaben des Systems und delegieren die Anfragen zur Bearbeitung an diese Services.
Der Einsatz von DTOs für die REST-Schnittstelle sorgt für eine größere Kapselung der internen Datenstruktur. Die RestController-Klassen müssen für die Delegierung an die Domänenschicht eine Umwandlung zwischen den DTOs und der Backend-internen Datenstruktur durchführen, auch umgekehrt für Rückgaben. Entsprechend der REST-Schnittstelle müssen die RestController-Klassen auch einen HTTP-Code in die Antwort integrieren, der einen Erfolgs- oder Fehlerfall signalisiert. Der Domänenschicht bleibt durch die Umwandlungen verborgen, dass überhaupt eine REST-API mit entsprechenden Datentypen angeboten wird.

4.6 Kontextsicht

Im Folgenden wird der fachliche und technische Kontext des Systems beschrieben.

System Im Zentrum steht das zu entwickelnde System zur Terminvermittlung (siehe Abbildung 4.2 und 4.3).

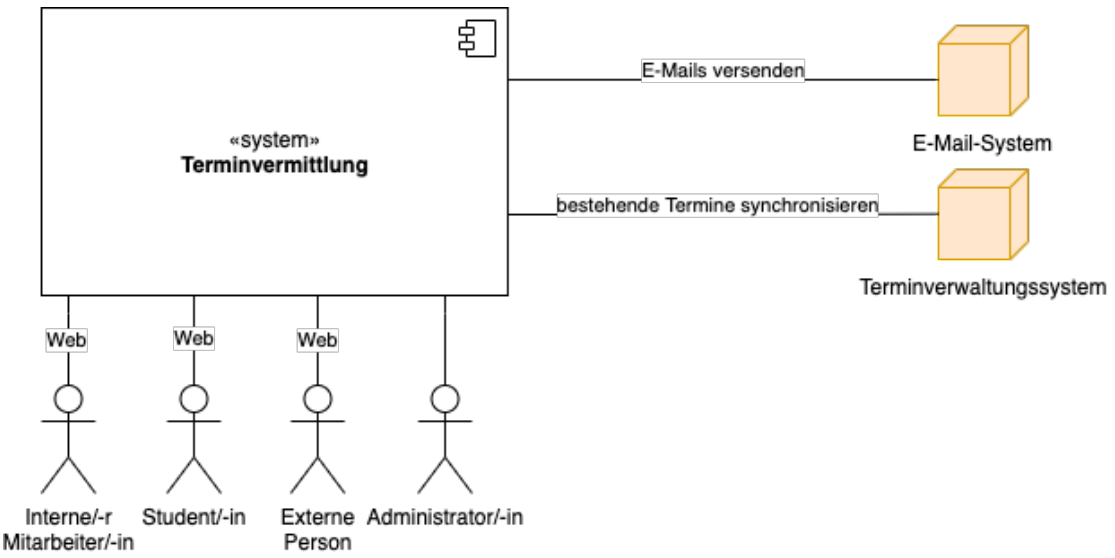

Abbildung 4.2: Fachliches Kontextdiagramm
Akteure werden in UML-Notation und Fremdsysteme als Würfel dargestellt. Für farblich hinterlegte Fremdsysteme wurde die Anbindung im Rahmen der prototypischen Entwicklung nicht umgesetzt. Die Verbindungslinien symbolisieren die Schnittstellennutzung aus fachlicher Sicht.

Um die fachlichen Funktionalitäten des Systems zu nutzen, können interne Mitarbeiter, Studenten und externe Personen damit interagieren. Diese Akteure greifen über das Web auf das System mithilfe eines Webbrowsers zu.
Aus fachlicher Sicht ist die Schnittstelle kritisch, weil ohne sie die entsprechenden Akteure keine Funktionalität in Anspruch nehmen können. Da unter anderem personenbezogene Daten und Anmeldedaten übertragen werden können, ist die Schnittstelle aus Sicht der Vertraulichkeit (siehe QA-9) und indirekt aus Sicht der Integrität (wenn abgehörte Anmeldedaten zur Veränderung von Daten genutzt werden) (siehe QA-11) ebenfalls kritisch. Deshalb sollte die Schnittstelle gesichert sein. Durch einen Ausfall der Schnittstelle wäre das gesamte System nicht nutzbar.
Die Schnittstelle wird im technischen Kontext mit HTTP umgesetzt. Eine entsprechende Konfiguration ermöglicht die Nutzung einer sicheren Verbindung mit HTTPS. Eine sichere Verbindung wird im Rahmen der prototypischen Entwicklung nicht eingesetzt.

Administratoren greifen für die Wartung ins System ein, z.B. schauen sie sich bei Bedarf Logs an oder starten das System neu. Dabei werden unterschiedliche Schnittstellen genutzt. Da ggf. auch auf personenbezogene Daten und Anmeldedaten zugegriffen werden kann, sollten die Schnittstellen im Rahmen der Möglichkeiten vor fremden Zugriff und

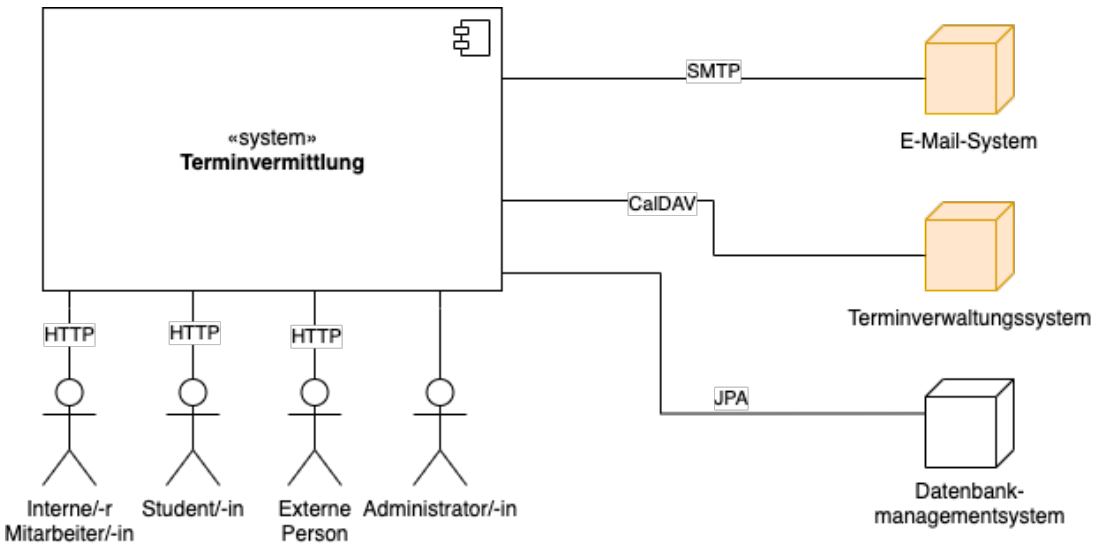

Abbildung 4.3: Technisches Kontextdiagramm
Die Darstellung erfolgt analog zur Abbildung 4.2, mit dem Unterschied, dass Verbindungslinien die Nutzung von Schnittstellen mit konkreten Protokollen symbolisieren.

fremder Einsicht geschützt sein. Diese Schnittstellen sind weniger kritisch, weil sie nur im Rahmen der Wartung benötigt werden.

Fremdsysteme Fremdsysteme sind im fachlichen und technischen Kontext (Abbildung 4.2 und 4.3) das E-Mail-System und das Terminverwaltungssystem. Sie wurden im Rahmen der Arbeit nicht angebunden. Das externe Datenbankmanagementsystem besteht nur im technischen Kontext und ist am System angebunden. Die Stabilität der Schnittstellen ist unklar, weil sie abhängig von den Garantien des jeweiligen Anbieters ist.

Mithilfe des externen E-Mail-Systems kann das Terminvermittlungssystem Benachrichtigungen als E-Mails an interne Mitarbeiter, Studenten oder externe Personen versenden, solange dies erwünscht ist. Beispielsweise wäre als Benachrichtigung die Bestätigung für eine erfolgreiche Terminbuchung möglich.
Die Schnittstelle ist funktional. Aus fachlicher Sicht ist die Schnittstelle unkritisch, weil sie nur für Benachrichtigungen eingesetzt wird, und die wichtigsten fachlichen Vorgänge (z.B. die Erstellung/Buchung eines Termins) ohne eine Benachrichtigung auskommen können. Falls E-Mails versendet werden, damit ein Akteur zum Beispiel Anfragen bestätigt, würde sie fachlich kritisch sein. Daher sollte das Terminvermittlungssystem fehlertolerant gegenüber eines vorübergehenden Ausfalls des E-Mail-Systems sein.

Hinsichtlich der Sicherheit ist die E-Mail-Schnittstelle (siehe QA-9) kritisch, weil die E-Mails vertrauliche Daten enthalten können. Indirekt könnten diese vertraulichen Daten, falls sie bekannt würden, zur Änderung und Löschung von Daten, also zur Verletzung der Integrität, führen. Daher ist eine gesicherte Verbindung sinnvoll.
Technisch wird für das Versenden von E-Mails eine SMTP-Schnittstelle eingesetzt. Sie ist für eine gesicherte Verbindung konfigurierbar. Überwiegend sollte der E-Mail-Versand asynchron zu der Bearbeitung von Anfragen der Akteure erfolgen, solange der Versand aus fachlicher Sicht nicht kritisch für den entsprechenden Vorgang ist.

Das externe Terminverwaltungssystem kann bestehende Termine mit denen des Terminvermittlungssystems über eine funktionale Schnittstelle abgleichen. Dadurch müssen interne Mitarbeiter bestehende oder neu erstellte Termine nicht mehr manuell übertragen. Eine Voraussetzung für die Anbindung ist, dass das Terminverwaltungssystem überhaupt von der Einzelperson eingesetzt wird und sie den Zugang konfiguriert.
Aus fachlicher Sicht ist die Schnittstelle nicht kritisch, weil die Anbindung an ein externes Terminverwaltungssystem für die Terminanbieter optional ist. Bei fehlender Synchronisation kann der Fall auftreten, dass Termine in Zeiträumen vergeben werden, in denen schon Termine bestehen. Möglich ist auch, dass ein Terminanbieter neu gebuchte Termine nicht im externen Terminverwaltungssystem abrufen kann. Vorübergehende Ausfälle sollten also vom System toleriert werden können.
Da Termine einer Person vertraulich sind, sollte die Verbindung gesichert sein.
Technisch wird für die Schnittstelle der Synchronisation der Termine das CalDAV-Protokoll genutzt. Die Synchronisation der Termine sollte asynchron zur Bearbeitung von Anfragen der Akteure geschehen, um Termine im Hintergrund abzugleichen. Das kann einerseits dafür sorgen, dass vom Fremdsystem veränderte Termine nicht unmittelbar berücksichtigt werden können, andererseits wird die Performance der Anfrageverarbeitung nicht direkt von der Synchronisation beeinflusst.

Das externe Datenbankmanagementsystem besteht nur im technischen Kontext (Abbildung 4.3), weil damit die Daten persistiert werden und keine konkrete fachliche Funktionalität geboten wird. Die Schnittstelle ist eine Datenschnittstelle. Da die Nutzung der Datenbank für die Bearbeitung von Anfragen elementar ist, ist keine Fehlertoleranz zugelassen. Da in der Datenbank vertrauliche Daten gesichert werden, deren Vertraulichkeit und Integrität zu schützen ist, sollte der Zugriff auf die Schnittstelle gesichert sein.
Das Terminvermittlungssystem greift über JPA auf das Datenbankmanagementsystem zu. Je nach Konfiguration kann das Datenbankmanagementsystem auch im System integriert sein.

4.7 Bausteinsicht

Level 1

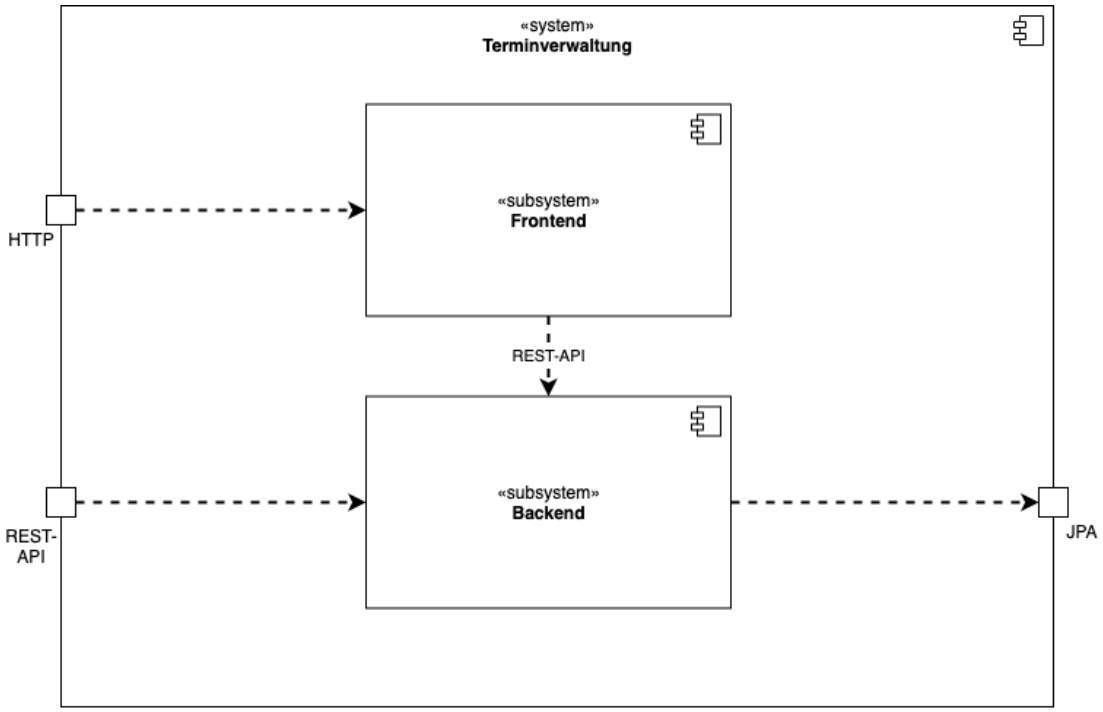

Abbildung 4.4: Komponentendiagramm des Gesamtsystems

Das Gesamtsystem für die Terminvermittlung ist in zwei Subsysteme (gemäß Abschnitt 4.3) unterteilt, das Frontend und Backend (siehe Abbildung 4.4).
Das Frontend-Subsystem ist für die Benutzerinteraktion zuständig. Es umfasst die Präsentationsschicht. Über eine HTTP-Schnittstelle kann auf die Benutzeroberfläche zugegriffen werden, sie ist im fachlichen und technischem Kontext repräsentiert.

Das Backend-Subsystem stellt dem Frontend Funktionalitäten bereit, um die fachlichen Aufgaben umzusetzen. Es umfasst die Applikationsschicht, die Domänenschicht und teilweise die Infrastrukturschicht. Das Backend greift auf eine externe Datenbank der Infrastrukturschicht über eine JPA-Schnittstelle zu. Je nach Konfiguration ist es auch möglich, dass die Datenbank ins Backend integriert ist.
Das Backend bietet eine REST-API an, die gemäß QA-4 das Level 2 nach dem Richardson Maturity Model erfüllen soll. Der Zugriff auf die REST-API vom Frontend aus wird

trotz einer unterschiedlichen Verteilung des Frontends und Backends (siehe Verteilungssicht) als systemintern angesehen. Generell ist ein externer Zugriff auf die REST-API des Backends möglich, somit könnte ein Endnutzer potenziell externe, alternative Frontend-Lösungen für das System einsetzen.

Level 2 (Backend)

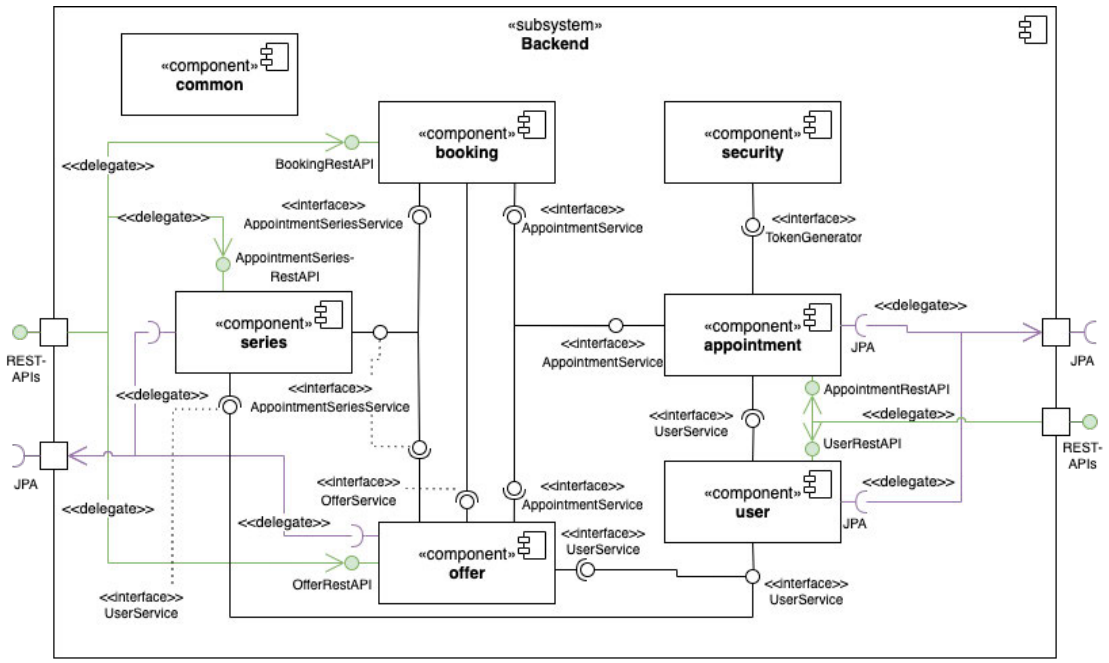

Abbildung 4.5: Komponentendiagramm des Backend-Subsystems

Aus Gründen der Übersichtlichkeit werden die externen Schnittstellen auf der rechten und linken Seite dupliziert dargestellt, obwohl sie zur gleichen Schnittstelle gehören.

Das in Abbildung 4.5 dargestellte Backend-Subsystem ist gemäß der horizontralen Architekturstuktur (siehe Abschnitt 4.1.2) modular in jeweils fachlich kohäsive Komponenten unterteilt.
Die Komponenten interagieren zwecks der Wartbarkeit mithilfe von Schnittstellen. Das senkt die Kopplung und stärkt die Kapselung. Mit dem Schnittstelleneinsatz sind Vorteile wie eine erhöhte Austauschbarkeit bzw. Änderbarkeit verbunden. Die fachlichen Zusammenhänge spiegeln sich in der Interaktion der Komponenten über ihre Schnittstellen wider.

Das Backend bietet als externe Schnittstelle eine REST-API an, die als Komposition aller Einzel-REST-APIs zu verstehen ist, deren jeweilige Adressierung mithilfe des Uniform Resource Identifiers einer Anfrage geschieht. Die REST-APIs werden in RestController-Klassen der Applikationsschicht mithilfe von passenden Annotationen definiert.
Zudem greift das Backend auf die externe JPA-Schnittstelle zur Persistenz von Daten innerhalb der Infrastrukturschicht zu, zum Beispiel für Termindaten oder Nutzerdaten. Der Zugriff erfolgt innerhalb der Komponenten durch eine interne Nutzung von Repository-Interfaces nach dem Repository Pattern. Die Implementierung wird vom Spring-Framework automatisch bereitgestellt (vgl. [67]).

Die „common“-Komponente beinhaltet gemeinsam genutzte Inhalte, die von mehreren Komponenten eingesetzt werden, zum Beispiel Utility-Klassen oder Datentypen. Aus Gründen der Übersichtlichkeit sind diese Abhängigkeiten nicht in der Abbildung 4.5 eingetragen.

Nutzer werden von der „user“-Komponente verwaltet. Sie bietet die „UserRestAPI“ an, mit der öffentliche Nutzerinformationen abgerufen werden können. Zudem bietet die Komponente die „UserService“-Schnittstelle an. Damit können Nutzer abgerufen werden, und es kann abgefragt werden, ob ein Nutzer und welcher Nutzer angemeldet ist. Die Funktionalität der Schnittstelle wird zurzeit der Domänenschicht zugeordnet. Die Teilfunktionalität, ob ein Nutzer und welcher Nutzer, der eine REST-Anfrage stellt, angemeldet ist, ist eher der Applikationsschicht zuzuordnen, weil die Domänenschicht von dem Kontext der REST-Anfrage entkoppelt sein sollte. Das ist zukünftig zu verbessern und auszulagern, dann müssten auch den Schnittstellen der Domänenschicht, die eine Anmeldung voraussetzen, Informationen bereitgestellt werden, welcher Nutzer angemeldet ist, zum Beispiel als Parameter.

Für die Sicherheit ist die „security“-Komponente zuständig. Diese Komponente umfasst in der Applikationsschicht die Konfiguration der Sicherheit für Interaktionen mit HTTP-Anfragen. In der Domänenschicht ist ein Generator für zufällige Token vorgesehen, der die „TokenGenerator“-Schnittstelle für die Komponente anbietet.

Die „appointment“-Komponente verwaltet Termine und ermöglicht die Erstellung von Terminen mit der Schnittstelle „AppointmentRestAPI“. Zudem bietet die Komponente die „AppointmentService“-Schnittstelle an, mit der Termine erstellt und für gegebene Zeiträume abgefragt werden können. Die Erstellung eines Termins berücksichtigt auch Gäste. Damit diese Zugang zu einem Termin ohne Nutzerregistrierung erhal-

ten, erstellt die „appointment“-Komponente jeweils Zugriffstokens durch Nutzung der „TokenGenerator“-Schnittstelle der „security“-Komponente.

Für die Verwaltung von Terminserien ist die „series“-Komponente zuständig. Sie stellt die „AppointmentSeriesRestAPI“ bereit, mit der Terminserien erstellt oder alle Terminserien des angemeldeten Nutzers abgerufen werden können. Die „AppointmentSeriesService“-Schnittstelle ermöglicht die Erstellung sowie das Abrufen von Terminserien für einen Nutzer und den Abruf einer Terminserie mittels einer Id.

Die „offer“-Komponente ist für Terminangebote zuständig. Dazu gehört neben der Ermittlung von Terminangeboten auch die Verwaltung von Terminangebotszeiträumen. Die bereitgestellte Schnittstelle „OfferRestAPI“ ermöglicht die Erstellung von Terminangebotszeiträumen und die Ermittlung von angebotenen Terminserien, Daten und Zeiträumen für mögliche Termine. Mit der ebenfalls angebotenen Schnittstelle „OfferService“ können Daten und Zeiträume für Terminangebote ermittelt werden. Um bestehende Termine für Terminangebote zu berücksichtigen, greift die offer“-Komponente auf die „AppointmentService“-Schnittstelle der „appointment“-Komponente zu.

Angebotene Termine können mithilfe der „booking“-Komponente gebucht werden. Mit der angebotenen „BookingRestAPI“ kann eine Buchung eines Terminangebots vollzogen werden. Dafür überprüft die Komponente durch einen Aufruf der „OfferService“-Schnittstelle, ob das zu buchende Terminangebot noch besteht. Zudem wird die zugehörige Terminserie mit der „AppointmentSeriesService“-Schnittstelle abgerufen, um sie als Vorlage für den zu buchenden Termin zu nutzen. Der Termin wird durch den Aufruf der „AppointmentService“-Schnittstelle erstellt.

Level 2 (Frontend)

Das in Abbildung 4.6 dargestellte Frontend weist als Subsystem analog zum Backend eine modulare Aufteilung in Komponenten auf. Es greift für fachliche Funktionalität auf die vom Backend angebotene externe REST-API zu.
Eine Angular-Component ist durch das Stereotyp „«ng-component»“ repräsentiert und ein Angular-Service durch das Stereotyp „«ng-service»“. Beide Stereotypen treten hier jedoch als konkrete Klasse, nicht als gesondert definierte Schnittstelle auf.
Alle Angular-Components mit der Endung „PageComponent“ repräsentieren nach der Referenzarchitektur jeweils eine Seite innerhalb der Präsentationsschicht. Angular-Services, die mit „DataService“ enden, dienen dem Zugriff auf die REST-API des Backends und

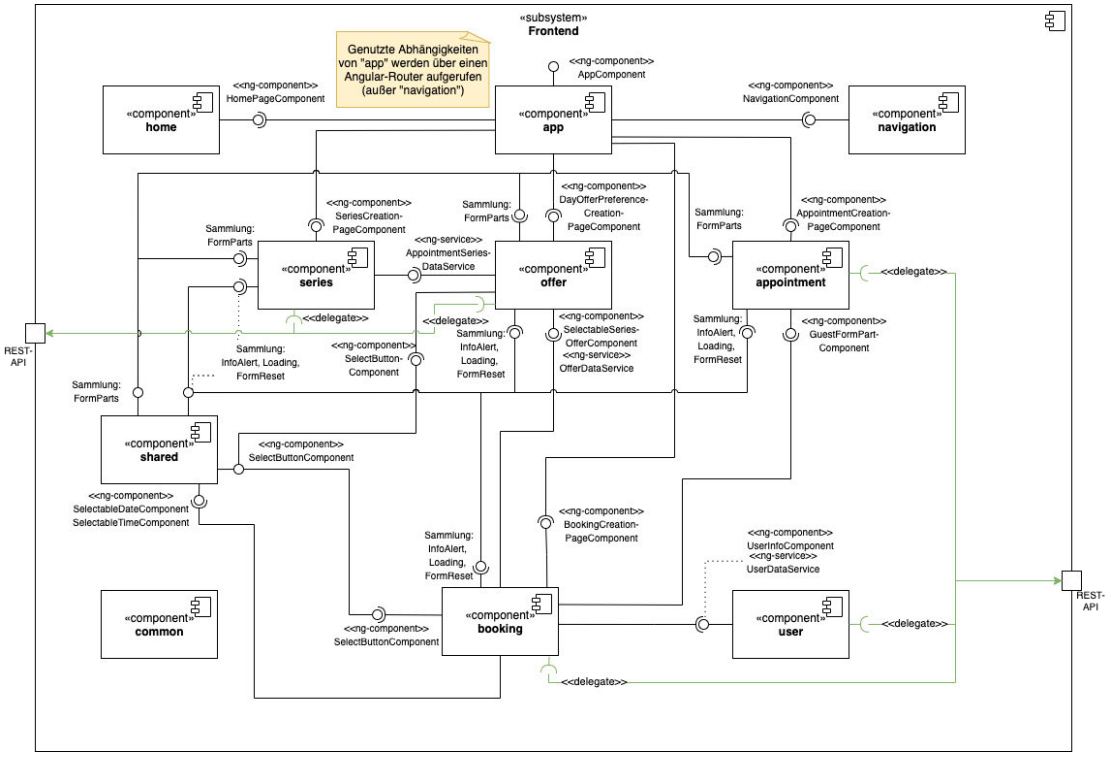

Abbildung 4.6: Komponentendiagramm des Frontend-Subsystems

sind gemäß der Referenzarchitektur der Kernschicht zuzuordnen.
Da Angular-Components üblicherweise von anderen Angular-Components ohne Interfaces eingebunden werden, wurden für sie keine zusätzlichen Schnittstellen definiert. Die Nutzung von Interfaces für Angular-Services erfolgt bisher nicht, weil für jede Seite (Page), falls erforderlich, genau ein konkreter Angular-Service angebunden wird, der nicht für andere Seiten wiederverwendet wird. Aus Gründen der Einfachheit bestehen für diese Art von Services keine Interfaces. Deren Vermeidung wurde aus Gründen der Konsistenz auf alle Angular-Services ausgeweitet. Der Einsatz von Interfaces für Angular-Services ist eine zukünftige Verbesserungsmöglichkeit.
Beschriftungen in der Abbildung 4.6, die mit „Sammlung:“ beginnen, repräsentieren mehrere Angular-Components und/oder Angular-Services.

Die zentrale „app“-Komponente ist für die Orchestrierung der Gesamtanwendung zuständig. Deren „AppComponent“ repräsentiert die gesamte Benutzeroberfläche des Frontends. Sie bindet für die Navigationsleiste die „NavigationComponent“ von der Komponente „navigation“ ein.
Alle anderen dargestellten Abhängigkeiten der „app“-Komponente bestehen indirekt, weil

sie zusätzlich einen Angular-Router für die Anzeige der Inhalte einsetzt und konfiguriert. Der Angular-Router stellt eine Seite mithilfe einer eingebundenen Angular-Components abhängig von der URL dar. Alle für diesen Zweck genutzten Angular-Components enden mit der Bezeichnung „PageComponent“.

Die „home“-Komponente ist für die Startseite zuständig und bietet sie als „HomePageComponent“ an.

Für die Verwaltung der Termine ist die „appointment“-Komponente zuständig. Sie bietet „AppointmentCreationPageComponent“ zur Erstellung eines Termins als Schnittstelle an. Zudem stellt sie ein Formular für die Erstellung eines Gasts („AppointmentCreationPageComponent“) im Rahmen der Wiederverwendung bereit.

Die „series“-Komponente ist für die Verwaltung von Terminserien zuständig. Um Terminserien zu erstellen, bietet sie die „SeriesCreationPageComponent“. Der „AppointmentSeriesDataService“ wird angeboten, damit andere Angular-Services Daten bezüglich Terminserien abrufen können.

Terminangebote und -präferenzen fallen in den Zuständigkeitsbereich der „offer“-Komponente. Sie ermöglicht die Erstellung von Terminangebotszeiträumen mithilfe der „DayOfferPreferenceCreationPageComponent“. Um bestehende Terminserien zu ermitteln, greift sie auf den „AppointmentSeriesDataService“ zu. Für das Abrufen von Daten bezüglich Terminangeboten bietet die „offer“-Komponente den „OfferDataService“ an. Für die Auswahl einer angebotenen Terminserie stellt sie die „SelectableSeriesOfferComponent“ bereit.

Die „user“-Komponente ist für Nutzer zuständig. Sie stellt die „UserInfoComponent“ zur Darstellung von Nutzerinformationen und den „UserDataService“ für Nutzerdaten bereit.

Mit der „booking“-Komponente wird die Buchung von Terminangeboten ermöglicht. Für die Auswahl und Buchung von Terminangeboten stellt die Komponente die „BookingCreationPageComponent“ bereit. Das Abrufen von Terminangeboten und die Anzeige angebotener Terminserien wird durch einen Zugriff auf den „OfferDataService“ bzw. auf die „SelectableSeriesOfferComponent“ der „offer“-Komponente ermöglicht. Um für Endnutzer Informationen über Terminanbieter zu ermitteln und anzuzeigen, setzt sie den „UserDataService“ sowie die „UserInfoComponent“ der „user“-Komponente ein. Für Personenangaben wird das Gastformular „GuestFormPartComponent“ aus der „appointment“-Komponente wiederverwendet.

Die „common“-Komponente bietet analog zur gleichnamigen Komponente im Backend allgemeine Inhalte, insbesondere Utility-Klassen oder Konstanten. Aus Gründen der Übersichtlichkeit wurden ebenfalls keine Abhängigkeiten in der Abbildung 4.6 eingetragen.

Im Vergleich zur „common“-Komponente bietet die „shared“-Komponente Inhalte, die weniger abstrahiert sind und sich stärker auf das konkrete System bzw. auf dessen Anwendungsbereich beziehen. Die Inhalte werden von mehreren Komponenten benötigt, sind daher nicht einer einzelnen zuzuordnen.
So bietet die „shared“-Komponente für andere Komponenten Formularteile, zum Beispiel für die Angabe einer 5-Minuten-granulären Dauer. Hinzu kommen Angular-Components und/oder Angular-Services für Informationsmeldungen, das Signalisieren des Ladens und das Zurücksetzen von Formularen. Ebenfalls werden Angular-Components für eine auswählbare Zeit, ein auswählbares Datum oder für eine Auswahlschaltfläche angeboten.

Level 3 (Backend)

Die für Terminangebote und Terminangebotszeiträume zuständige „offer“-Komponente des Backends, in Abbildung 4.7 dargestellt, umfasst eine Klasse für die Bereitstellung der REST-API für Terminangebote („OfferRestController“) und eine Klasse, welche die Schnittstelle für Terminangebote („OfferService“) anbietet („ConcreteOfferService“). Zudem bestehen Subkomponenten, die für Terminangebotszeiträume („preference“-Komponente) und für die Ermittlung und Bewertung von Terminoptionen („option“- bzw. „rating“-Komponente) zuständig sind.

Die angebotene „OfferRestAPI“ repräsentiert die Gesamtheit der REST-Schnittstellen für Terminangebote und Terminangebotszeiträume (siehe auch Abbildung 4.5). Zudem bietet die „offer“-Komponente die Schnittstelle „OfferService“ an und greift auf die Schnittstellen „AppointmentService“, „AppointmentSeriesService“ sowie JPA (für die Datenspeicherung) zu.

Die Subkomponente „preference“ bietet die „DayOfferPreferenceRestAPI“ für Terminangebotszeiträume, an die passende REST-Anfragen delegiert werden. Die angebotene Schnittstelle „DayOfferPreferenceService“ ermöglicht die Erstellung von Terminangebotszeiträumen, die Abfrage von Terminserien eines Nutzers, die durch Terminangebotszeiträume angeboten werden, sowie die Abfrage von Terminangebotszeiträumen nach Terminserie oder Datum. Um einen angemeldeten oder speziellen Nutzer abzufragen, greift sie auf die „UserService“-Schnittstelle zu. Für den Umwandlungsvorgang eines

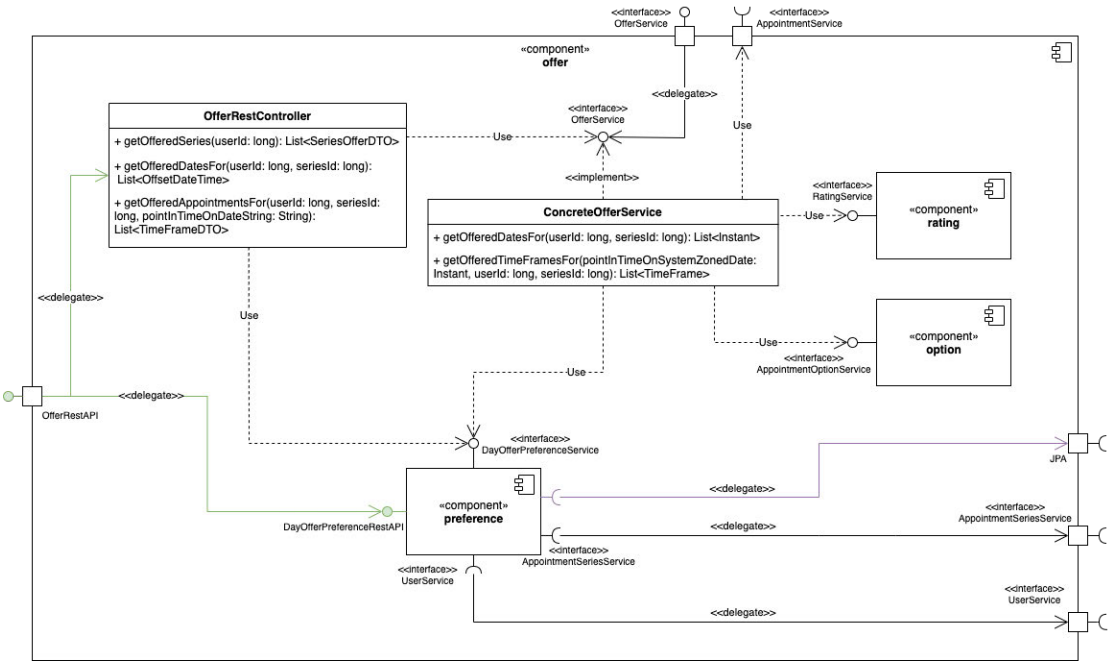

Abbildung 4.7: Komponentendiagramm der offer-Komponente des Backends
Dieses Diagramm berücksichtigt auch Klassen der „offer“-Komponente. Aus Gründen der Übersichtlichkeit sind darin keine privaten Felder, keine Konstruktoren und keine Informationen über mögliche Exceptions von Methoden enthalten.

DTOs der REST-API ist der Abruf einer Terminserie durch die Schnittstelle „AppointmentSeriesService“ notwendig (siehe Abschnitt 5.2). Für die Datenverwaltung greift die Komponente auf die JPA-Schnittstelle zu.

Die „option“-Komponente bietet die Schnittstelle „AppointmentOptionService“. Damit lassen sich alle Optionen ermitteln, an denen ein Termin für eine Terminserie angeboten werden kann. Dafür sind eine Terminserie, angebotene Zeiträume, bestehende Termine, ein Zeitpuffer und Pausenpräferenzen zu berücksichtigen, die als Parameter gegeben sein müssen. Die ermittelten Optionen umfassen auch alle möglichen Pausenvariationen. Es kann alternativ überprüft werden, ob überhaupt eine Option für angebotene Termine besteht.
Innerhalb der „option“-Komponente wird das Strategy Pattern eingesetzt, um bei Bedarf den Austausch eines Algorithmus für die Ermittlung von Optionen zu vereinfachen.

Die „rating“-Komponente bietet die Schnittstelle „RatingService“. Damit lassen sich die am höchsten bewerteten Terminoptionen ermitteln. Dafür müssen neben den Terminoptionen weitere Informationen vorhanden sein, weil für die Bewertungen unterschiedliche

Kriterien, z.B. möglichst wenige Lücken zwischen Terminen, herangezogen werden.
Für austauschbare Bewertungsheuristiken und die Zuordnung von bestehenden Terminserienpräferenzen zu den Bewertungsheuristiken werden innerhalb dieser Komponente das Strategy Pattern und das Abstract Factory Pattern kombiniert genutzt. Jede Strategy repräsentiert ein Bewertungskriterium von einer Terminserienpräferenz. Abweichend zu dem Strategy Pattern wird nicht eine, sondern mehrere Strategies verwendet. Mithilfe des Abstract Factory Patterns erzeugt eine Factory aus einer gegebenen Terminserienpräferenz alle Bewertungskriterien als Strategies. Bei Bedarf können andere Factories eingesetzt werden, die andere Strategies, zum Beispiel mit verbesserten Algorithmen, erzeugen.

Die Schnittstelle „OfferService“ der „offer“-Komponente ermöglicht es, angebotene Daten für eine Terminserie eines Nutzers zu ermitteln, ebenso angebotene Terminzeiträume für ein konkretes Datum, die dem Anbieter am Besten passen. Die Schnittstelle wird von der Klasse „ConcreteOfferService“ (in der Domänenschicht) angeboten.
Für die Umsetzung der Schnittstelle ruft die Klasse bestehende Termine mit der „AppointmentService“-Schnittstelle und Terminangebotszeiträume mit der „DayOfferPreferenceService“-Schnittstelle ab. Um Terminoptionen (für das Anbieten von Daten oder Zeiträumen) zu ermitteln, wird die „AppointmentOptionService“-Schnittstelle eingesetzt. Zusätzlich werden für die Ermittlung der am Besten passenden Terminzeiträume die Terminoptionen mithilfe der „RatingService“-Schnittstelle bewertet.

An die Klasse „OfferRestController“ der Applikationsschicht werden REST-Anfragen für angebotene Terminserien, Daten und (dem Anbieter am Besten passende) Zeiträume für Termine delegiert.
Für die Bearbeitung der Anfrage angebotener Terminserien greift „OfferRestController“ auf die „DayOfferPreferenceService“-Schnittstelle zurück. Um angebotene Daten und Terminzeiträume zu ermitteln, nutzt diese Klasse die Schnittstelle „OfferService“.
Rückgaben werden für REST-Anfragen vom Spring-Framework automatisch in JSON-Repräsentationen umgewandelt.

Level 3 (Frontend)

Die „offer“-Komponente des Frontends für Terminangebote und Terminangebotszeiträume, dargestellt in der Abbildung 4.8, umfasst eine Subkomponente für Terminangebotszeiträume („day-offer-preference“) und eine Angular-Component für die Darstellung einer

auswählbaren, angebotenen Terminserie („SelectableSeriesOfferComponent“). Zudem ist in der „offer“-Komponente ein Angular-Service („OfferDataService“) für das Abrufen von angebotenen Terminserien, Termindaten und -zeiträumen enthalten.

Die „day-offer-preference“-Komponente ermöglicht die Erstellung von Terminangebotszeiträumen mit der „DayOfferPreferenceCreationPageComponent“. Der „AppointmentSeriesDataService“ wird von dieser Komponente benötigt, um Terminserien für die Erstellung von Terminangebotszeiträumen zu ermitteln. Die „day-offer-preference“-Komponente nutzt weitere Angular-Components und -Services für Erfolgs- bzw. Fehlermeldungen, den Ladezustand, das Rücksetzen eines Formulars sowie für Formularteile.
Für die Durchführung der Erstellung von Terminangebotszeiträumen greift diese Komponente auf die REST-API des Backends zu.

Die Klasse „SelectableSeriesOfferComponent“ stellt als Angular-Component eine angebotene Terminserie dar, kann ausgewählt sein oder lässt sich auswählen. Bei Auswahl ruft sie eine Funktion auf, die ihr bei der Erzeugung mitgegeben wurde. Die Klasse wird von der „offer“-Komponente angeboten. Um eine Auswahlschaltfläche einzubinden, greift sie auf die „SelectButtonComponent“ zu.

Die Klasse „OfferDataService“ ermöglicht als Angular-Service das Abrufen angebotener Terminserien, Terminzeiträume und -daten. Mithilfe von Dependency Injection wird eine Instanz von „HttpClient“ für REST-Anfragen bereitgestellt. Pro Methode sind von „OfferDataService“ als Parameter Callback-Funktionen für den Erfolgs- und Fehlerfall erforderlich, weil die resultierende REST-Anfrage asynchron gestellt wird.
Je nach Methode sind fachliche Informationen erforderlich. Zum Beispiel muss für angebotene Terminzeiträume ein gewünschtes Datum, durch einen Zeitpunkt (als „Date“-Typ) repräsentiert, mitgegeben werden.

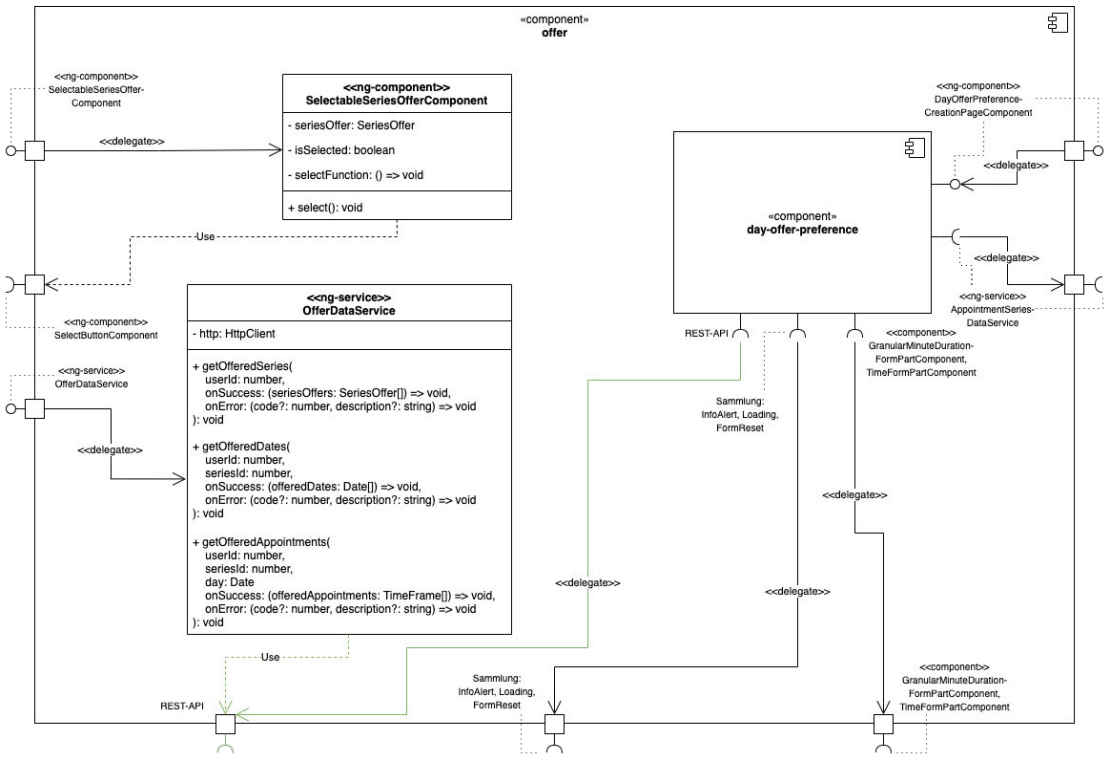

Abbildung 4.8: Komponentendiagramm der offer-Komponente des Frontends

Hinweise zur Abbildung:
Private Klassen-Felder können hinsichtlich ihrer Benennung von der konkreten Implementierung durch vorangestellte Unterstriche abweichen. Konstruktoren, Initialisierungsmethoden, Getter und statische Konstanten sind hier nicht dargestellt. Ebenso wurden URL-Teile, Methoden für den URL-Aufbau sowie Abhängigkeiten von genutzten Utility-Klassen und Bibliotheken nicht dargestellt.
Zu einer Klasse des Stereotyps „«ng-component»“ (Angular-Component) gehören zusätzlich (hier nicht dargestellte) Benutzeroberflächendefinitionen (durch HTML und CSS).

4.8 Laufzeitsicht

Im Rahmen der Laufzeitsicht werden für die Terminbuchung (siehe US-015 und UC-004) Teilvorgänge anhand von Sequenzdiagrammen für das Frontend und Backend dargestellt und beschrieben.
Die Ausgangslage besteht darin, dass ein(e) Student(in) oder externe Person eine angebotene Terminserie, ein Datum und einen Zeitraum für den zu buchenden Termin ausgewählt hat. Ebenso sollte dieser Nutzer ein Buchungsformular bereits ausgefüllt haben. Die Sequenzdiagramme umfassen die Schritte 4 bis 8 des Erfolgsszenarios von UC-004, in denen der Endnutzer die Buchung anfordert, die vom System überprüft und gebucht wird. Letztlich wird dem Benutzer eine Bestätigung angezeigt. Die Berücksichtigung des E-Mail-Versands und von Zeiträumen, an denen die anfragende Person nicht kann, besteht nicht, weil entsprechende funktionaler Anforderungen bei der iterativen Entwicklung nicht umgesetzt wurden.

Für die Sequenzdiagramme ist zu beachten, dass Logging aus Gründen der Übersichtlichkeit nicht dargestellt wird. Einige Vorgänge werden verkürzt dargestellt, als Aufruf innerhalb des eigenen Objekts (als „zusammengefasst“ oder „Zusammenfassung“ gekennzeichnet). Darüber hinaus kann es in der Darstellung vorkommen, dass Aufrufe, die innerhalb einer weiteren privaten Methode erfolgen, direkt dargestellt werden, ohne dass im Diagramm zuvor die zugehörige private Methode aufgerufen wird.

Frontend

Hinweise für Sequenzdiagramme: Die Nutzung von Gettern wird als Methodenaufruf dargestellt, obwohl diese in der Umsetzung ohne Klammern aufgerufen werden (wie „object.field()“ statt „object.field“). Der „?“-Operator ruft eine Methode eines Objekts nur auf, wenn es weder null noch undefined ist, ansonsten wird undefined zurückgegeben. Ausgehende HTTP-Anfragen werden vereinfacht dargestellt. Dafür werden Hilfsklassen des Angular-Frameworks eingesetzt.

Einige Angular-Services, die in folgenden Sequenzdiagrammen dargestellt werden (Benennung endet mit „Service“), entsprechen zudem nicht der Service-Definition von Evans, falls sie einen internen Zustand haben (vgl. [17, S. 105-106]). Das betrifft neben Seiten-Services (siehe Referenzarchitektur) auch Angular-Services, die für spezielle Funktionalitäten einer Seite, wie Erfolgs- bzw. Fehlermeldungen oder die Signalisierung des Ladezu-

stands, genutzt werden, weil in Bezug auf diese Funktionalitäten für jede Seite eine eigene Angular-Service-Instanz mit einem eigenem Zustand eingesetzt wird. Angular-Services, die mit „DataService“ enden, sind zustandslos und nicht betroffen, sie greifen auf die REST-API des Backends zu.

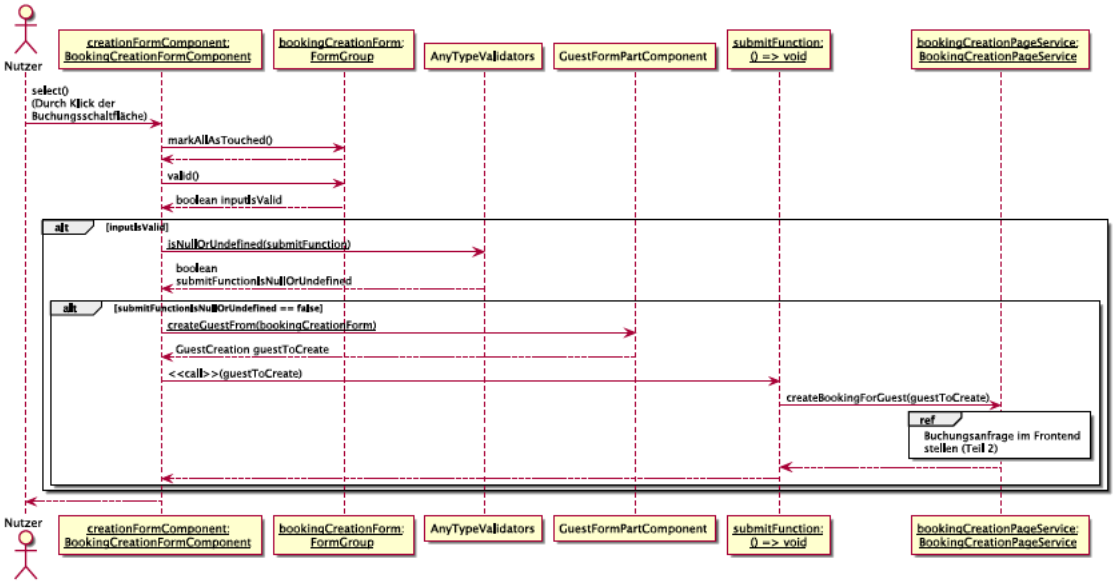

Abbildung 4.9: Sequenzdiagramm: Buchungsanfrage im Frontend stellen (Teil 1): Gastinformationen aus dem Formular validieren und extrahieren

Die Durchführung einer Buchung beginnt, wie in Abbildung 4.9 dargestellt, mit der Anforderung der Buchung durch Klicken auf die Buchungsschaltfläche (Schritt 4 des Erfolgsszenarios von UC-004). Damit wird innerhalb der Präsentationsschicht die Methode „select()“ der Angular-Component „BookingCreationFormComponent“ aufgerufen.
„BookingCreationFormComponent“ ist für das Buchungsformular zuständig, in das der Nutzer Daten eingegeben hat. Sie verwaltet eine „FormGroup“-Instanz, an welche die Nutzereingaben gebunden sind und in der Validatoren für die Eingaben hinterlegt sind. Für die Darstellung des Formulars für Gäste verwendet „BookingCreationFormComponent“ die Angular-Component „GuestFormPartComponent“ wieder, die auch für die Terminerstellung eingesetzt wird.
Nach dem „select()“-Aufruf wird für die Überprüfung der Eingaben (Schritt 5 des Erfolgsszenarios von UC-004) signalisiert, dass alle Felder als angefasst zu behandeln sind, um eine Eingabeüberprüfung für jedes Formularfeld auszulösen. Bei falschen oder fehlenden Eingaben erscheint automatisch ein Hinweis bei dem entsprechenden Feld (für den Fehlerfall 5-a).

Außerdem wird explizit die Validität der Eingaben überprüft. Falls diese ungültig sind, endet der weitere Programmfluss, um die Buchung zu verhindern. Es sind keine weiteren Maßnahmen mit Hinweisen an den Nutzer erforderlich, weil entsprechende Meldungen schon bei den Eingabefeldern angezeigt werden.
Für den Fall einer Buchung wurde der „BookingCreationFormComponent“ im Voraus eine Funktion „submitFunction“ bereitgestellt, die von der übergeordneten Angular-Component „BookingCreationPageComponent“ erstellt wurde. Bei gültigen Nutzereingaben und einer gegebenen Funktion werden die Nutzereingaben in eine andere Repräsentation umgewandelt und der Funktion beim Aufruf mitgegeben. Die Funktion delegiert die Buchungsanfrage an den „BookingCreationPageService“ (siehe Abbildung 4.9 und 4.10).

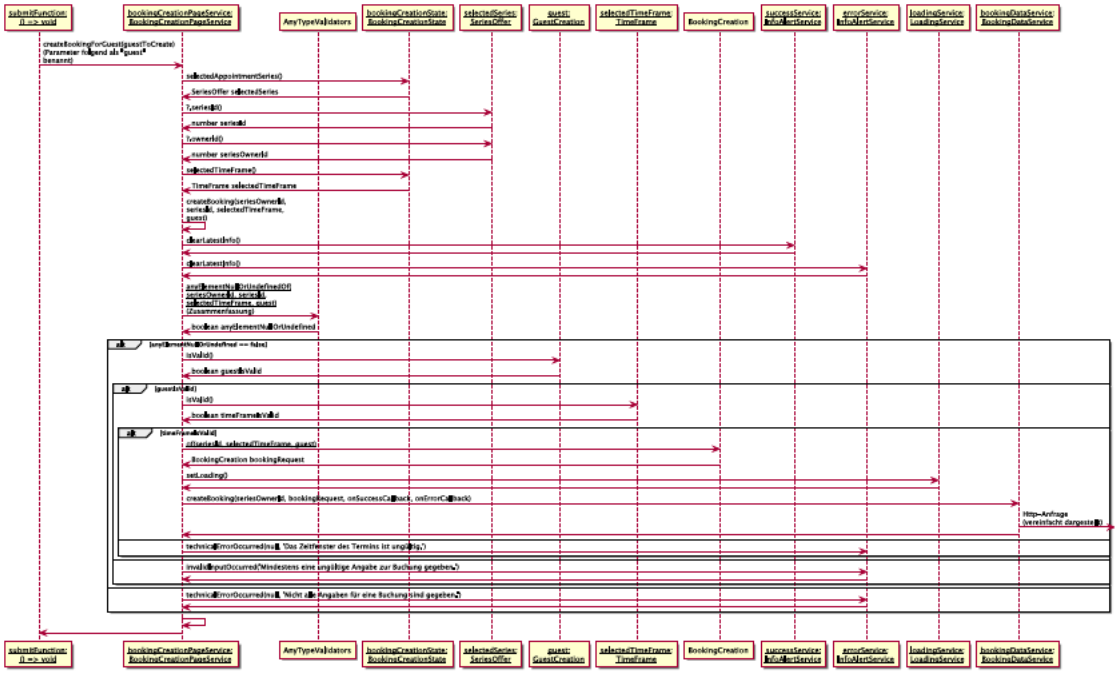

Abbildung 4.10: Sequenzdiagramm: Buchungsanfrage im Frontend stellen (Teil 2): Anfrage mit dem BookingCreationPageService stellen

Die Funktion „submitFunction“ ruft den „BookingCreationPageService“ für die Buchung eines Termins mit den Gastinformationen auf (siehe Abbildung 4.10). Diese Klasse ist gemäß der Referenzarchitektur für die Funktionalität der Buchungsseite in der Abstraktionsschicht zuständig.
Nach einer Ermittlung der ausgewählten Terminserie, des Anbieters (als Ids) sowie des Terminzeitraums wird sichergestellt, dass keine Erfolgs- und Fehlermeldungen mehr für

die Seite vorhanden sind.
Falls die Gastinformationen oder weitere ermittelte Informationen nicht vorhanden sein sollten, wird eine technische Fehlermeldung signalisiert und der Programmfluss der „submitFunction“ endet. Analog wird bei ungültigen Gastinformationen oder einem ungültigen angebotenen Zeitraum verfahren. Bei ungültigen Gastinformationen wird keine technische Fehlermeldung, sondern eine Fehlermeldung zur Korrektur von Nutzereingaben signalisiert (für den Fehlerfall 5-a von UC-004), weil die Gastinformationen auf Nutzereingaben basieren.
Anschließend werden für eine Buchung erforderliche Informationen zusammengestellt, und es wird signalisiert, dass ein Ladevorgang erfolgt.

Für die Delegierung des weiteren Buchungsvorgangs an das Backend wird der Service „BookingDataService“ in der Kernschicht (für die Schritte 6 und 7 von UC-004) aufgerufen. Diesem werden Callbacks für den Erfolgs- und Fehlerfall bereitgestellt. Der „BookingDataService“ delegiert die Buchungsanfrage über eine asynchrone HTTP- bzw. REST-Anfrage ans Backend.
Danach erhält das Frontend eine Antwort (außerhalb der Abbildung 4.10), deren Bearbeitung erfolgt asynchron. Die Antwort wird im Erfolgsfall von einem DTO in einen Datentyp der Buchungsbestätigung umgewandelt. Es erfolgt entweder ein Aufruf des Callbacks für den Fehlerfall oder Erfolgsfall.

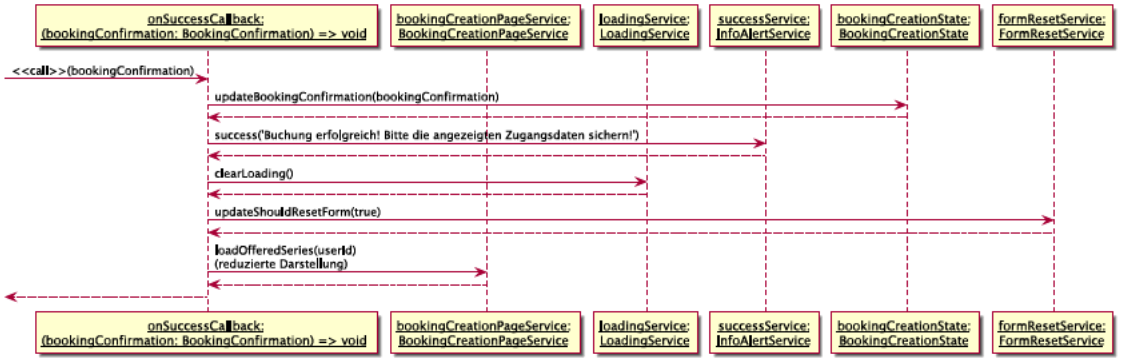

Abbildung 4.11: Sequenzdiagramm: Verarbeitung einer Buchungsbestätigung im Frontend

Es wird der Erfolgsfall ab dem Callback-Aufruf, wie in Abbildung 4.11 dargestellt, näher betrachtet. Der Vorgang ist dem Schritt 8 des Erfolgsszenarios von UC-004 (Anzeige der Buchungsbestätigung und Zugriffsinformationen) zuzuordnen.
Die Callback-Funktion „onSuccessCallback“ erhält als Parameter die Buchungsbestäti-

gung und verarbeitet diese. Der Zustand für die Buchungserstellung wird in der Kernschicht bezüglich der Buchungsbestätigung aktualisiert. Ebenfalls signalisiert die Callback-Funktion eine Erfolgsmeldung, das Ende des Ladevorgangs und die erforderliche Rücksetzung des Formulars für die Buchung in der Kernschicht.
Um die Buchungsbestätigung anzuzeigen, werden die Zustandsänderungen der Buchungsbestätigung automatisch durch Observables (nach dem Observer Pattern) an abonnierende Angular-Components in der Präsentationsschicht weitergereicht. Die Darstellung der Buchungsbestätigung erfolgt dann mithilfe von Data Binding.
Zudem wird das Neuladen von angebotenen Terminserien (in der Abstraktionsschicht) angestoßen, um weitere Buchungen zu ermöglichen. Der Inhalt des Neuladevorgangs wurde in der Abbildung 4.11 aus Gründen der Übersichtlichkeit ausgelassen.

Backend

Hinweise für Sequenzdiagramme: Optionals unterschiedlicher Typisierungen werden vereinfacht als eine einzige Instanz dargestellt.
Aus Gründen der Übersichtlichkeit wird das Werfen von Fehlern (Exceptions) innerhalb von „break“ als Rückgabe ohne vorige Erstellung der Exceptions dargestellt. Ist die Bedingung eines „break“-Blocks erfüllt, wird dessen Inhalt ausgeführt. Der Kontrollfluss des gesamten Sequenzdiagramms endet dann mit dem Block.

Das Backend tritt für den Buchungsvorgang in Aktion, wenn es eine entsprechende REST-Anfrage vom Frontend erhält. Die Schritte 6 und 7 des Erfolgsszenarios von UC-004 können den dabei ausgeführten Vorgängen zugeordnet werden, indirekt auch Schritt 5, da die Angaben der REST-Anfrage überprüft werden.

Die REST-Anfrage zur Buchung wird von Spring an die Buchungsmethode von der „BookingRestController“-Klasse (Applikationsschicht) aus der „booking“-Komponente delegiert. Vor dem Methodenaufruf werden die Inhalte der REST-Anfrage automatisch in die Datentypen der Methodenargumente umgewandelt und mithilfe von definierten Validatoren überprüft. Generell wird die Rückgabe des Methodenaufrufs bzw. eine auftretende Exception automatisch in eine entsprechende HTTP-Antwort umgewandelt.

Somit wird die Methode „createBooking“ mit Informationen über die Id des Terminanbieters und über die angefragte Buchung (als DTO) aufgerufen, siehe Abbildung 4.12. Bei einem nicht gegebenen Parameter wird eine entsprechende Exception geworfen.

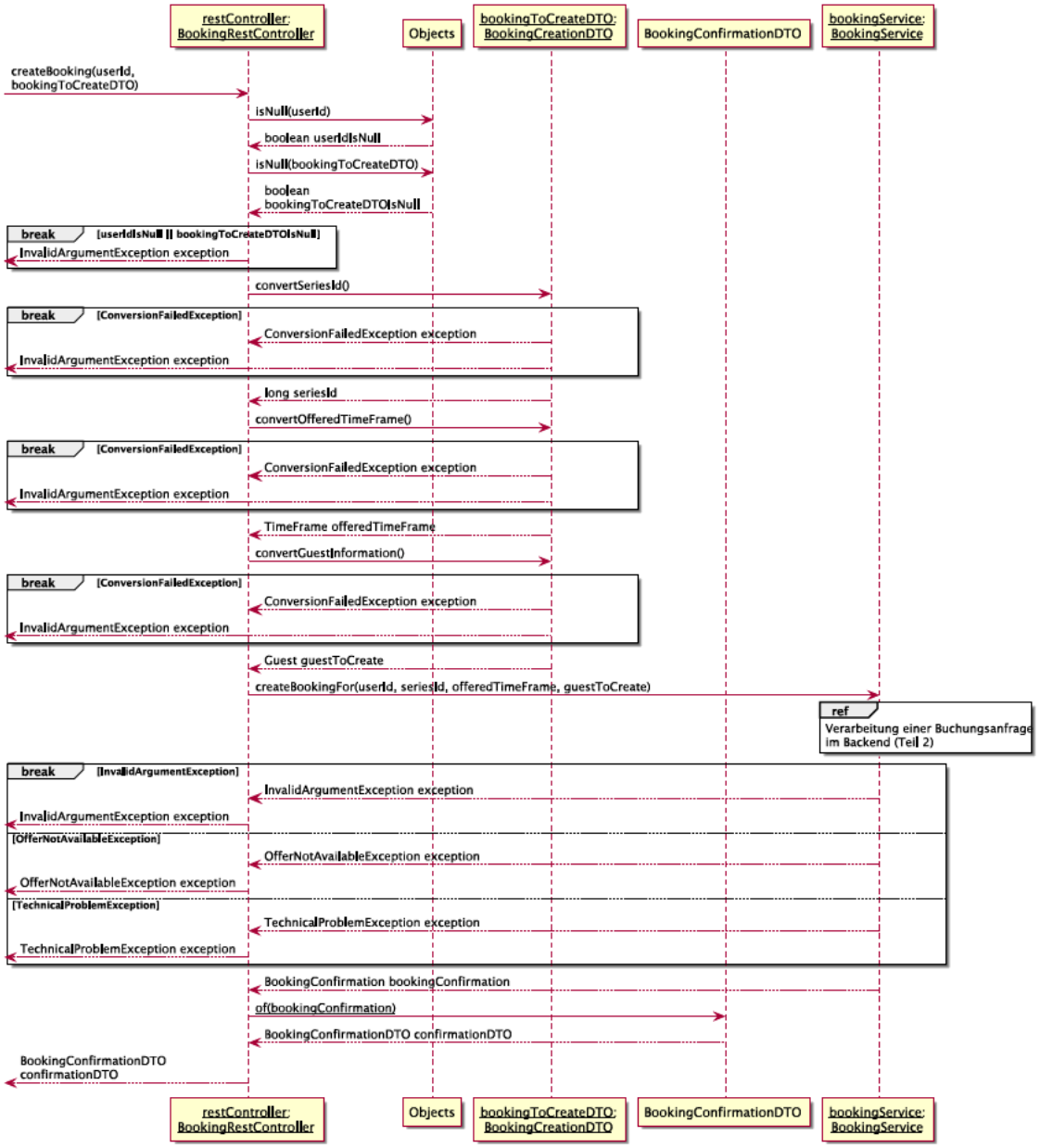

Abbildung 4.12: Sequenzdiagramm: Verarbeitung einer Buchungsanfrage im Backend (Teil 1): BookingRestController

Danach wird versucht, aus dem DTO Informationen wie die Id der Terminserie, der angebotene Zeitraum und die Gastinformationen zu extrahieren und in entsprechende Objekte umzuwandeln. Falls ein Konvertierungsfehler auftritt, wird eine Exception für ein ungültiges Argument geworfen.

Anschließend wird der Buchungsauftrag zur Bearbeitung an die Schnittstelle „BookingService“ der Domänenschicht delegiert (siehe Abbildung 4.12). Dabei auftretende Exceptions werden weitergereicht. Weitere Details zum Schnittstellenaufruf werden folgend im Kontext zur Abbildung 4.13 betrachtet.
Das Ergebnis des Methodenaufrufs ist die Bestätigung der Buchung. Sie wird in eine entsprechende DTO-Repräsentation umgewandelt. Deren Terminfelder, die für den Terminanbieter privat sind, sind aus Gründen der Vertraulichkeit (siehe QA-9) auf nicht vorhanden (null) gesetzt. Ein Beispiel ist die Vorbereitungszeit des Terminanbieters, über die der Terminnehmer nichts wissen sollte. Das DTO wird für die Beantwortung der HTTP-REST-Anfrage zurückgegeben.

Nun erfolgt die Weiterverarbeitung der Buchungsanfrage in der Domänenschicht durch Delegierung zur Methode „createBookingFor“ der „BookingService“-Schnittstelle, wie in Abbildung 4.13 dargestellt. Die konkrete Klasse bleibt dem Aufrufer verborgen, weil eine Schnittstelle und Dependency Injection von Spring genutzt wird. Diese Klasse ist hier „ConreteBookingService“.

Die Methode ist transaktional, dabei werden zu persistierende Änderungen (in der Implementerung durch Mechanismen von Spring) automatisch rückgängig gemacht, wenn Exceptions wie „InvalidArgumentException“, „OfferNotAvailableException“ oder „TechnicalProblemException“ auftreten.
Zu Beginn der Methodenausführung wird sichergestellt, dass alle Parameter valide sind. Falls nicht, wird eine Exception geworfen, die ungültige Parameter signalisiert (Fehlerfall 6-a von UC-004). Speziell für die Buchung ist wichtig, dass ein Termingrund angegeben ist. Dieser ist bei anderen Arten der Terminerstellung, wie der Erstellung durch einen Anbieter, nicht zwingend erforderlich. Weitere Prüfungen der Gültigkeit der Gastinformation erfolgen später im „AppointmentService“.

Nach der Ermittlung des Terminstarts wird die persistente Terminserie unter Angabe der Id abgerufen. Sie dient als Vorlage für einen zu buchenden Termin. Auftretende Exceptions für technische Probleme werden weitergeworfen. Falls die Terminserie doch nicht bestehen sollte, wird angenommen, dass das Terminangebot auch nicht mehr besteht. Das

wird durch eine Exception signalisiert und ist dem Schritt 6, ob die Terminbuchungsmöglichkeit noch besteht, sowie dem Fehlerfall 6-a von UC-004 zuzuordnen.
Aus Informationen der Terminserie sowie dem Startdatum wird ein neues Terminobjekt erstellt. Zudem werden teilnehmende Gäste zusammengestellt. Das Persistieren erfolgt noch nicht.
Nun startet ein kritischer Abschnitt („synchronized“-Aufruf), in dem die Überprüfung der Beständigkeit des zu buchenden Terminangebots und die tatsächliche persistente Erstellung des Termins zusammenhängend durchgeführt werden. Ansonsten wären Mehrfachbuchungen wahrscheinlicher, wenn mehrere Threads gleichzeitig bemerken, dass ein angefragter Termin noch angeboten wird und sie diesen buchen. Voraussetzung für den kritischen Abschnitt ist jedoch, dass nur eine einzige Instanz des „ConcreteBookingService“ besteht und dass das Backend nicht mehrfach ausgeführt wird. Zudem sollte eine Transaktionsisolation vorhanden sein, die einen Dirty Read verhindert (zum Beispiel die Isolationsebene Read Committed).
Im kritischen Abschnitt wird überprüft, ob der zu buchende Terminzeitraum nach wie vor angeboten wird. Dafür erfolgt ein Methodenaufruf der Schnittstelle „OfferService“ (in Abbildung 4.13 zusammengefasst). Diese Überprüfung gehört zum Schritt 6 des UC-004. Der Verfall eines Terminangebots (Fehlerfall 6-a von UC-004) und auftretende technische Probleme (Fehlerfall 7-b) werden mithilfe von entsprechenden Exceptions signalisiert.
Danach erfolgt im kritischen Abschnitt das Persistieren des Termins sowie der Gäste durch einen Aufruf von „createAppointment“ der „AppointmentService“-Schnittstelle (Schritt 7 von UC-004). Es kann dabei eine Exception für ungültige Argumente auftreten, die weitergeworfen wird (für Fehlerfall 5-a). Andere Exceptions sind nicht auf Nutzerfehler zurückzuführen und werden daher als technische Fehler weitergeworfen (Fehlerfall 7-b).
Es folgt das Ende des kritischen Abschnitts. Falls entgegen der Erwartung kein Gast für den gebuchten (persistierten) Termin besteht, wird ein technisches Problem durch eine Exception signalisiert (Fehlerfall 7-b).
Zuletzt wird eine Buchungsbestätigung zusammengestellt und zurückgegeben, die neben dem Termin und Gästen auch Zugangsdaten für den buchenden Gast enthält.

Da die gesamte Methode „createBookingFor“ transaktional ist, also erst ein Commit der zu persistierenden Daten am Ende der Methode nach (und nicht in) dem kritischen Abschnitt erfolgt, können Doppelbuchungen von Terminen nicht ausgeschlossen werden.

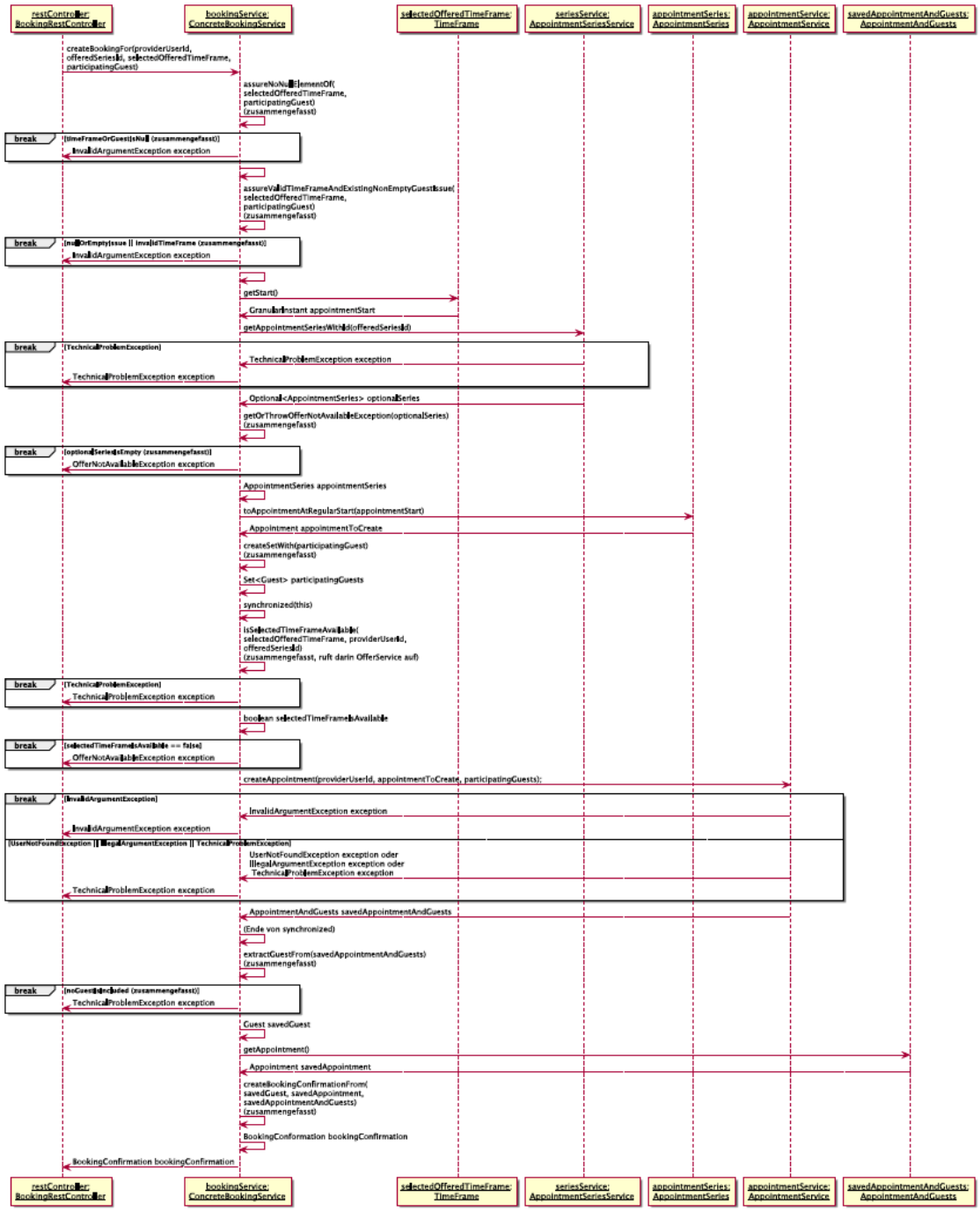

Abbildung 4.13: Sequenzdiagramm: Verarbeitung einer Buchungsanfrage im Backend (Teil 2): ConcreteBookingService

Zum Beispiel wurde in einem Thread der kritische Abschnitt durchlaufen und darin ein Termin persistiert, ohne dass bisher ein Commit geschehen ist, weil die Methode noch nicht zu Ende ausgeführt wurde. Währenddessen durchläuft ein anderer Thread den kritischen Abschnitt, merkt, dass derselbe gewünschte Termin noch angeboten wird, und sieht dabei die Buchung des ersten Threads nicht, weil dafür noch kein Commit geschehen ist. Nun führen beide Threads die Methode zu Ende aus und haben zwei Termine zu denselben Zeiten gebucht.
Um eine Buchung desselben Termins auszuschließen, müsste man sicherstellen, dass der Commit innerhalb des kritischen Abschnitts erfolgt. Trotzdem können Terminanbieter unabhängig von der Buchung und des dafür vorgesehenen kritischen Abschnitts Termine erstellen, die sich mit gebuchten Terminen überschneiden können.

4.9 Verteilungssicht

Während der Ausführung des Systems werden, wie in Abbildung 4.14 dargestellt, drei Komponenten benötigt: die ICC, das Endnutzergerät und der Google-Server.

Innerhalb von Servern der ICC wird das Backend ausgeführt und das Frontend ans Endnutzergerät ausgeliefert.
Um Bereitstellungen verwalten zu können, wird Kubernetes in der ICC eingesetzt. Innerhalb von Kubernetes-Pods können Docker-Container ausgeführt werden. Diese sind in der Abbildung 4.14 als eine Ausführungsumgebung zusammengefasst.
Das Backend-Artefakt wird durch eine ausführbare JAR-Datei („Backend-JAR“) repräsentiert. In dem Artefakt ist auch ein Webserver integriert, der während der Ausführung eine HTTP-REST-Schnittstelle bereitstellt. Das Artefakt wird mithilfe der Ausführungsumgebung Java Virtual Machine (JVM) innerhalb eines Kubernetes-Pods ausgeführt. Das Backend greift für die Persistenz auf eine PostgreSQL-Datenbank zu, deren Ausführung in einem weiteren Kubernetes-Pod erfolgt. Daten für die Datenbank werden in einem separaten „Persistent Volume“ verwaltet und eingebunden.
Das Artefakt des Frontends besteht aus einer Sammlung von Dateien für eine bereitzustellende Webseite, wie HTML- oder Javascript-Dateien. Diese Dateien werden (ohne deren Ausführung) mithilfe eines NGINX-Webservers innerhalb eines Kubernetes-Pods ausgeliefert.

Mithilfe eines Endnutzergeräts kann das System innerhalb eines Webbrowsers genutzt werden. Zuerst kommuniziert der Webbrowser durch einen URL-Aufruf mit dem NGINX-

Webserver der ICC, wodurch das Frontend-Artefakt geladen wird (HTTP Port 80). Anschließend erfolgt dessen Ausführung (insbesondere von den JavaScript-Dateien). Dabei werden Design-Inhalte bzw. -Konfigurationen (zum Beispiel für Schriftarten oder Icons), als „Google-Fonts“-Artefakt dargestellt, von Google-Servern (mit HTTPS, Port 80) nachgeladen.
Während der Nutzerinteraktion mit der Benutzeroberfläche im Webbrowser kann es dazu kommen, dass Funktionalität des Backends benötigt wird. Dafür greift der Webbrowser auf die HTTP-REST-Schnittstelle (Port 80, zu Port 8080 weitergeleitet) des ausgeführten Backend-Artefakts zu.

Auf die ICC wird im Rahmen der Entwicklung mit dem Endnutzergerät nicht öffentlich über das Internet, sondern mithilfe einer lokalen Kubernetes-Proxy-Weiterleitung zugegriffen. Aufgrund des Einsatzes während der Entwicklung wird auch auf verschlüsselte HTTP-Zugriffe (HTTPS) auf die Webserver des Frontends und Backends verzichtet.
Ein echter Produktionseinsatz ist möglich. Dafür wäre es sinnvoll, die HTTP-Schnittstellen der Frontend- und Backend-Webserver verschlüsselt und öffentlich bereitzustellen. Darüber hinaus ist die Bereitstellung nicht auf die ICC begrenzt und könnte, entsprechende Konfiguration vorausgesetzt, auf beliebigen Clouds erfolgen, die Docker ermöglichen. Eine zukünftige Verbesserungsmöglichkeit wäre, falls möglich, die Inhalte von Google nicht einzubinden bzw. nachladen zu müssen, um die Abhängigkeiten zu reduzieren.

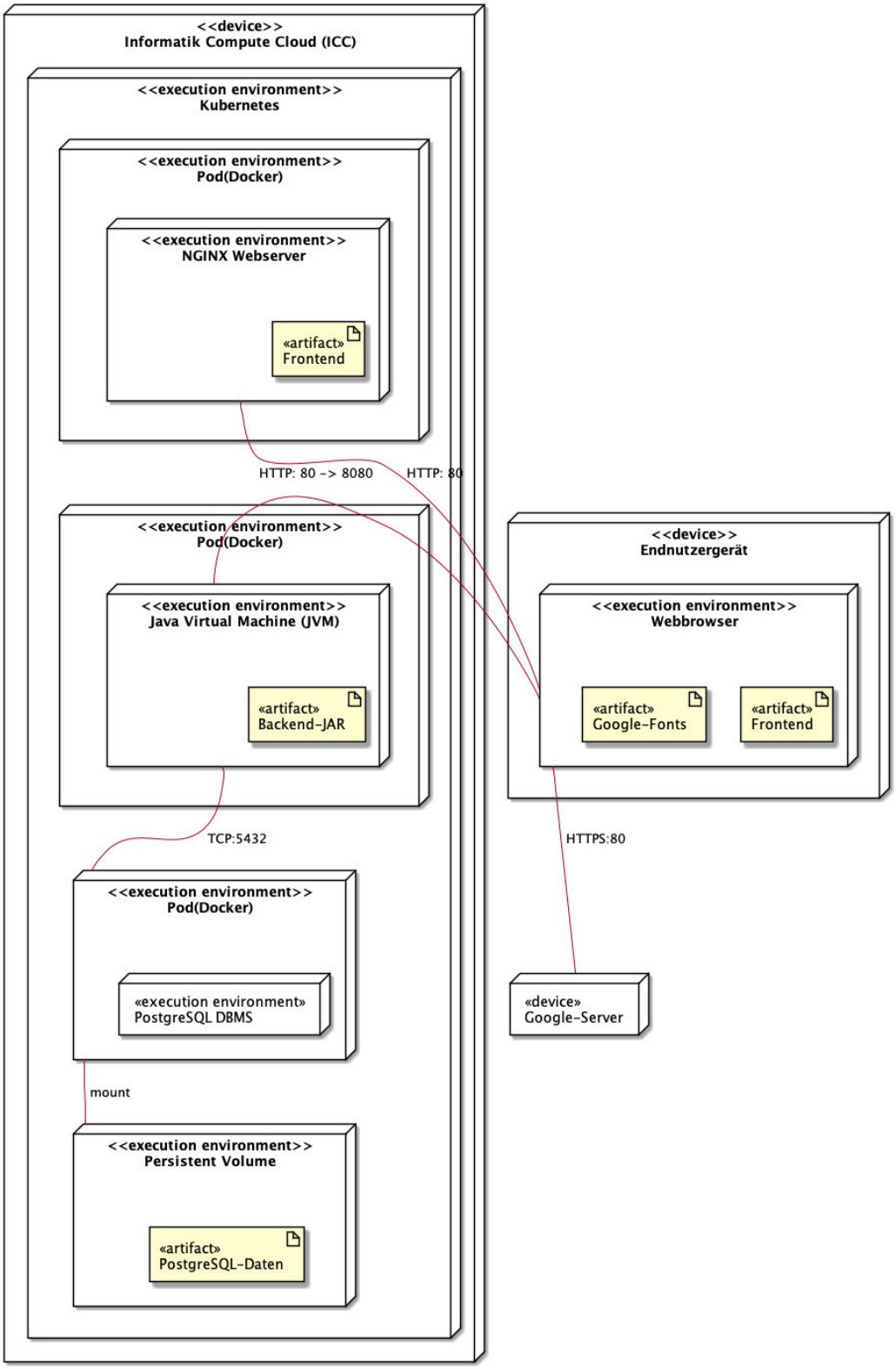

Abbildung 4.14: Verteilungsdiagramm mit Bereitstellung des Systems in der Informatik Compute Cloud der HAW (ICC)

4.10 REST-API

Es besteht eine vom Backend angebotene HTTP-REST-API, um dem Frontend zu ermöglichen, fachliche Operationen zu delegieren. Zudem dient sie der Entkopplung der beiden Subsysteme und damit der Erhöhung der Wartbarkeit. Diese Schnittstelle soll das Level 2 nach dem Richardson Maturity Model erfüllen (siehe QA-4). Das umfasst neben der Adressierbarkeit nach Ressourcen auch die Nutzung der HTTP-Verben und von aussagekräftigen HTTP-Statuscodes (vgl. [28]). Für die Repräsentation wird JSON eingesetzt.

Nachfolgende Abbildungen für die REST-API wurden mithilfe von Swagger UI erstellt.

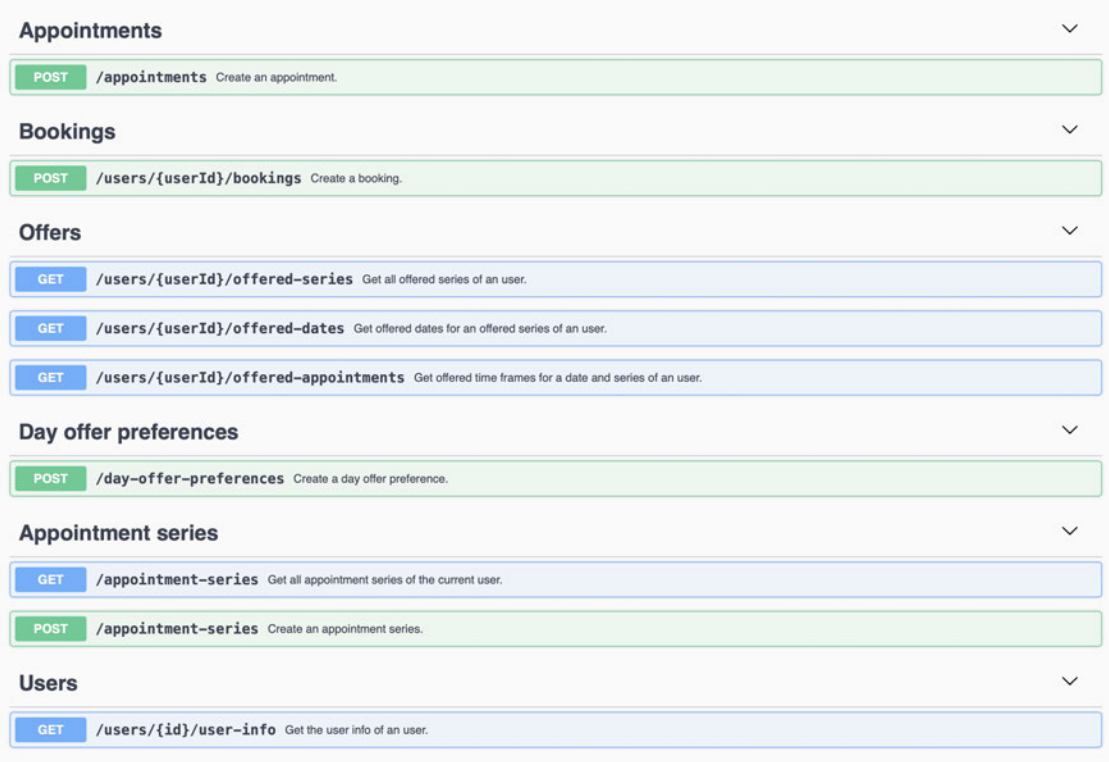

Abbildung 4.15: Überblick der REST-API

Die REST-API, wie in Abbildung 4.15 als Übersicht dargestellt, berücksichtigt die Adressierung von Ressourcen und die Nutzung von HTTP-Verben, die Ressourcenbezeichnungen sind im URI enthalten. So bestehen Schnittstellen mit den HTTP-Verben „GET“ für das idempotente Abrufen von Ressourcen und „POST“, um Ressourcen zu erstellen. Zum Beispiel können damit Terminserien mit „GET“ „/appointment-series“ abgerufen werden.

Die Möglichkeit der Erstellung einer Buchung mittels „POST“„/users/userId/bookings“ unterstreicht die Ressourcenorientierung, weil damit signalisiert wird, dass eine Buchung für den Nutzer der Id „userId“ erstellt werden soll. Der Nutzer repräsentiert in diesem Kontext den Terminanbieter für eine Buchung. Weitere HTTP-Verben wie „DELETE“ sind aufgrund der iterativen Entwicklung noch nicht vorhanden, aber vorgesehen.
Die Buchungen und Angebote werden zwar in der REST-API als Ressourcen repräsentiert, im Backend sind sie jedoch nicht als konkrete Ressourcen gespeichert. So werden die Angebote dynamisch mithilfe der Auswertung von bestehenden Terminangebotszeiträumen („day-offer-preferences“) und von bestehenden Terminen unter Beibehaltung der Idempotenz ermittelt. Für die Buchung eines angebotenen Termins wird der Termin direkt erstellt, es besteht intern im Backend keine konkrete Buchungsressource, jedoch wird beim Aufruf der Schnittstelle eine Buchungsbestätigung mit erforderlichen Informationen zurückgegeben.

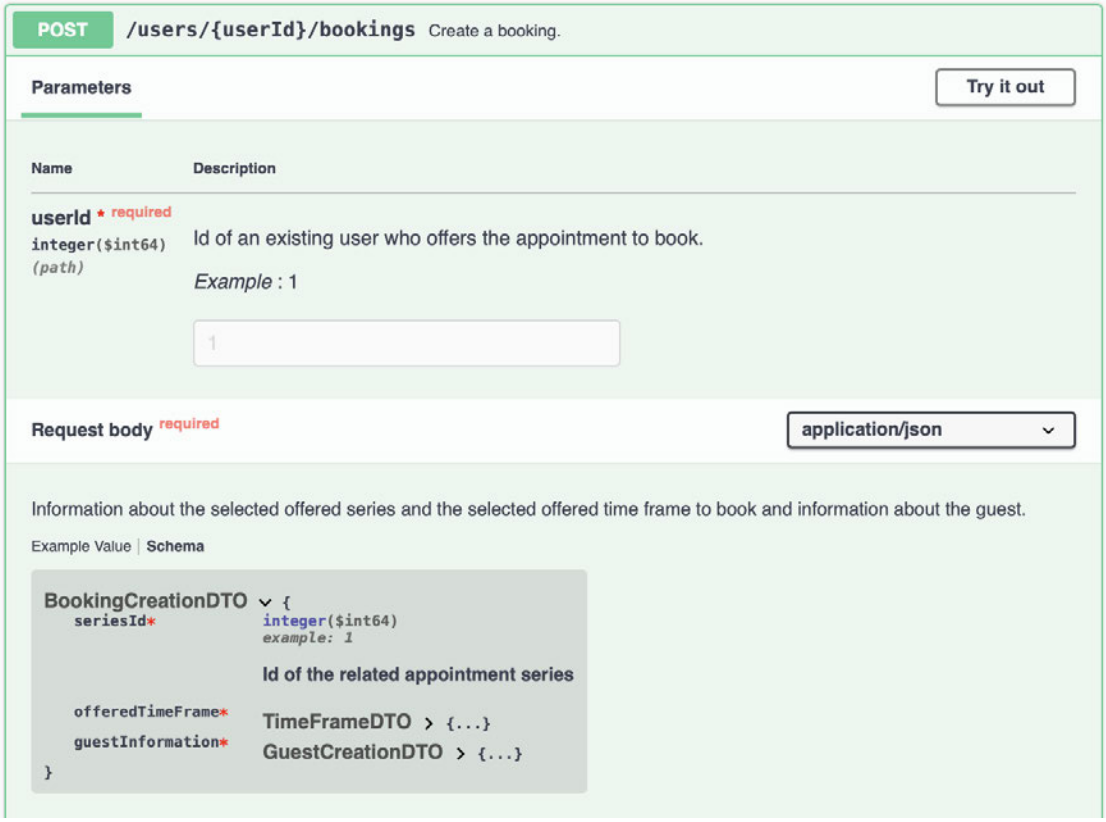

Abbildung 4.16: REST-API: Parameter für die Schnittstelle zur Buchung eines Termins

Die Schnittstelle zur Buchung eines Termins wird als Beispiel (siehe Abbildung 4.16) detaillierter beschrieben. Sie gehört zum Anwendungsfall UC-004, der Buchung eines Termins (primär Schritt 6 bis 7 des Erfolgsszenarios) und zur funktionalen Anforderung US-015.
Um eine Buchung erstellen zu können, ist es nötig, zu wissen, auf welchen Terminanbieter sich die Buchung bezieht. Das erfolgt mithilfe eines Parameters im URI, der Parameter gibt die Id des Terminanbieters an. Im Body der Anfrage wird ein DTO mitgegeben, das die Informationen enthält, für welche Terminserie des Anbieters (als Id) ein Termin gebucht werden soll, in welchem Zeitraum der Termin stattfinden soll und wie die Kontaktinformationen des Gasts, inklusive des Anliegens, lauten.

Für Rückgaben (in Abbildung 4.17 dargestellt) werden aussagekräftige HTTP-Statuscodes eingesetzt. Im Erfolgsfall signalisiert der Code 201 (Created) eine erfolgreiche Buchung des angefragten Termins, bei der der Termin erstellt wurde.
Es wird dabei auch ein DTO als Buchungsbestätigung mitgegeben. Das DTO enthält Zugangsdaten für den Gast, darunter dessen Id und ein Zugangstoken, um später auf den Termin zugreifen zu können und Änderungen zu tätigen. Ebenso enthält es ein DTO des erstellten Termins, wobei Felder, die für den Terminanbieter privat sind, zum Beispiel die Vorbereitungszeit, hier explizit null sein müssen. Diese Felder werden aufgrund der Wiederverwendung des Termin-DTOs für die Erstellung eines Termins benötigt.
Innerhalb des Termin-DTOs sind auch DTOs für alle teilnehmenden Gäste enthalten. Die DTO-Repräsentation des Gasts umfasst explizit kein Feld für das Zugangstoken und für die Angabe, ob E-Mails erwünscht werden, damit diese persönlichen Daten nicht versehentlich offengelegt werden.
Falls ungültige Argumente mitgegeben wurden, wird der HTTP-Status 400 (Bad Argument) verwendet. Dieser steht indirekt mit dem Fehlerfall 5-a (Prüfung von Formularangaben, UC-004) in Verbindung, weil bisher unerkannte, ungültige Eingaben aufgedeckt werden, auch wenn die Schnittstelle primär den Schritten 6 bis 7 des Erfolgsszenarios zuzuordnen ist. Im Falle eines fehlerhaften Aufrufs der Schnittstelle, vom Frontend ausgehend, kann auch der Fehlerfall 7-b für technische Fehler zutreffen.
Wenn die in der Anfrage angegebene Terminmöglichkeit nicht mehr verfügbar sein sollte (Fehlerfall 6-a), wird dies mit dem Fehlercode 409 (Conflict) signalisiert. Für technische Fehler (Fehlerfall 7-b) innerhalb des Backends ist der Code 500 (Internal Server Error) vorgesehen.

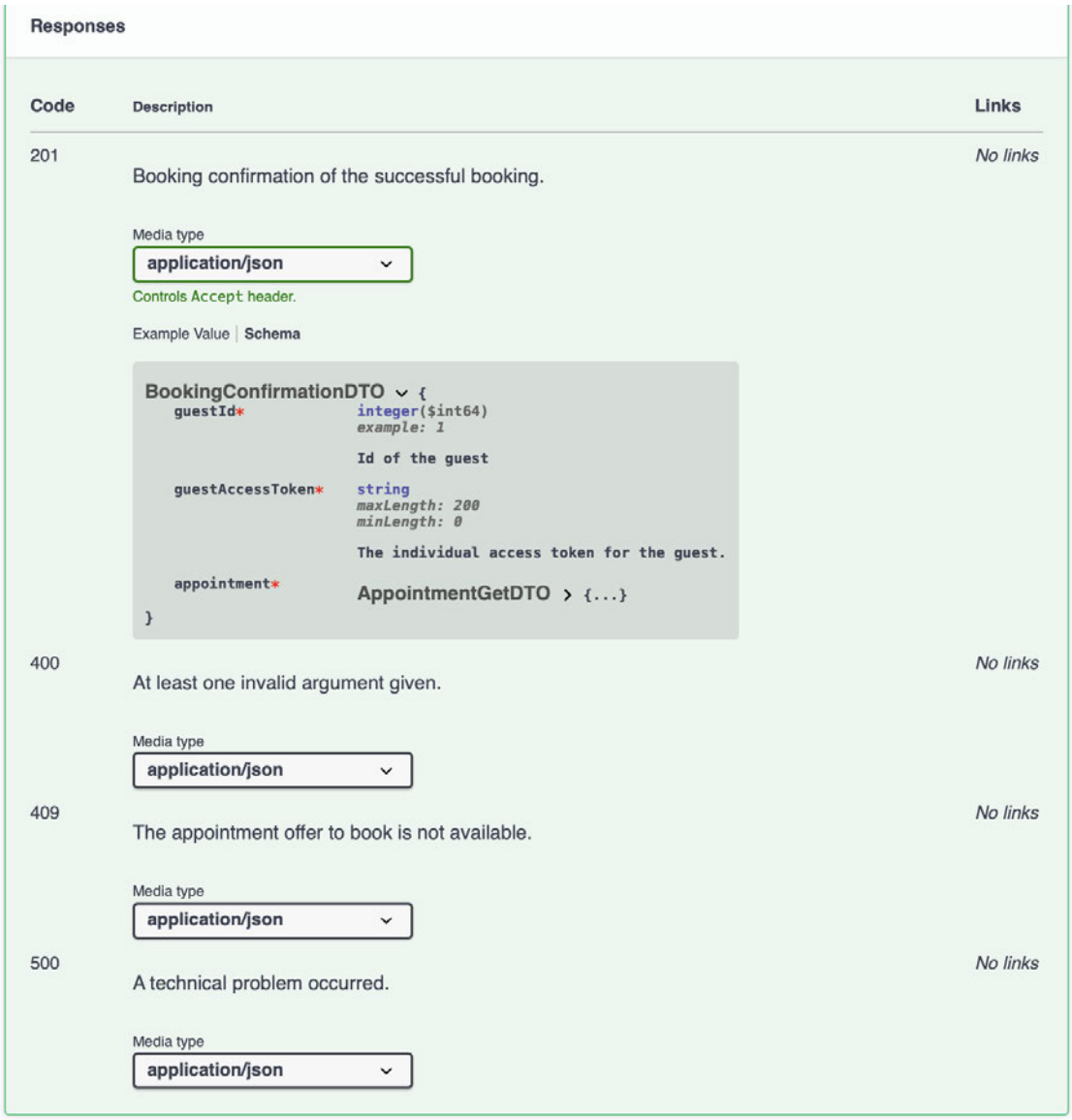

Abbildung 4.17: REST-API: Rückgaben für die Schnittstelle zur Buchung eines Termins

4.11 Auswahl von Abbildungsfunktionen für die Ermittlung einer Bewertung von Terminmöglichkeiten

Damit Termine angeboten werden können, die dem Anbieter am Besten passen (für US-013), müssen alle Terminmöglichkeiten bewertet werden. Dafür sind mehrere Kriterien vorgesehen, die aus den Serienpräferenzen der Terminangebotszeiträume hervorgehen.
Ein Kriterium ist in diesem Kontext nicht mit einer Bewertung gleichzustellen. Es definiert hier einen zu ermittelten Wert und eine Art, wie dieser für eine Bewertung zu interpretieren ist. Mithilfe einer Abbildungsfunktion erfolgt anschließend die Ermittlung einer normierten Bewertung. Ein Beispiel für ein Kriterium ist die zu ermittelnde Anzahl an freien Zwischenräumen, die zu einer besseren Bewertung führen soll, je niedriger sie ist.
Ein Typ solcher Werte ist eine Bedingung, die einen Wahrheitswert als Ergebnis liefert, zum Beispiel, ob eine Terminmöglichkeit innerhalb eines Zeitraums liegt. Darüber hinaus besteht ein anderer Typ aus reellen Zahlen. Das können zum Beispiel Abstände zwischen Zeiträumen, die Anzahl an freien Zeiträumen oder deren Standardabweichung sein. Je nach Kriterium für die Zahlen soll die Bewertung umso besser oder schlechter ausfallen, je höher ein Wert ist.

Um eine Gesamtbewertung pro Terminmöglichkeit zu bilden, wird ein gewichteter Mittelwert aller Einzelbewertungen der Kriterien ermittelt. Da pro Terminmöglichkeit variable Pausen möglich sind, erfolgt die beschriebene Gesamtbewertung jeweils für alle Pausenkombinationen einer Terminmöglichkeit. Die maximale Gesamtbewertung aus allen Pausenkombinationen bildet die Gesamtbewertung für die zugehörige Terminmöglichkeit.
Die normierte Wertemenge einer Bewertung soll das Intervall [0 , 1] sein. Je höher der Wert ist, desto besser ist die Bewertung. Eine Begrenzung der maximalen Bewertung bei 1 ist sinnvoll, um eine Aushebelung der Gewichtung zu verhindern. Ohne Maximalbegrenzung könnte die Einzelbewertung sehr hoch, zum Beispiel 1 Mio., sein und andere niedrige Bewertungen, die trotzdem eine hohe Bewertung repräsentieren sollen, trotz Gewichtung überdecken.

Eine Abbildung des ermittelten Kriteriumswerts auf die Wertemenge der Bewertung ist erforderlich. Für die Wertemenge W_f einer Abbildungsfunktion f sollte $W_f \subseteq [0\ ;\ 1]$ gelten.

Bei Kriterien mit Bedingungen erfolgt eine einfache Abbildung des Wahrheitswerts auf die Ganzzahl 0 oder 1 (siehe Abbildung 4.18).

$$D_{f_1} = \{Ja,\ Nein\};\ W_{f_1} = \{0\ ,\ 1\}$$
$$f_1(x) = \begin{cases} 1 & x = Ja \\ 0 & \text{sonst} \end{cases}$$

Abbildung 4.18: Abbildungsfunktion der Wahrheitswerte als Bewertung

Kriterien, für die eine reelle Zahl zu ermitteln ist, erfordern eine komplexere Abbildungsfunktion. Kein Kriterium, das im Rahmen dieser Arbeit berücksichtigt werden soll, weist als Ergebnis eine negative Zahl auf. Zudem besteht keine Begrenzung der Werte in die positive Richtung, sie können sich der positiven Unendlichkeit annähern. Daher sollte für eine passende Abbildungsfunktion gelten: $D = [0\ ;\ +\infty[$.
Eine weitere Anforderung besteht darin, dass f_2 injektiv und entweder streng monoton wachsend oder streng monoton fallend sein muss. So führt ein höherer Eingabewert, abhängig vom Kontext des Kriteriums, entweder zu einer höheren oder niedrigeren Bewertung.

Für die Auswahl der Abbildungsfunktionen wird eine Übersicht der Funktionstypen und deren Eigenschaften genutzt (siehe [56]). Um die beschriebenen Anforderungen zu erfüllen, erfolgen Modifikationen wie Spiegelungen, Verschiebungen, Streckungen oder Stauchungen.
Es eignen sich Potenzfunktionen mit einem negativen Exponenten und Exponentialfunktionen mit einer Basis zwischen (beides exklusive) 0 und 1. Potenzfunktionen mit negativen Exponenten wurden ausgewählt, weil der Aufwand der Berechnung mit feststehendem Exponenten gleichmäßiger als bei variablen Exponenten der Exponentialfunktionen ist.

Die konkreten ausgewählten Abbildungsfunktionen für die Ermittlung der Bewertung aus Ergebnissen von Zahlenkriterien sind in der Abbildung 4.19 abgebildet. Für Kriterien, die einen möglichst geringern Wert verfolgen, besteht die fallende Funktion f_2, während für möglichst hohe Ergebniswerte eines Kriteriums f_3 (steigend) passend ist.
Beide Funktionen enthalten die Variable a. Sie bewirkt die Streckung oder Stauchung in Richtung der x-Achse. Die Anpassung ist hilfreich für Fälle, in denen die Eingabespanne eher klein (zum Beispiel die Anzahl an freien Zeiträumen) oder groß (beispielsweise der Abstand zu einem Zeitpunkt in Minuten) ist.

$D_{f_{2,3}} = [0\ ;\ +\infty[$
$W_{f_2} =]0\ ;\ 1]$ $\quad W_{f_3} = [0\ ;\ 1[$
$f_2(x) = (ax+1)^{-1}$ $\quad f_3(x) = -(ax+1)^{-1} + 1$
mit $a \subseteq]0\ ; +\infty[$

Graph für f_2 und f_3 mit $a = 1$ (ohne Streckung/Stauchung)

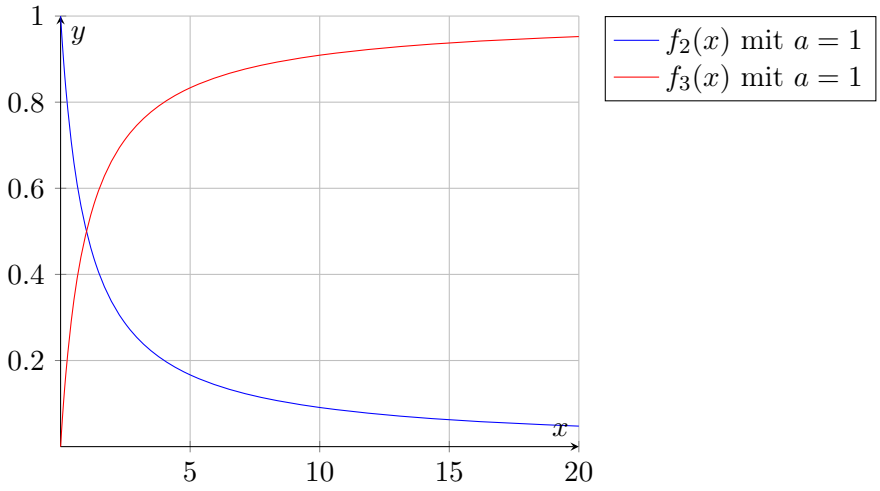

Abbildung 4.19: Monoton steigende und monoton fallende Abbildungsfunktionen mit Graph für die Bewertung nach Kriterien anhand von reellen, nichtnegativen Zahlen

4.12 Zusammenfassung

Im Entwurf wurde eine monolithische 4-Schichten-Architektur des Systems mit modularer Aufteilung festgelegt. Die Art der Benutzeroberfläche ist eine Webanwendung. Es besteht eine Aufteilung in zwei Subsysteme, das Frontend mit der Angular-Technologie und das Backend mit der Spring-Technologie.
Das System wird in der ICC ausgeführt, es erfolgt eine Anbindung an ein externes Datenbankmanagementsystem. Die Schnittstelle des Backends für eingehende Anfragen ist eine REST-API, die das Level 2 nach dem Richardson Maturity Model erfüllen soll.
Als Abbildungsfunktion für die Ermittlung von Bewertungen wurde bei Wahrheitswerten eine einfache Abbildungsfunktion und bei Zahlen eine Potenzfunktion mit negativem Exponenten ausgewählt.
Der Entwurf ist für das folgende Kapitel, Implementierung, relevant, weil er die umzusetzende Systemarchitektur festlegt.

5 Implementierung

Dieses Kapitel beschreibt die Implementierung des Systems, basierend auf dem Entwurf. Es werden Highlights und Herausforderungen dargelegt, die durch die Umsetzung entstanden sind bzw. während der Umsetzung aufgetreten sind.

5.1 Highlights

Benutzeroberfläche des Frontends

Ein Highlight des Systems ist die Benutzeroberfläche des Frontends.
Um dem Endnutzer ein modernes, ansprechendes Layout zu bieten, aber gleichzeitig den Entwicklungsaufwand für Elemente der Benutzeroberfläche zu begrenzen, ist die Bibliothek Angular Material im Frontend eingebunden. Sie umfasst Angular-Components im Material-Design (vgl. [40]). Diese werden in Verbindung mit eigenen Inhalten, wie zum Beispiel einer Schaltfläche mit eigener Beschriftung, eingesetzt.

Die Benutzeroberfläche der Abbildung 5.1 gehört zum Anwendungsfall UC-002, die Einsicht von buchbaren Terminmöglichkeiten. Die Abbildung zeigt die Benutzeroberfläche nach Schritt 8, der Anzeige von Terminbuchungsmöglichkeiten (als Zeiträume). Konkret ist die Angular-Component „BookingCreationPageComponent“ für die Angebots- und Buchungsseite (ohne Navigationsleiste) verantwortlich.
Es finden sich einige Angular-Components von Angular Material wieder. Dazu gehört bzw. gehören die obere Navigationsleiste am Seitenbeginn als Toolbar, die Schaltflächen, Trennlinien und Boxen.
Im Vergleich zum Wireframe in der Abbildung 3.3 sind alle Kernelemente repräsentiert, die zum Anwendungsfall der Einsicht buchbarer Terminmöglichkeiten gehören. Das sind zum Beispiel auswählbare angebotene Terminserien, Daten und die Anzeige von buchbaren Zeiträumen.

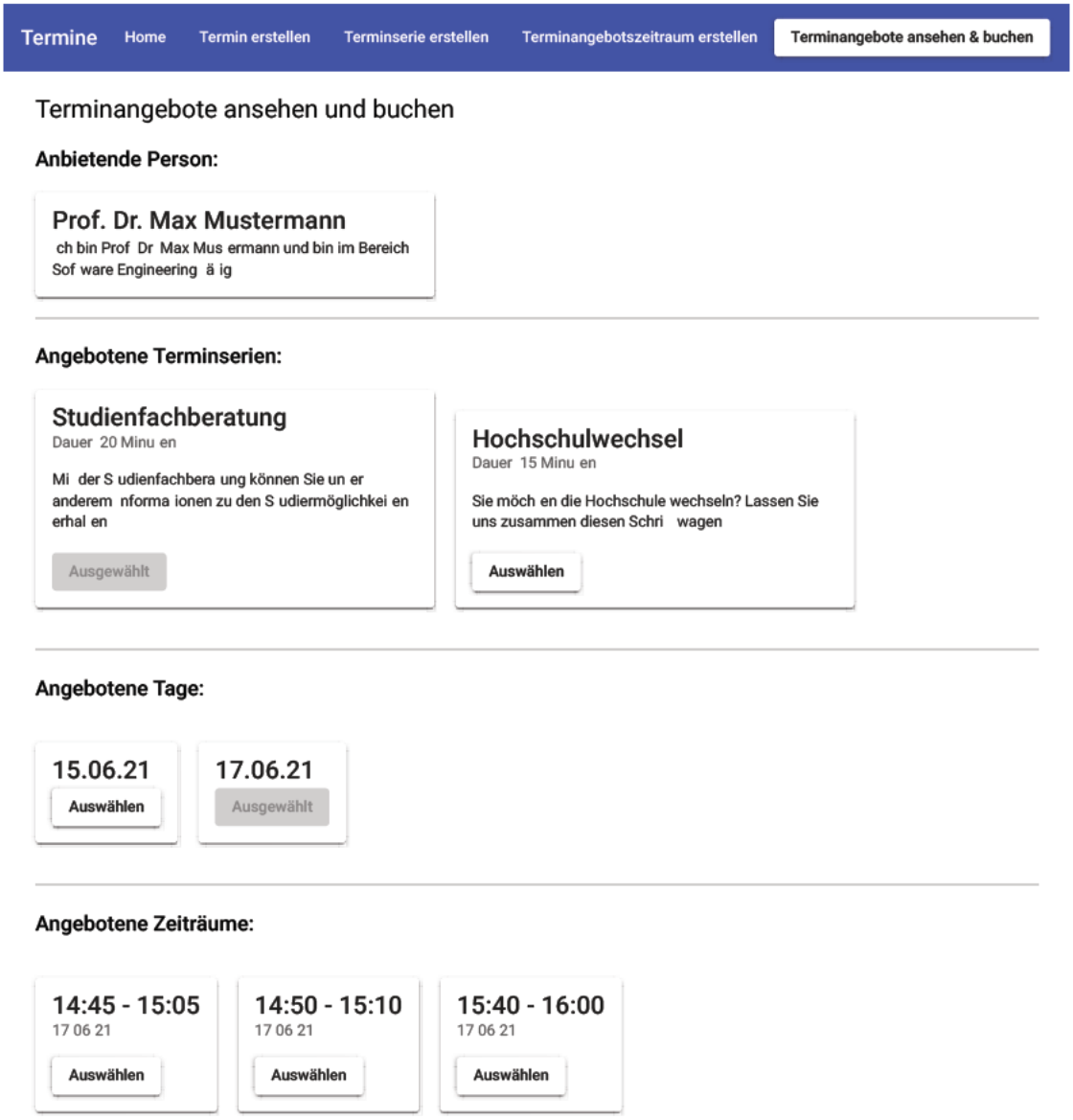

Abbildung 5.1: Benutzeroberfläche zum Einsehen buchbarer Terminmöglichkeiten

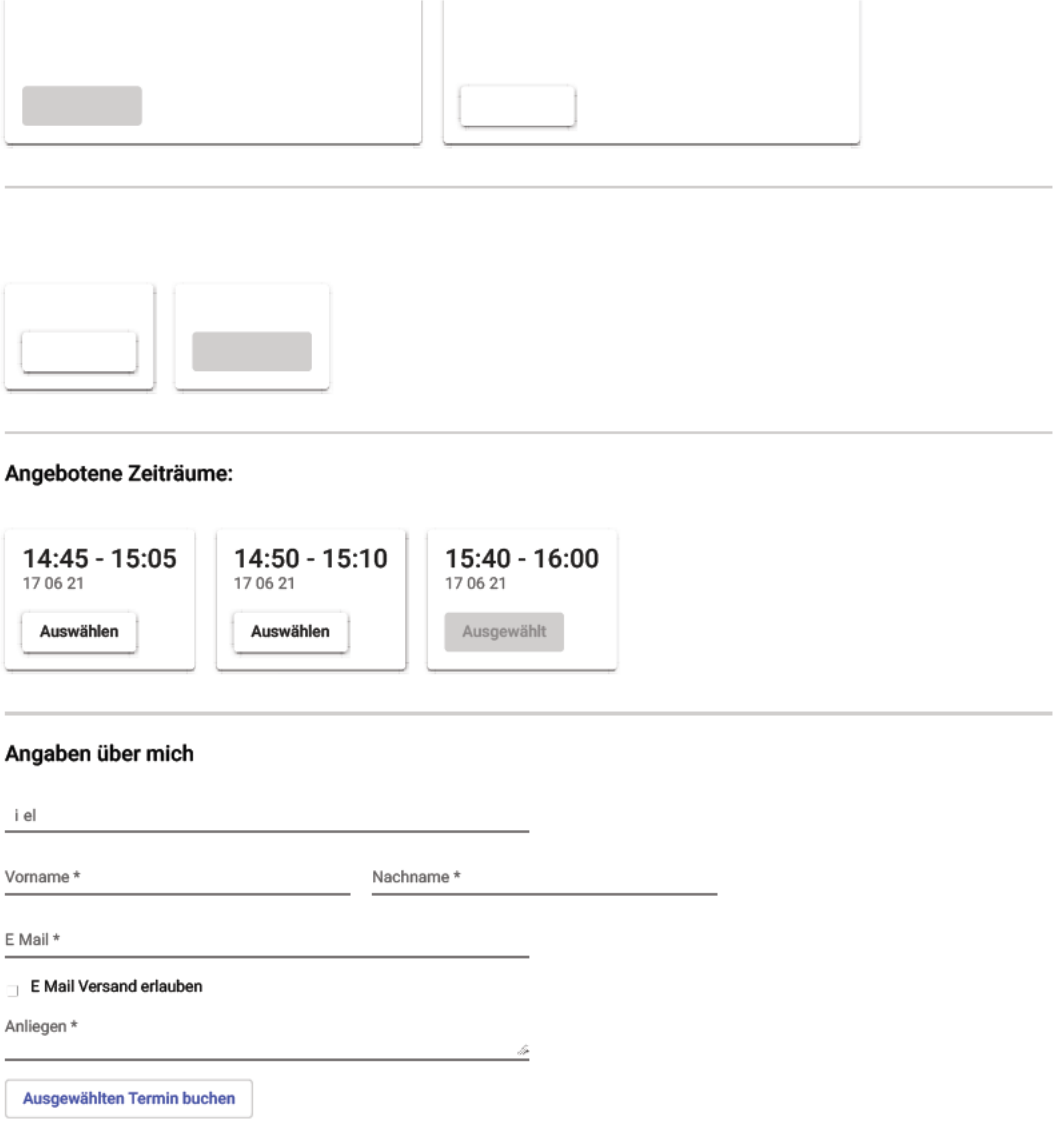

Abbildung 5.2: Erweiterung der Benutzeroberfläche für die Buchung von Terminen (zu Abbildung 5.1)

Für die Buchung von Terminen im Anwendungsfall UC-004 ist der Zeitraum einer angebotenen Terminmöglichkeit vom Nutzer auszuwählen (Schritt 1) und ein Formular wird vom System angezeigt (Schritt 2). Die Abbildung 5.2 zeigt die Benutzeroberfläche nach dem Schritt der Formularanzeige. Da ihr Anwendungsfall auf dem der Abbildung 5.1 (Einsehen buchbarer Terminmöglichkeiten) aufbaut, enthält sie aus Gründen der Vermeidung von Wiederholungen nur die Benutzeroberfläche ab den angebotenen Zeiträumen. Die Angular-Component „BookingCreationPageComponent“ ist analog zur Einsicht von Terminmöglichkeiten für die Seite verantwortlich.

Die Felder für die Formulareingaben, bis auf das Feld für die Wahrheitswertangabe, sind mit Angular-Components von Angular Material ermöglicht. Darüber hinaus erfolgt eine Überprüfung der Eingaben pro Feld, sobald der Nutzer sie einträgt. Das erleichtert die unmittelbare Korrektur der Eingaben, weil der Nutzer sofort auf Fehler hingewiesen wird.

Komplexere, feldübergreifende Prüfungen der Formulareingaben erfolgen weiterhin erst nach Betätigung der Buchung.

Ferner zeigt der Vergleich des Formulars in der Benutzeroberfläche der Abbildung 5.2 mit dem Wireframe (Abbildung 3.3), dass sich einige Beschriftungen und die Positionierung des Nachname-Felds unterscheiden.

Zusammenfassend orientiert sich die Benutzeroberfläche des Frontends an den Wireframes, wie im Beispiel der Terminangebote bzw. Terminbuchung, und es kommen einige Angular-Components von Angular Material zum Einsatz.

Strukturierung und Wiederverwendung von Angular-Components in der Benutzeroberfläche

Das Frontend weist aufgrund des eingesetzten Angular-Frameworks eine hierarchische Struktur von Angular-Components auf. Da sie die Ansicht definieren, spiegelt sich die Struktur auch in der Benutzeroberfläche wider.
Die Spitze der Hierarchie bildet „AppComponent“. Sie repräsentiert die gesamte Anwendung des Frontends und kann Angular-Components verwenden. Konkret nutzt sie eine Navigationsleiste („NavigationComponent“), unter dieser wird eine durch Routing bestimmte Angular-Component verwendet (siehe Abbildung 5.3).

Ein Beispiel ist die Benutzeroberfläche für die Erstellung eines Terminangebotszeitraums (US-009), von der in Abbildung 5.3 ein Ausschnitt zu sehen ist. Dafür ist die „DayOfferPreferenceCreationPageComponent“ zuständig. Sie umfasst auch eine Angular-Component, die für das Formular zuständig ist. Darin sind einige Formularfelder in weitere Angular-Components ausgelagert, um sie wiederverwenden zu können. Dazu gehören Felder für einen Zeitraum, die darin eingesetzten Uhrzeiten sowie die Angabe einer 5-Minuten-granulären Dauer. Für die Möglichkeit der Angabe von mehreren Pausen wird pro Pause eine „BreakCreationFormComponent“ eingesetzt.

Die Strukturierung in Angular-Components fördert neben der Modularisierung die Trennung von Zuständigkeiten, weil die Zuständigkeiten einzelnen Angular-Components zugeordnet werden können. Das kann zum Beispiel die Zuständigkeit für eine Seite oder für ein Formular sein (wie in Abbildung 5.3).
Die Trennung von Zuständigkeiten verbessert die Analysierbarkeit, weil eine Angular-Component anhand ihrer Zuständigkeit einfacher (im Quelltext) lokalisierbar ist.

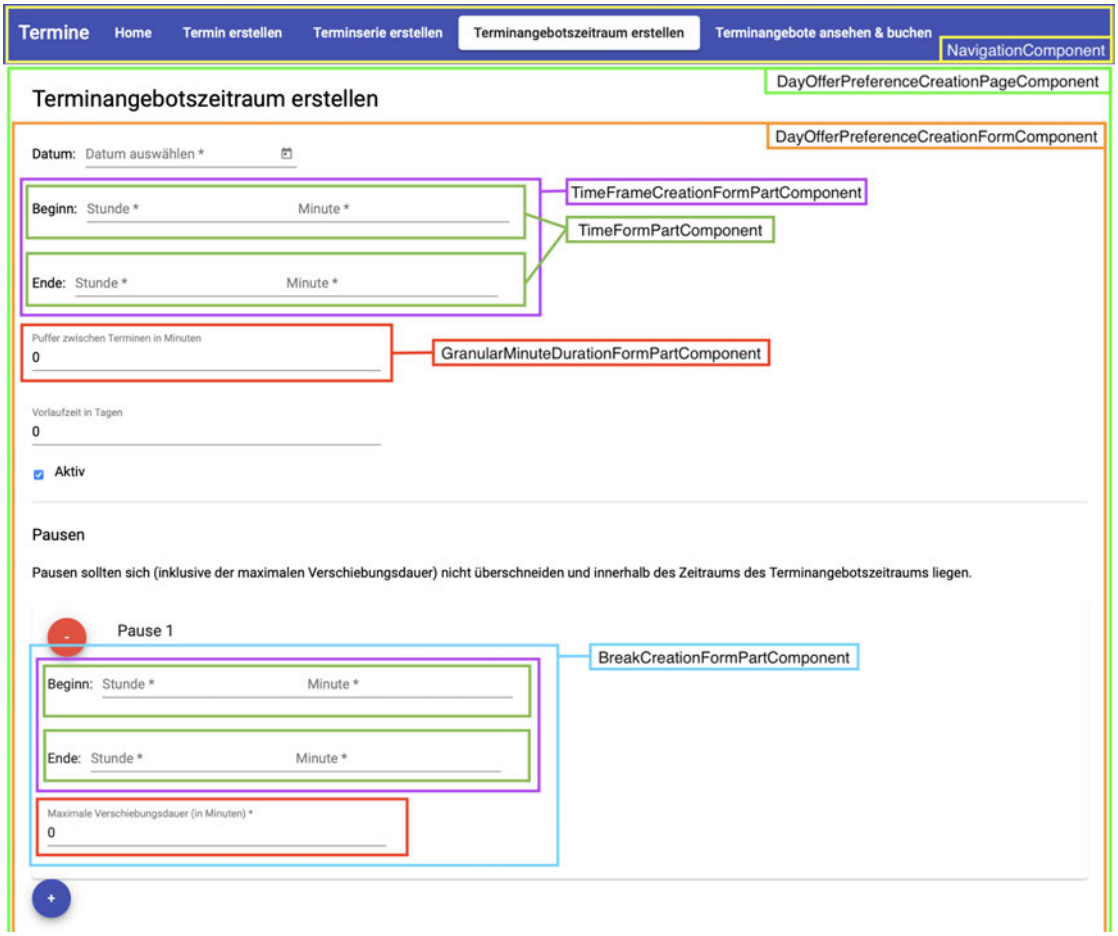

Abbildung 5.3: Ausschnitt der Benutzeroberfläche und Zuordnung zu Angular-Components für die Erstellung eines Terminangebotszeitraums

Einige mehrfach benötige Elemente der Benutzeroberfläche, wie zum Beispiel das Formularteil einer 5-Minuten-granulären Dauer, sind in eigene Angular-Components nach dem DRY-Prinzip ausgelagert, um Code-Duplikate zu vermeiden und die Wiederverwendbarkeit zu stärken. Das erhöht die Qualitätseigenschaft der Änderbarkeit, weil ein mehrfach auftretendes Element nur an einer einzigen Stelle zu ändern ist. Zudem sinkt die Fehleranfälligkeit, weil in Duplikaten mehr Fehler gemacht werden können als in einer zentralen Implementierung. Bei einer fehlerhaften Duplizierung könnten sich weitere Fehler einschleusen. Die Lesbarkeit, und damit verbunden auch die Analysierbarkeit, wird verbessert, weil der Code durch die Auslagerung übersichtlicher wird.

Zusammenfassend wirkt sich die Strukturierung in Angular-Components hinsichtlich der Trennung von Zuständigkeiten und der Wiederverwendung verbessernd auf die Wartbarkeit aus.

Bewertung von Terminmöglichkeiten im Backend

Ein Ziel des Systems ist es, Terminmöglichkeiten („AppointmentOption“), die dem Terminanbieter am Besten passen, für mögliche Terminnehmer vorzuschlagen. Die Vorlieben des Terminanbieters, nachfolgend auch Kriterien genannt, sind in den Terminangebotszeiträumen, insbesondere in den enthaltenen Serienpräferenzen, repräsentiert.
Ausgehend davon, dass bereits die Terminmöglichkeiten, inklusive flexibler Pausenmöglichkeiten, ermittelt wurden, findet die Bewertung und Ermittlung der bestbewerteten Terminmöglichkeiten nach den Kriterien in der Methode „maxRatedAppointmentOptionsFrom“ statt (siehe Abbildung 5.4). Die Methode gehört zur Klasse „ConcreteRatingService“, welche die „RatingService“-Schnittstelle der „offer.rating“-Komponente umsetzt.
Für eine Bewertung anhand der Kriterien müssen Informationen über die Terminmöglichkeiten (inkl. Pausen), die Wünsche des Terminanbieters in einer Serienpräferenz, die bestehenden Termine und allgemeine Informationen, wie gewünschte Pufferlängen oder der maximale Zeitraum des Terminangebotszeitraums, vorhanden sein.
Um die Vorlieben des Terminanbieters aus der Serienpräferenz zu berücksichtigen, wird jedem gegebenen Kriterium eine Bewertungsheuristik zugeordnet. Für die Bewertungsheuristiken wird das Strategy Pattern genutzt, wobei eine Bewertungsheuristik (als Objekt mit der „AppointmentRater“-Schnittstelle) eine Strategy ist. Jede Bewertungsheuristik analysiert die gegebenen Informationen anhand des Kriteriums, ermittelt daraus einen Wert, wie einen Wahrheitswert oder eine positive reelle Zahl, und erstellt eine Bewertung (siehe Abschnitt 4.11). Aufgrund der geringeren Kopplung durch die Schnittstelle bleiben die Bewertungsheuristiken erweiterbar, beispielsweise kann eine neue Bewertungsheuristik mit einer neuen Klasse der „AppointmentRater“-Schnittstelle erstellt werden. Abweichend zum Strategy Pattern-Konzept kommt nicht eine einzelne Bewertungsheuristik, sondern mehrere gleichzeitig (anhängig von der Anzahl an Kriterien) zum Einsatz, wobei das Ergebnis eine Liste aller Einzelbewertungen ist.

```java
@Override
public List<AppointmentOption> maxRatedAppointmentOptionsFrom(
        @NonNull List<AppointmentOption> appointmentOptions, @NonNull SeriesPreference seriesPreference,
        @NonNull DayOfferPreferenceTimeFrameAndBuffer dayOfferPreferenceTimeFrameAndBuffer,
        @NonNull List<Appointment> existingAppointments
) {
    if (isAnyElementNullWithin(appointmentOptions) || isAnyElementNullWithin(existingAppointments)) {
        throw new IllegalArgumentException("There is at least one null element in a given list");
    }

    List<AppointmentRater> strategies = appointmentRaterFactory.createRatersFrom(seriesPreference);

    // If no AppointmentRater exists, all AppointmentOption instances have the maximal rating
    if (strategies.isEmpty()) {
        return new ArrayList<>(appointmentOptions);
    }

    AppointmentRaterContext raterContext = AppointmentRaterContext.ofStrategies(
            strategies, dayOfferPreferenceTimeFrameAndBuffer, existingAppointments
    );

    List<AppointmentOptionAndRating> appointmentOptionAndRatingsSortedAscending =
            appointmentOptions.stream() Stream<AppointmentOption>
                    .map(appointmentOption ->
                            AppointmentOptionAndRating.of(
                                    appointmentOption,
                                    maxRatingFor(appointmentOption, raterContext)
                            )
                    ) Stream<AppointmentOptionAndRating>
                    .sorted()
                    .collect(Collectors.toList());

    List<AppointmentOptionAndRating> maxElements = maxElementsFromAscendingSortedList(
            appointmentOptionAndRatingsSortedAscending
    );

    return maxElements.stream()
            .map(AppointmentOptionAndRating::getAppointmentOption)
            .collect(Collectors.toList());
}
```

```java
private static Percentage maxRatingFor(AppointmentOption appointmentOption, AppointmentRaterContext raterContext) {
    TimeFrame appointmentTimeFrame = appointmentOption.getAppointmentTimeFrame();

    List<List<TimeFrame>> breakCombinations = cartesianProductOfBreakOptionsWithin(appointmentOption.getVariationsOfAllBreaks());

    // Case: There are no breaks: Early return because there is only one rating with no break
    if (breakCombinations.isEmpty()) {
        return ratingFor(appointmentTimeFrame, List.of(), raterContext);
    }

    List<Percentage> ascendingRatings = breakCombinations.stream() Stream<List<TimeFrame>>
            .map(breakCombination -> ratingFor(appointmentTimeFrame, breakCombination, raterContext)) Stream<Percentage>
            .sorted()
            .collect(Collectors.toList());

    int highestRatingIndex = ascendingRatings.size() - 1;
    return ascendingRatings.get(highestRatingIndex);
}

private static Percentage ratingFor(TimeFrame appointmentTimeFrame, List<TimeFrame> breaks, AppointmentRaterContext raterContext) {
    List<RatingAndWeighting> ratingAndWeightings = raterContext.rate(appointmentTimeFrame, breaks);
    return RatingComposer.composeWeightedRatings(ratingAndWeightings);
}
```

Abbildung 5.4: Ausschnitte der Klasse „ConcreteRatingService“ für die Ermittlung bestbewerteter Terminmöglichkeiten
Die dargestellte Formatierung kann von der Originalformatierung abweichen.

Für die konkrete Zusammenstellung der Bewertungsheuristiken aus den Wünschen in einer Serienpräferenz kommt das Abstract Factory Pattern zum Einsatz. Eine Factory ermittelt alle Präferenzen einer Serienpräferenz, wie zum Beispiel Zeitpunktpräferenzen, und ordnet ihnen passende bewertende Objektinstanzen als Strategies zu, die sie zurückgibt. Die Austauschbarkeit der Factory erhöht die Flexibilität hinsichtlich der Zuordnung von Wünschen zu Bewertungsheuristiken, zum Beispiel um zukünftig für leistungsschwache Ausführungsumgebungen einfache Heuristiken bereitzustellen.
Falls keine Strategie für die Bewertung bereitstehen sollte, der Terminanbieter also keine Wünsche angegeben hat, werden alle Terminmöglichkeiten als gleichwertig passend betrachtet und daher alle zurückgegeben.
Das System ermittelt für jede Terminmöglichkeit eine Gesamtbewertung. Dafür werden alle Einzelbewertungen der Strategies, zusammen mit einer in der Strategy-Umsetzung festgelegten Gewichtung, gesammelt und gewichtet zu einer Gesamtbewertung zusammengefasst (siehe „ratingFor“ der Abbildung 5.4). Da der Terminanbieter variable Pausenverschiebungen angeben kann und deren Variationen die Bewertung unterschiedlich beeinflussen können, werden die Bewertungen für jede mögliche Pausenkombination einer Terminmöglichkeit durchgeführt und pro Pausenkombination zusammengefasst. Anschließend erhält die untersuchte Terminmöglichkeit die bestmögliche Gesamtbewertung aller ihrer Pausenkombinationen (siehe „maxRatingFor“ der Abbildung 5.4).
Danach erfolgt die Sortierung aller Terminmöglichkeiten nach der jeweiligen Gesamtbewertung. Es werden alle Terminmöglichkeiten mit der höchsten Gesamtbewertung von der Methode „maxRatedAppointmentOptionsFrom“ (Abbildung 5.4) zurückgegeben.

Ein Beispiel für ein Kriterium ist eine möglichst hohe Standardabweichung der Länge aller freien Zeiträume. Es vermeidet, dass alle freien Zeiträume (unter Berücksichtigung der Terminmöglichkeit) gleich lang sind, damit einerseits die Termine näher aufeinander folgen und andererseits längere Zeiträume frei bleiben, zum Beispiel für weitere zu vergebende Termine. Die Abbildung 5.5 zeigt eine Implementierung der zugehörigen Bewertungsheuristik aus der Klasse „MaxStandardDeviationRater“.
Hierfür werden alle freien Zeiträume innerhalb des Terminangebotszeitraums ermittelt, die sich nicht mit bestehenden Terminen, der Terminmöglichkeit oder mit den konkreten Pausenzeiträumen überschneiden und nicht zu Puffern oder zur Vor-/Nachbereitung gehören. Von den Längen aller freien verbleibenden Zeiträume ausgehend erfolgt die Berechnung der Standardabweichung.
Um das Ergebnis für die Schnittstelle der Bewertung auf einen einheitlichen Wertebereich zwischen 0 und 1 abzubilden, wird die Standardabweichung in die Abbildungsfunktion

```
@Override
public RatingAndWeighting rate(
        @NonNull TimeFrame appointmentTimeFrame, @NonNull DayOfferPreferenceTimeFrameAndBuffer
        dayOfferPreferenceTimeFrameAndBuffer, @NonNull List<TimeFrame> breaks,
        @NonNull List<Appointment> existingAppointments) {
    if (CollectionUtils.isAnyElementNullWithin(existingAppointments)) {
        throw new IllegalArgumentException("No null element of existing appointments allowed");
    }

    List<TimeFrame> emptySpaces = EmptySpacesUtils.extractEmptySpacesFrom(
            dayOfferPreferenceTimeFrameAndBuffer.getMaximalTimeFrame(),
            appointmentTimeFrame,
            existingAppointments,
            breaks,
            dayOfferPreferenceTimeFrameAndBuffer.getBufferBetweenAppointments()
    );

    double standardDeviation = RatingFunctionUtils.standardDeviation(emptySpacesLengthsFrom(emptySpaces));

    return RatingAndWeighting.of(
            RatingFunctionUtils.negativePowerIncreasing(standardDeviation, NEGATIVE_POWER_INCREASING_FACTOR),
            WEIGHTING
    );
}
```

Abbildung 5.5: Ausschnitt der Klasse „MaxStandardDeviationRater“ für die Bewertung nach dem Kriterium einer möglichst hohen Standardabweichung von Längen freier Zeiträume
Die dargestellte Formatierung kann von der Originalformatierung abweichen.

f_3 aus der Abbildung 4.19 des Abschnitts 4.11 eingesetzt. Zuletzt wird ein festgelegter Gewichtungswert mitgegeben, der für die spätere Ermittlung einer Gesamtbewertung aus allen Einzelbewertungen wichtig ist.

5.2 Herausforderungen

Während der Implementierungsphase sind einige Herausforderungen aufgetreten. Zwei ausgewählte Herausforderungen werden im Folgenden genauer erläutert.

Unterschiedliche Repräsentationen von Referenzen in der Applikations- und Logikschicht

Für die Implementierung der REST-Schnittstelle bestehen im Backend innerhalb der Applikationsschicht RestController-Klassen, an die REST-Anfragen delegiert werden. Für die Bearbeitung von Anfragen müssen deren Inhalte ggf. an Schnittstellen von Services in

der Domänenschicht weitergereicht werden. Dafür sind unter anderem Objekte als Parameter erforderlich. In der Domänenschicht bestehen unabhängige Objektdefinitionen, die von denen der REST-Schnittstelle entkoppelt sein sollen. Daher werden für die REST-Schnittstelle in der Applikationsschicht DTOs eingesetzt, zum Beispiel für die Erstellung eines Termins.
Die DTOs sollten sich bestenfalls direkt in ein Domänenobjekt umwandeln lassen, zum Beispiel durch einen Methodenaufruf des DTO-Objekts. Das wird überwiegend so umgesetzt, konkret ruft ein RestController die Konvertierungsmethode des DTOs auf.
Für Referenzen auf persistente Entitäten besteht eine Problematik bei der Umwandlung.

```
public class SeriesPreferenceCreationDTO {

    @{...}
    private Long seriesId;

    @{...}
    private List<@NotNull @Valid TimeFrameDTO> timeFrames;

    @{...}
    private List<@NotNull @Valid PointInTimePreferenceDTO> pointInTimePreferences;
```

Abbildung 5.6: Ausschnitt der DTO-Klasse für die Erstellung einer Serienpräferenz

In den DTOs sind Referenzen durch einen Verweis auf die eindeutige Id der Entität repräsentiert, um die Information möglichst kompakt und einfach zu übermitteln. Ein Beispiel ist das DTO für eine Terminserienpräferenz (siehe Abbildung 5.6, auch Serienpräferenz genannt). Es wird für die Erstellung eines Terminangebotszeitraums eingesetzt. Hier verweist „seriesId“ auf die Id der persistierten Terminserie, auf welche sich die zu erstellende Serienpräferenz bezieht.

In der Domänenschicht sind die Referenzen auf andere Entitäten durch konkrete Objekte umgesetzt. Das wird durch Object-Relational Mapping von Hibernate ermöglicht. So enthält beispielsweise die Entitätsklasse der Serienpräferenz, wie in Abbildung 5.7 dargestellt, eine Referenz auf das Terminserienobjekt im Feld „series“.
Die Umwandlung von einer Id zum Entitätsobjekt kann nicht ohne weiteres Wissen innerhalb der DTO-Klasse erfolgen. Das Abrufen von Entitätsobjekten bei gegebenen Ids müsste an die Domänenschicht delegiert werden.
Es ist zu entscheiden, wo die Umwandlung der DTOs zu Entitätsobjekten innerhalb der Applikationsschicht geschieht, wenn Referenzen mit umzuwandeln sind. Dafür werden drei Möglichkeiten betrachtet: die Umwandlung in der DTO-Klasse mit Zugriff auf Do-

```
public class SeriesPreference implements Validatable {

    @{...}
    private Long id;

    // This series preference is not deleted automatically via delete cascade
    @{...}
    private AppointmentSeries series;

    @{...}
    private List<@NotNull TimeFrame> timeFrames;

    @{...}
    private List<@NotNull PointInTimePreference> pointInTimePreferences;
```

Abbildung 5.7: Ausschnitt der Serienpräferenz-Entitätsklasse

mänenservices, innerhalb der jeweiligen RestController-Klasse oder ausgelagert in einer Konvertierungsklasse.

Einerseits würde für die Referenzumwandlung in der DTO-Klasse sprechen, dass konsistent vorgegangen wird, also der Nutzer die Umwandlung ausnahmslos innerhalb der DTO-Klasse verorten kann. Andererseits wäre die Trennung der Zuständigkeiten nicht eindeutig, weil es ist nicht die Hauptaufgabe der DTO-Klasse ist, Konvertierungen durchzuführen, sondern das DTO zu repräsentieren. Es wird die Kopplung erhöht, weil die Klasse auf Services der Logikschicht zugreifen muss. Darüber hinaus erhöht sich die Komplexität, weil die Referenz auf eine Serviceinstanz ermittelt werden muss. Ein Nutzer würde eher erwarten, dass die DTO-Klasse simpel ist, und keine schichtübergreifenden Operationen passieren.
Die Argumente gegen die Referenzumwandlung in der DTO-Klasse, besonders die erhöhte Kopplung und die abweichende Erwartung der Nutzer überwiegen, daher wird davon abgesehen.

Vorteilhaft für Umwandlung innerhalb einer RestController-Klasse ist der geringere Zusatzaufwand für die Beschaffung der Serviceinstanz der Domänenschicht, weil in der Klasse bereits Dependency Injection eingesetzt wird. Zudem benötigt die RestController-Klasse die Umwandlung der DTOs, um später die Services der Domänenschicht mit umgewandelten Objekten aufrufen zu können. Nachteilhaft ist die größere Kopplung zur DTO-Klasse und der entsprechenden Domänenklasse, weil Wissen über die Felder für die Umwandlung nötig ist. Darüber hinaus wird dann die Komplexität der Klasse erhöht, weil Umwandlungslogik hinzukommt.

Insbesondere aufgrund der Erhöhung der Komplexität und der Kopplung für die Umwandlung soll die Lösung der Umwandlung innerhalb der RestController-Klassen nicht umgesetzt werden.

Es bleibt die Möglichkeit der Auslagerung der Umwandlung in eine neue Konvertierungsklasse. Es besteht wie bei der Lösung mit der RestController-Klasse eine hohe Kopplung zur DTO- und Entitätsklasse. Die Umsetzung für den Erhalt von Instanzen von Domänenservices ist einfach, weil Dependency Injection durch die Verwaltung einer einzelnen Klasseninstanz mithilfe von Spring ermöglicht werden kann. Die eindeutige Zuständigkeit solcher Konvertierungsklassen für die Umwandlung von DTOs, die Referenzen enthalten, ist ein großer Vorteil.

Letztlich eignet sich die Auslagerung der Umwandlung von DTO-Instanzen zu Instanzen von Entitätsklassen für enthaltene Referenzen in Konvertierungsklassen am besten, weil sie nicht die Komplexität bestehender Implementierungen weiter erhöht und eine klare Zuständigkeit der Konvertierungsklassen besteht. Die Tatsache der inkonsistenten Verortung der Umwandlung, dass für Umwandlungen von DTOs, die keine Referenzen enthalten, eine DTO-Klasse zuständig ist, und in den anderen Fällen eine Konvertierungsklasse, ist verkraftbar, weil bei den alternativen Lösungen noch größere Nachteile bestehen würden.

Hohe Laufzeit eines Tests zur Erstellung von Terminangebotszeiträumen

Bei der Entwicklung des Frontend-Subsystems für die Erstellung von Terminangebotszeiträumen (US-009) ist eine ungewöhnlich hohe Laufzeit bei einem Test, der die Erstellung eines Terminangebotszeitraums mit möglichst vielen Angaben testet, aufgefallen. Das hat unter anderem in der automatisierten Testausführung der ICC zu einem Timeout-Fehler geführt. In der lokalen Ausführung auf einem leistungsstärkerem Referenzsystem trat kein Timeout auf, aber die Laufzeit des Einzeltests war hoch (ca. 17 Sekunden). Eine noch akzeptable Laufzeit wäre beispielsweise vier Sekunden gewesen.

In diesem Test erfolgt das Ausfüllen und die Bestätigung des Formulars für einen neuen Terminangebotszeitraum, die Überprüfung der ausgehenden REST-Anfrage mit Mocking der Antwort sowie die Überprüfung der Darstellung einer erfolgreichen Erstellung. Analysen mithilfe von Zeitpunktausgaben zwischen den Schritten ergeben, dass das Ausfüllen des Formulars für die lange Laufzeit verantwortlich ist. Die hohe Laufzeit besteht nur im

Kontext der automatischen Testausführung, das Verhalten konnte durch eine manuelle Nachprüfung nicht reproduziert werden.
Im Formular werden alle vorhandenen Felder gefüllt, darunter zum Beispiel der maximale Zeitraum oder ein Puffer. Zusätzlich werden drei Pausen und drei Terminserienpräferenzen (im Folgenden mit „Serienpräferenz“ abgekürzt) dem Formular zugefügt. Pro Serienpräferenz werden zwei Zeiträume, fünf allgemeine Präferenzen, vier Zeitpunkt- und Zeitraumpräferenzen angegeben, damit jede Art einer Präferenz einmal vertreten ist.

Eine weitere Analyse der Testausführungsdauer innerhalb der Formulareingabe ergibt, dass die Angabe der Serienpräferenzen, in der Benutzeroberfläche auch buchbare Terminserien genannt, den Großteil der Laufzeit erfordert. Durch den Klick auf eine „+“-Schaltfläche beginnt die Angabe einer Serienpräferenz. Danach wird zusätzlich ein Formularteil angezeigt, der Eingaben für die neue Serienpräferenz ermöglicht. Je mehr davon erstellt werden müssen, desto mehr Formularteile dafür enthält die Seite.
Die Eingabe der Daten erfolgt in der Testimplementierung durch die Suche des letzten Seitenelements, das mit einer bestimmten Id markiert ist, und durch eine Interaktion damit. Beispielsweise wird für das Zufügen einer neuen Serienpräferenz die letzte „+“-Schaltfläche für das Hinzufügen von Serienpräferenzen gesucht und ausgewählt.

Als Hypothese wird für die Testausführung eine Laufzeit von $\mathcal{O}(n^2)$ mit der Anzahl von Serienpräferenzen n unter Vernachlässigung der variablen Anzahl von Pausen vermutet. Die Laufzeit ist vermutlich quadratisch, weil n Serienpräferenzen hinzugefügt werden müssen und die Dauer des Hinzufügens einer Serienpräferenz abhängig von der Anzahl bereits im Formular vorhandener Seitenelemente ist. Ein Grund dafür ist, dass für die Ermittlung des letzten erwünschten Seitenelements mindestens alle vorigen durchsucht werden müssen. Daher ist die durchschnittliche Dauer pro Hinzufügen einer Serienpräferenz vermutlich $(n * l)/2$ (l: konstante Dauer der Durchsuchung des Formularteils mit einer bestehenden Serienpräferenz).
Um die Hypothese zu überprüfen, erfolgt eine Laufzeitmessung des Hinzufügens der n-ten Serienpräferenz im Formular für einen Terminangebotszeitraum während der automatischen Testausführung mithilfe von Konsolenausgaben. Es wird nicht die Gesamtlaufzeit des kompletten Tests, sondern die tatsächliche Dauer des Hinzufügens je Serienpräferenz gemessen, um spezifischere Erkenntnisse darüber zu erlangen. Die gemessene Laufzeit pro n-te hinzugefügte Serienpräferenz sollte linear sein, weil n Formularteile bestehen, die für das Ausfüllen zu durchsuchen sind.

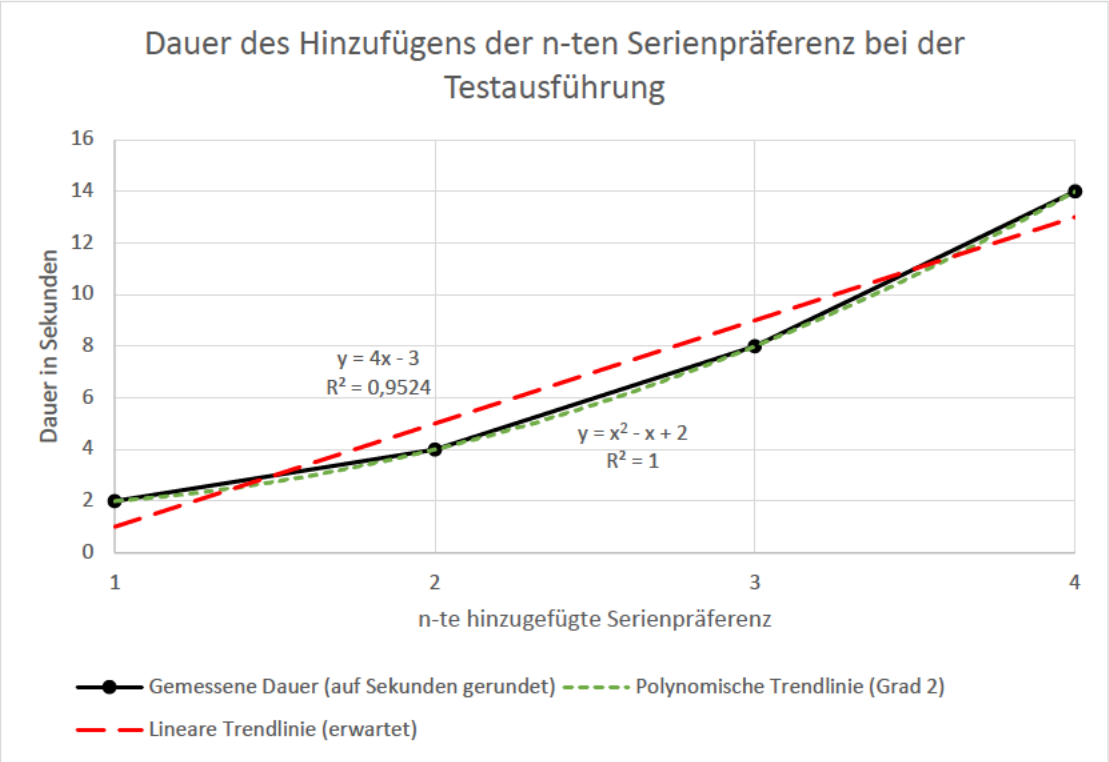

Abbildung 5.8: Dauer des Hinzufügens der n-ten Serienpräferenz bei der Testausführung

Die Ausführung der Messung mit einer Rundung auf Sekunden erfolgte auf einem macOS-System mit 16 GB Arbeitsspeicher und einem 3,6 GHz Quad-Core Prozessor. Aus Zeit- und Priorisierungsgründen wurden wenige weitere Messungen desselben Aufbaus durchgeführt, sie stimmten gerundet überein. Eine Begrenzung auf maximal vier Serienpräferenzen ist notwendig, weil eine höhere Anzahl zu einem Timeout führen würde.

Die Ergebnisse der Messung, in Abbildung 5.8 dargestellt, weisen keine konstante Steigung auf, die bei dem erwarteten linearen Verhalten gegeben sein müsste. Im Vergleich mit einer linearen und quadratischen Trendlinie (beide mithilfe von Excel erzeugt), besteht bei beiden Trendlinien ein hohes Bestimmtheitsmaß, jedoch ist es bei der quadratischen Trendlinie maximal.
Eine quadratische Laufzeit ließe sich nicht durch die eigene Implementierung erklären. Möglich ist, dass eingesetzte Testwerkzeuge von Angular Material für den Zugriff auf Seitenelemente eine erhöhte Laufzeitkomplexität aufweisen.

Die Ergebnisse haben eine geringe Aussagekraft, weil die Werte durch die Rundung auf Sekunden ungenau sind und aufgrund des begrenzenden Timeouts die Anzahl von ma-

ximal vier hinzuzufügenden Serienpräferenzen zu gering ist. Zudem wurden zu wenig Wiederholungen durchgeführt, sodass sich verzerrende Leistungsschwankungen besonders in den Ergebnissen widerspiegeln können.
Da die polynomische Trendlinie einen höheren Grad als die lineare Trendlinie aufweist, kann sie (insbesondere) bei der geringen Anzahl von hinzuzufügenden Serienpräferenzen besser an die Messergebnisse angepasst werden. Demnach kann sie bei einer höheren Anzahl stärker von den Messungen abweichen.

Wegen der geringen Aussagekraft kann nicht bestimmt werden, welches Laufzeitverhalten für das Hinzufügen der n-ten Serienpräferenz besteht. Eine konstante Laufzeit lässt sich jedoch grob ausschließen, weil die Messungen nicht konstant sind, somit scheint das Laufzeitverhalten mindestens linear zu sein.
Letztlich kann für den Gesamtvorgang der Erstellung eines Terminangebotszeitraums mindestens eine quadratische Laufzeitkomplexität vermutet werden, weil sie pro einzelner Serienpräferenz mindestens linear zu sein scheint, auch ein höherer Grad ist möglich.

Um die hohe Laufzeit bei Tests zu umgehen, ist die Anzahl der beim Test hinzuzufügenden Serienpräferenzen auf zwei begrenzt, um eine noch vertretbare Laufzeit zu haben und gleichzeitig die Möglichkeit mehrerer Serienpräferenzen testen zu können.

5.3 Zusammenfassung

Als Resultat der Implementierung ist die Benutzeroberfläche mit Einsatz von Angular Material und dynamisch überprüften Eingabefeldern ein Highlight. Als weiteres Highlight wird die Strukturierung und Wiederverwendung von Angular-Components anhand eines Benutzeroberflächenausschnitts demonstriert. Für das Backend ist die Bewertung von Terminmöglichkeiten mit einem Beispiel einer Bewertungsimplementierung ein Highlight.
Herausforderungen sind die unterschiedlichen Repräsentationen von Referenzen in der Applikations- und Logikschicht des Backends, die mithilfe von Konvertierungsklassen umgewandelt werden. Zudem trat bei der Frontend-Implementierung eine hohe Laufzeit bei einem (automatischen) Test der Erstellung von Terminangebotszeiträumen mit mehreren Serienpräferenzen auf. Die Laufzeitkomplexität ließ sich anhand von Laufzeitmessungen nicht eindeutig bestimmen und wird als mindestens quadratisch eingeschätzt. Das Problem wurde durch eine Begrenzung der Anzahl von im Test hinzuzufügenden Serienpräferenzen umgangen.

6 Evaluation

In diesem Kapitel wird das im Rahmen dieser Arbeit implementierte System dahingehend evaluiert, inwiefern es die funktionalen Anforderungen und Qualitätsanforderungen aus der Spezifikation erfüllt.
Um die Überprüfung der Qualitätsanforderung des Zeitverhaltens (QA-2) zu unterstützen, erfolgt vorher eine Durchführung der zugehörigen Qualitätsattributsszenarien.

6.1 Funktionale Anforderungen

Zu Beginn der Entwicklung wurde erwartet, dass das System aufgrund der iterativen Vorgehensweise und der begrenzten Ressourcen im Rahmen der Bachelorarbeit zumindest diejenigen funktionalen Anforderungen umsetzen sollte, die als Basis für das Ziel der Vereinbarung von Terminen unter Berücksichtigung von Wünschen des Terminanbieters anzusehen sind.

Die folgenden funktionalen Anforderungen wurden mit dem System umgesetzt und erfüllt. Nicht erfüllte funktionale Anforderungen wurden wegen der hohen Anzahl an ermittelten funktionalen Anforderungen nicht dargestellt.

- US-001: Als Administrator kann ich als Artefakte konfigurierbare Docker-Images erzeugen, um das System betreiben zu können.
- US-002: Als Administrator kann ich Systemereignisse und Fehlermeldungen persistent protokollieren lassen, um getätigte Aktionen im Zweifel nachweisen zu können (bezieht sich nur auf Systemteile, die fachliche Daten verwalten und auch fachliche Funktionalität bieten).
- US-003: Als interner Mitarbeiter kann ich einen Einzeltermin erstellen, damit außerhalb des Systems bestehende Termine auch berücksichtigt werden.

- US-006: Als interner Mitarbeiter kann ich eine Terminserie erstellen, um diese für zukünftige zu vereinbarende Einzeltermine zu nutzen.
- US-009: Als interner Mitarbeiter kann ich einen Terminangebotszeitraum für Terminserien erstellen, damit andere Nutzer einfach Einzeltermine buchen können.
- US-012: Als Student oder externe Person kann ich angebotene Terminserien einer Person einsehen, um eine Vorlage für einen Einzeltermin auszusuchen.
- US-013: Als Student oder externe Person kann ich buchbare Terminmöglichkeiten einer Terminserie einsehen, um einen für mich passenden Zeitpunkt für einen Einzeltermin zu finden.
- US-015: Als Student oder externe Person kann ich eine Terminmöglichkeit buchen, um einen Einzeltermin festzulegen.

Die erfüllten funktionalen Anforderungen stellen, bis auf diejenigen für die Protokollierung von Systemereignissen (US-002) und Erzeugung von Docker-Images (US-001), die funktionalen Kernanforderungen für das Ziel des Systems dar.
Das Buchen eines Termins (US-0015) ist erst möglich, wenn Terminangebote bestehen und einsehbar sind (US-012 und US-013). Für Terminangebote ist die vorherige Erstellung von Terminangebotszeiträumen (US-009) notwendig. Da Terminangebotszeiträume Angebote für Terminserien definieren, müssen diese vorher erstellt werden können (US-006). Zudem müssen für Terminangebote bestehende Termine berücksichtigt werden können. Um sie einzutragen, ist die Erstellung von Einzelterminen notwendig (US-003).
Da die Funktionalität für die Ansicht bestehender Termine und Benachrichtigungen nicht umgesetzt wurde, ist es dem Terminanbieter nicht möglich, zu wissen, welche Termine gebucht wurden.
Die erfüllten funktionalen Anforderungen ermöglichen die Kernaufgabe des Systems, das Anbieten von Terminen. Die fehlende Möglichkeit des Terminanbieters, Informationen über gebuchte Termine zu erhalten, ist zwar für einen produktiven Einsatz notwendig, wird hier jedoch nicht zu den funktionalen Kernanforderungen gezählt. Die Buchung eines Termins erfolgt im System, unabhängig davon, ob im Nachhinein beteiligte Personen informiert werden.
Die funktionalen Anforderungen für die Protokollierung (US-002) und die Erzeugung von Docker-Images (US-001) wurden umgesetzt, um jede Nutzerrolle, auch die des Administrators, zu berücksichtigen.

Somit wurden die eigenen Erwartungen an das System hinsichtlich der funktionalen Anforderungen erfüllt, weil die Kernanforderungen für die Vereinbarung von Terminen unter Berücksichtigung von Wünschen des Terminanbieters umgesetzt wurden, auch wenn zusätzliche funktionale Anforderungen für einen produktiven Einsatz erforderlich sind.

6.2 Qualitätsattributsszenarien

Für die Überprüfung der Qualitätsanforderung QA-2 zum Zeitverhalten (Reaktion in mindestens 85% der Fälle innerhalb von vier Sekunden im Normalzustand) bestehen zwei Qualitätsattributsszenarien. Ein Qualitätsattributsszenario orientiert sich an der Dauer eines Benutzeroberflächenaufrufs (QAS-01 in Tabelle 3.7) und das andere an der Dauer des Vorganganstoßes bis zur Bestätigungsanzeige (QAS-02 in Tabelle 3.8).

Aufbau der Testumgebung

Die Erwartungen folgen den in den Qualitätsattributsszenarien beschriebenen Antwortmaßen. So soll die gemessene Dauer in mindestens 85 % der Fälle maximal vier Sekunden betragen. Das gilt für die Dauer, bis die (durch eine URL) aufgerufene Benutzeroberfläche vollständig geladen und angezeigt ist (QAS-01) und für die Dauer zwischen dem Anstoß eines Vorgangs bis zur Anzeige einer Bestätigung (QAS-02).

Beide Szenarien werden für gängige Anwendungsfälle, die Erstellung eines Terminangebotszeitraums (UC-006 im Anhang, Abschnitt A.3) und die Buchung eines Termins (UC-004, Abschnitt 3.8), manuell durchgeführt.

Für den Anwendungsfall der Buchung (UC-004, Abschnitt 3.8) wird der Seitenaufruf durch den Schritt 1 von UC-001 (Ansicht angebotener Terminserien, im Anhang, Abschnitt A.3)) repräsentiert. Der Anstoß des Buchungsvorgangs bis zur Darstellung der Bestätigung ist den Schritten 4 bis 8 (von UC-004) zuzuordnen.
Bei dem Anwendungsfall zur Erstellung eines Terminangebotszeitraums (UC-006 im Anhang, Abschnitt A.3) wären dem Seitenaufruf die Schritte 1 bis 2 zuzuordnen. Das erfolgt momentan durch einen einzigen URL-Aufruf ohne Anmeldung, weil die Funktionalität der Anmeldung (US-017) noch nicht umgesetzt wurde. Zum Anstoß der Erstellung bis zur Anzeige der Bestätigung gehören die Schritte 4 bis 7.

Für die erfolgreiche Erstellung eines Terminangebotszeitraums wird ein Zeitraum mit einer Pause und einer Serienpräferenz, die eine Zeitraumpräferenz umfasst, erstellt. Für jede Ausführungswiederholung wird ein anderes Datum genutzt, der Speicher der Datenbank für das Backend wird nicht bei jeder Wiederholung zurückgesetzt. Für die erfolgreiche Buchung eines Terminangebots mit fiktiven Gastinformationen sind vorher erstellte Terminangebotszeiträume vorausgesetzt, damit Terminmöglichkeiten bestehen.

Der Aufbau erfolgt gemäß der Beschreibung der Verteilungssicht im Abschnitt 4.9. Auf das in der ICC bereitgestellte System wird mit einem Proxy durch Kubernetes zugegriffen. Der Zugriff auf die Benutzeroberfläche erfolgt ausschließlich mit dem Webbrowser Safari auf einem macOS-System mit 8 GB Arbeitsspeicher, einem 2,3 GHz Quad-Core Prozessor und einer WLAN-Internetverbindung.
Die Durchfühung jedes Qualitätsattributsszenarios erfolgt zehnfach. Für die Messungen ist der Webbrowser auf halber vertikalen Bildschirmgröße geöffnet. Für das Szenario des Aufrufs der Benutzeroberfläche durch eine URL (QAS-01) wird vor jeder Testausführung zusätzlich der Cache des Webbrowsers geleert. Die Messungen erfolgen mithilfe der Entwickler-Tools des Browsers.

Es wird in der Zeitleiste der Abstand zwischen der ersten und letzten Aktion des Browsers (der Rubriken Netzwerkanfragen, Layout & Rendern und JavaScript & Ereignisse) ermittelt. Hierfür werden nur Browser-Aktionen betrachtet, die sich auf das Laden einer Seite bzw. auf den Anstoß eines Vorgangs bis zur Darstellung der Benutzeroberfläche beziehen. Die Dauer wird in Sekunden, auf zwei Nachkommastellen gerundet, notiert.

Ergebnisse und Beobachtungen

Bei der Ausführung des Szenarios für den Benutzeroberflächenaufruf (QAS-01 in Tabelle 3.7) wurden für die Seite zur Erstellung eines Terminangebotszeitraums durchschnittlich 2,69 Sekunden (Spanne von 2,18 bis 3,24 Sekunden) gemessen. Für den Aufruf der Buchungsseite, die bei einem Erstaufruf angebotene Terminserien anzeigt, wurden durchschnittlich 2,21 Sekunden (Spanne von 1,92 bis 2,92 Sekunden) gemessen. Für eine Ansicht der Messwerte siehe Tabelle A.1 im Anhang.

Die Durchführung des Szenarios für den Anstoß eines Vorgangs bis zur Bestätigungsanzeige (QAS-02 in Tabelle 3.8) hat bei der Erstellung eines Terminangebotszeitraums eine Dauer von durchschnittlich 0,98 Sekunden (Spanne von 0,96 bis 1,05 Sekunden) ergeben,

bei einer Buchung waren es durchschnittlich 0,87 Sekunden (Spanne von 0,86 bis 0,88 Sekunden). Für eine Ansicht der Messwerte siehe Tabelle A.2 im Anhang.

Die Entwickler-Tools von Safari erschwerten aufgrund der grafischen Darstellung eine exakte Ermittlung der Gesamtdauer, diese wurde durch die grafische Auswahl eines Ausschnitts angenähert. Teilweise war es unklar, welche Browseraktion das Ende für die Anzeige einer Vorgangsbestätigung markiert, denn es traten zwischenzeitlich weitere Browser-Aktionen für Layout & Rendern sowie JavaScript & Ereignisse auf. Im Zweifel wurde für das Ende die letzte, infrage kommende Browseraktion genutzt.
Es fiel auf, dass zeitweise kein Internetzugriff des Endgeräts bestand. Der Zugriff auf das bereitgestellte System mithilfe des Kubernetes-Proxies funktionierte weiterhin. Beim Aufruf der Benutzeroberfläche wurde dann das Laden nicht vollständig abgeschlossen. Dadurch beeinträchtigte Messungen wurden verworfen.

Diskussion und Bewertung

Die Messungen zeigen wie erwartet in beiden Szenarien durchgehend eine Gesamtdauer von maximal vier Sekunden auf.

Beim Vergleich der Messergebnisse beider Szenarien fällt für den Aufruf der Benutzeroberfläche (QAS-01) eine größere Laufzeit mit größeren Schwankungen auf, als es beim Szenario vom Anstoß eines Vorgangs bis zur Darstellung einer Bestätigung (QAS-02) der Fall ist.
Es lässt sich dadurch erklären, dass die komplette Seite geladen werden muss, was mehr Abfragen und größere Datenmengen erfordert. Zudem müssen Daten für Design-Inhalte extern von Google geladen werden. Die größere Menge, Ziellokalität und Größe der geladenen Inhalte können dazu führen, dass Schwankungen bei der Geschwindigkeit und Zuverlässigkeit der Internetverbindung sich in den Messungen widerspiegeln. Die Dauer ist wahrscheinlich aufgrund der erhöhten Menge der (teils parallelen) Datenflüsse höher.

Für QAS-02 war die Dauer vermutlich geringer, weil weniger Aufwand für die Ausführung erforderlich ist. Die Seite ist schon geladen und wird bereits im Browser ausgeführt. Es erfolgen nur asynchrone HTTP-Anfragen ans Backend, um den vom Endnutzer initiierten Vorgang bearbeiten zu können. Das sind weniger HTTP-Anfragen und ist eine geringere Datenmenge als bei dem Neuabrufen der kompletten Benutzeroberfläche. Im

Browser erfolgt nur eine Verarbeitung der Antwort und eine entsprechende aktualisierte Darstellung.

Es war überraschend, dass der zeitweilige Internetausfall das Abrufen der Benutzeroberfläche, jedoch nicht den Abschluss eines Vorgangs beeinträchtigt hat. Das lässt sich durch das Laden von Design-Inhalten von Google für die Benutzeroberfläche erklären. Für den Abschluss eines Vorgangs wie die Buchung wurde die Seite schon vorher geladen und es wurde dabei nur ein asynchroner Aufruf der REST-API des Backends durchgeführt, der nicht von dem Ausfall betroffen war.
Eine vermutete Ursache für den zeitweiligen Internetausfall ist eine installierte Virenschutzlösung. Nach deren Deinstallation traten keine derartigen Probleme mehr auf.

Kleine Messungenauigkeiten sind aufgrund der grafischen Ermittlung der Laufzeit in der Zeitleiste des Browsers möglich. Zudem erhöht die schwierige Ermittlung des Endes die Messungenauigkeiten. Dadurch kann das Ergebnis verfälscht werden, indem es etwas höher ausfällt, als es tatsächlich ist. Entsprechende Abweichungen werden auf maximal ca. 30 Millisekunden geschätzt.
Darüber hinaus können unterschiedliche Auslastungen der ICC und des Endbenutzergeräts zu Schwankungen der gemessenen Laufzeiten führen.

Die Aussagekraft des Ergebnisses wird nicht durch verfälschende Einflüsse gemindert, weil der Abstand zur maximal erwarteten Laufzeit mit einer bzw. drei Sekunden ausreichend groß ist. Es wäre sinnvoll, die Tests öfter als zehn Mal durchzuführen, jedoch wurde dies wegen eines zu hohen Aufwands für manuelle Tests nicht gemacht. Dadurch sind die Messungen nur bedingt aussagekräftig. Zudem kann es bei abweichend ausgestatteten Endnutzergeräten fürs Frontend bzw. Servern fürs Backend, einer abweichenden Stabilität der Internetverbindung, einer abweichenden Systemauslastung oder aus anderen Gründen zu anderen Laufzeiten kommen.

Insgesamt erfüllen die Messungen die spezifizierten Qualitätsattributsszenarien, auch wenn eine geringe Testwiederholung die Aussagekraft reduziert. Sie liefern allerdings ein Indiz für die zu erwartende Laufzeit, wenn das System im Einsatz ist.

6.3 Qualitätsanforderungen

Es wird erwartet, dass alle Qualitätsanforderungen erfüllt werden, abgesehen von denen, die aufgrund von bisher nicht erfüllten funktionalen Anforderungen oder aus anderen wichtigen Gründen nicht erfüllt werden können.

In diesem Abschnitt erfolgt eine Überprüfung pro Qualitätsanforderung. Für die Überprüfung der Zeitverhaltens-Qualitätsanforderung (QA-2) werden die ermittelten Ergebnisse der Ausführung der Qualitätsattributsszenarien des vorangehenden Abschnitts berücksichtigt.
Aufgrund der Menge an Qualitätsanforderungen erfolgt eine kompakte Beschreibung des Sachverhalts.

Erfüllte Qualitätsanforderungen

QA-1: Die Korrektheit umgesetzter funktionaler Anforderungen wird mittels Tests in einem nach Ansicht des Entwicklers angemessenen Umfang überprüft, und alle Tests der Funktionen sind hinsichtlich funktionaler Korrektheit erfolgreich.
Das Backend und Frontend wurden unabhängig voneinander getestet. Alle Tests sind erfolgreich. Mindestens je ein Test für den Erfolgs- und Fehlerfall (falls anwendbar) wird erwartet und wurde umgesetzt. Tatsächlich wurden unter anderem für Erfolgsfälle Tests mit minimalen und (möglichst) maximalen Angaben erstellt.
Für die funktionalen Anforderungen US-001 (Erzeugung konfigurierbarer Docker-Images) und US-002 (Persistenz von Systemereignissen und Fehlermeldungen) wird nur ein Erfolgsfall erwartet, deren Umsetzung wurde manuell getestet.
Fürs Backend wird die REST-Schnittstelle daraufhin getestet, dass die Anfragenbearbeitung wie erwartet durchgeführt wird und eine passende Antwort zurückkommt. Im Frontend wird jede Seite mit Eingaben getestet, ob sie sich wie erwartet verhält, ggf. eine REST-Anfrage stellt und bei einer Antwort korrekte Daten darstellt.

QA-2: Das System reagiert in mindestens 85% der Fälle innerhalb von 4 Sekunden bei geringer Auslastung.
Die Überprüfung der Qualitätsattributsszenarien für die Laufzeit des Ladens bzw. der Darstellung der Benutzeroberfläche und des Abschlusses eines Vorgangs haben in 100 %

der Fälle eine Laufzeit (bzw. Reaktionszeit) von maximal vier Sekunden ergeben (siehe Abschnitt 6.2).

QA-3 und QA-19 (Zusammenfassung): Das System ist als (mindestens) ein Docker-Container lauffähig.
Das System ist mit Docker-Images im Rahmen der Überprüfung der Qualitätsattributsszenarien (siehe Abschnitt 6.2) lauffähig. Der Aufbau erfolgte nach der Verteilungssicht (siehe Abschnitt 4.9) unter Einsatz von Docker-Images. Aus dem Build des Frontends bzw. Backends lässt sich jeweils ein lauffähiges Docker-Image erstellen.

QA-4: Die Funktionalität des Systems ist mithilfe einer HTTP-Schnittstelle verfügbar, die dem REST Level 2 nach dem Richardson Maturity Model entspricht (Ausnahme: Benutzeroberfläche).
Die HTTP-Schnittstelle des Backends erfüllt das Level 2 des Richardson Maturity Models.

QA-6: Das System ist gegen Fehleingaben abgesichert. Als Fehleingaben gelten hier Eingaben, die im Format oder hinsichtlich fachlicher Regeln abweichen.
Der Schutz gegen Attacken ist nicht Gegenstand dieser Qualitätsanforderung. Fehleingaben, die vom Nutzer ausgehen, werden erkannt und entsprechend behandelt. Die Prüfung und dazugehörige Behandlung erfolgt mehrstufig.
Beim Frontend werden Eingaben durch Formular-Validatoren überprüft. Wenn sie erfolgreich sind, werden die Daten im weiteren Ausführungsverlauf weitergereicht und als Objekt-Repräsentation nochmals überprüft. Das Backend führt eine automatische Überprüfung einer REST-Anfrage anhand von Constraints durch. Zudem erfolgt eine Überprüfung der im Ausführungsverlauf weitergereichten Daten (bzw. Objekte).
Ein Restrisiko, dass unerkannte Fehler in der Implementierung die Absicherung gegen Fehleingaben verhindern, kann nicht ausgeschlossen werden.
Die Qualitätsanforderung ist unter der Vernachlässigung des Restrisikos erfüllt.

QA-13: Das Logging erfolgt in einer konsistenten Art und Weise.
Das Logging ist im Backend und Frontend (voneinander unabhängig) konsistent nach dem folgend beschriebenem Schema.
Im Backend werden unerwartete technische Fehler mit Stacktrace in der Fehler-Stufe und wichtige fachliche Ereignisse in der Info-Stufe geloggt.
Beim Frontend führen unerwartete Fehler und Fehler, die auf REST-Rückgaben zurück-

gehen, zu Logs in der Fehler-Stufe. Wichtige fachliche Ereignisse werden in der Info-Stufe geloggt.

QA-14: Ursprünge von auftretenden Fehlern können durch Informationen von Logs eingegrenzt werden.
Die Qualitätsanforderung umfasst keine Logs von Fremdentwicklungen und keine fachlich erwartbaren Fehler, die absichtlich nicht geloggt werden.
Fehler-Logs im Backend enthalten eine Klasse, Beschreibung der Exception und des zugehörigen Stacktraces, der eine Eingrenzung ermöglicht. Beim Frontend umfassen Fehler-Logs die loggende Klasse, den Methodennamen und eine Fehlerbeschreibung. Die Eingrenzung ist insbesondere durch den Klassen- und Methodennamen möglich.

QA-15: Alle externen Schnittstellen, welche die fachliche Funktionalität bereitstellen, sind dokumentiert (Ausnahme: Schnittstellen für Benutzeroberflächen).
Die einzige Schnittstelle, die fachliche Funktionalität bereitstellt und keine Benutzeroberfläche ausliefert, ist die REST-Schnittstelle des Backends. Die Dokumentation erfolgt dafür mithilfe von Annotationen in RestController-Klassen, woraus durch Swagger UI eine Benutzeroberfläche erzeugt wird, die sich während der Backend-Ausführung abrufen lässt.

QA-16: Das System lässt sich mit ausführbaren Tests testen (mit getrennten Tests der Subsysteme).
Sowohl Tests fürs Backend als auch Frontend lassen sich erstellen und ausführen.

QA-17: Automatisierbare (im Rahmen dieser Arbeit entwickelte) Tests des Systems lassen sich automatisiert ausführen.
Als automatisierbare Tests gelten hier Tests, die nicht manuell ausgeführt werden.
Zugang zu entsprechender Software wie Gitlab und einer verfügbare Ausführungsumgebung vorausgesetzt, lassen sich sowohl die Tests des Backends als auch des Frontends in einer entsprechenden Pipeline, zum Beispiel nach Übertragung eines Commits, automatisiert ausführen.

Teilweise erfüllte Qualitätsanforderungen

QA-5: Das System weist den Nutzer auf Fehleingaben hin, die nicht dem erforderlichen Format oder nicht der fachlichen Eingabemenge entsprechen.

Der Endnutzer wird direkt bei der Eingabe in ein Formularfeld auf Fehler hingewiesen. Eine Ausnahme stellen Ankreuzfelder und Auswahl-Felder dar. Für sie wurde die Bibliothek Angular Material nicht genutzt, weil es darin keine Fehlerangabemöglichkeiten für Ankreuzfelder gibt bzw. weil ein Auswahlfeld mit dynamisch ermittelten Auswahlmöglichkeiten zu einer zyklischen Ausführungsschleife geführt hätte. Deshalb können für die beiden Felderarten keine Fehlerhinweise angezeigt werden.
Die für ein Formular zuständige Angular-Component reicht Formulardaten nicht für die Verarbeitung weiter, wenn die Formulareingaben ungültig sind, weil angenommen wird, dass auf Fehler automatisch beim Eingabefeld hingewiesen wird. Deshalb wird der Nutzer in diesen besonderen Fällen nicht auf eine Fehleingabe hingewiesen.
Von der Formular-Angular-Component als gültig befundene Eingaben werden weitergeleitet und anschließend erneut geprüft, wodurch Felder-übergreifende Eingabefehler erkannt und in einem Fehlerbanner signalisiert werden.
Das Backend überprüft Eingaben der REST-Schnittstelle ebenfalls und weist auf Fehleingaben mit entsprechenden Fehlercodes hin, zum Beispiel 400 (Bad Request).

Die Qualitätsanforderung des Hinweisens auf Fehleingaben ist somit mehrheitlich, aber nicht vollständig erfüllt, weil bei bestimmten Arten von Eingabefeldern kein Hinweis auf Fehler wegen Problemen mit Angular Material (bzw. darin fehlender Funktionalität) erfolgt.

QA-11: Daten eines Nutzers, die nicht für die Bearbeitung freigegeben wurden, lassen sich nicht von einem anderen Nutzer bearbeiten oder löschen.
Diese Qualitätsanforderung bezieht sich auf reguläre Funktionalität des Systems. Attacken, wie eine SQL-Injektion, sind ausgenommen.
Keine bisher umgesetzte funktionale Anforderung ermöglicht explizit das Bearbeiten oder Löschen von Daten. Wenn die Erstellung eines Terminangebotszeitraums als Änderung angebotener Terminzeiträume gilt, ist die Qualitätsanforderung nur teilweise erfüllt, weil aufgrund der fehlenden Anmeldung (US-017) zu Entwicklungszwecken angenommen wird, dass ein Beispielnutzer standardmäßig angemeldet sei.

QA-18: Build-Artefakte des Systems sind mit Umgebungsvariablen konfigurierbar.
Für das Backend ist das Build-Artefakt als JAR-Datei hinsichtlich der Datenbankverbindung mit Umgebungsvariablen konfigurierbar.
Das Build-Artefakt des Frontends kann nicht mit Umgebungsvariablen konfiguriert werden, weil es technologiebedingt aus Dateien für eine Webseite besteht, die in einem

Webserver eingesetzt und von diesem ausgeliefert werden. Es wäre zu aufwändig, den Webserver so zu modifizieren, dass Umgebungsvariablen ausgelesen und die Build-Artefakt-Dateien daran angepasst werden.

Nicht erfüllte Qualitätsanforderungen

QA-9: Daten von Nutzern, die ausschließlich für sie selbst bestimmt sind, lassen sich nicht von anderen Personen abrufen.
Die fehlende Umsetzung der Anmeldung eines Nutzers (US-017), die für einen Zugriffsschutz sorgt, verhindert die Erfüllung dieser Qualitätsanforderung.

QA-10: Ein nicht angemeldeter Nutzer kann nicht die Funktionen aufrufen, die ausschließlich für angemeldete Nutzer zugänglich sein sollen.
Da die Anmeldung (US-017) bisher nicht umgesetzt wurde und aus Entwicklungszwecken angenommen wird, ein Beispielnutzer sei bereits angemeldet, kann jede Person auf alle Funktionen zugreifen. Diese Qualitätsanforderung ist daher nicht erfüllt.

QA-12: Die Identität eines Nutzers kann durch Angabe eines Benutzernamens und Passworts bestätigt werden.
Diese Qualitätsanforderung ist nicht erfüllt, weil die Anmeldung eines Nutzers (US-017) nicht umgesetzt wurde.

Gesamtevaluation der Qualitätsanforderungen

Die Mehrheit der Qualitätsanforderungen wurde erfüllt. Insbesondere bezüglich der Sicherheit konnten Qualitätsanforderungen nicht erfüllt werden, weil die funktionale Anforderung für die Anmeldung eines Nutzers (US-017) nicht umgesetzt wurde.
Zudem konnten andere Qualitätsanforderungen, wie die Konfiguration mithilfe von Umgebungsvariablen (QA-18) oder das Hinweisen auf Fehleingaben (QA-5) unter Angabe von wichtigen Gründen nur teilweise erfüllt werden.

Die Erfüllung aller Qualitätsanforderungen, die nicht von fehlenden Umsetzungen funktionaler Anforderungen oder anderen Umständen unter Angabe von wichtigen Gründen betroffen sind, entspricht den Erwartungen.

6.4 Zusammenfassung

Im Rahmen der Evaluation wurde festgestellt, dass die Erwartungen an funktionale Anforderungen, Kernanforderungen für das Ziel der Vereinbarung von Terminen unter Berücksichtigung von Wünschen des Terminanbieters zu ermöglichen, erfüllt werden. Für einen produktiven Einsatz reichen die umgesetzten funktionalen Anforderungen nicht aus.
Das System hat alle Qualitätsattributsszenarien für die Laufzeit bestanden und erfüllt erwartungsgemäß alle Qualitätsanforderungen, die nicht durch eine fehlende Umsetzung erforderlicher funktionaler Anforderungen oder aus anderen wichtigen Gründen blockiert bzw. verhindert wurden.

Die Evaluation ist für das folgende Fazit der Arbeit relevant, um zu überprüfen, inwieweit das in dieser Arbeit entwickelte System die Zielsetzung erfüllt.

7 Fazit und Ausblick

Im Fazit wird die Zielstellung der Arbeit aufgegriffen, die Vorgehensweise innerhalb der Arbeit anhand vorangegangener Kapitel beschrieben und überprüft, inwiefern die Problemstellung bzw. Zielstellung gelöst wurde. Zudem wird über wichtige Erkenntnisse berichtet.
Der Ausblick zeigt zukünftige Verbesserungs- und Erweiterungsmöglichkeiten des in dieser Arbeit entwickelten Systems auf.

7.1 Fazit

Ziel der Arbeit war es, ein prototypisches System zu entwerfen und zu entwickeln, das es Personen ermöglicht, Einzeltermine anzubieten und zu buchen. Dafür waren die Termine und Wünsche des Terminanbieters bei der Vergabe zu berücksichtigen. Für die Bedienung sollte eine grafische Benutzeroberfläche vorhanden sein.

Zu Beginn der Arbeit wurde eine Anforderungsanalyse und Spezifikation durchgeführt. Darin wurde die Ausgangslage mittels Interviews und die aktuelle Situation bestehender Lösungen deutlich. Anhand der gewonnenen Informationen wurden Zielgruppen, funktionale Anforderungen, Anwendungsfälle, Qualitätsanforderungen mit zugehörigen Szenarien und ein fachliches Datenmodell spezifiziert. Es erfolgte die Auswahl der Systemkategorie als ein interaktives Online-System.
Im Entwurf wurde die grundlegende Architektur als monolithische, modular aufgeteilte 4-Schichten-Architektur festgelegt. Die Art der Benutzeroberfläche soll eine Webanwendung sein. Das System wurde in ein Frontend-Subsystem mit der Technologie Angular und in ein Backend-Subsystem mit dem Spring-Framework aufgeteilt. Die Kommunikation vom Frontend zum Backend erfolgt über eine REST-API (Level 2). Es wurde zudem der Systementwurf anhand unterschiedlicher Sichten beschrieben und dargestellt. Eine Auswahl von Abbildungsfunktionen für die Bewertung von Terminmöglichkeiten ist erfolgt.

Für die Implementierung wurden Highlights, wie die Benutzeroberfläche und die Strukturierung sowie die Wiederverwendung von Angular-Components für das Frontend und die Bewertung von Terminmöglichkeiten für das Backend, aufgezeigt. Herausforderungen der Implementierung bestanden in unterschiedlichen Repräsentationen von Referenzen in der Applikations- und Logikschicht und in der hohen Laufzeit eines Frontend-Tests.
Das implementierte System wurde anschließend bezüglich der spezifizierten funktionalen Anforderungen, Qualitätsanforderungen und der zugehörigen Qualitätsattributsszenarien evaluiert.

Die Problemstellung und Zielsetzung der Terminvermittlung, mit Einbeziehung von Vergabewünschen, wurde durch das System erfolgreich umgesetzt.
Es wurden mehr funktionale Anforderungen ermittelt, als im Rahmen dieser Arbeit umsetzbar waren. So wurden die für die Zielstellung relevantesten funktionalen Anforderungen umgesetzt, dazu gehört unter anderem die Erstellung von Terminen, Terminserien, von Terminangebotszeiträumen und die Ansicht sowie Buchung von Terminmöglichkeiten. Für einen produktiven Einsatz reicht das noch nicht aus.
Die Mehrheit der Qualitätsanforderungen und alle Qualitätsattributsszenarien konnten mit dem System erfüllt werden. Einige Qualitätsanforderungen wurden nicht vollständig erfüllt, unter anderem aufgrund teils noch nicht umgesetzter fachlicher Anforderungen und aufgrund anderer wichtiger Gründe, wie Einschränkungen durch eingesetzte Technologien.

Insgesamt traten beim Entwurf und der Entwicklung des prototypischen Systems wenig Schwierigkeiten auf. Es ließ sich die Erkenntnis gewinnen, dass die iterative Entwicklung vorteilhaft für die Arbeit war. Sie führte zu einer flexiblen Priorisierung von funktionalen Anforderungen und zu einer durchgehenden Funktionstüchtigkeit des Systems. Im Falle einer längeren oder kürzeren Umsetzungsdauer einer fachlichen Anforderung konnte die Planung umzusetzender funktionaler Anforderungen flexibel angepasst werden.
Darüber hinaus war die Festlegung von Qualitätsanforderungen anspruchsvoller als erwartet, weil diese für die Evaluierung überprüfbar sein müssen.

7.2 Ausblick

Das System setzt die wichtigsten funktionalen Anforderungen um. Dennoch sind für einen produktiven Einsatz Erweiterungen sinnvoll, welche die Anmeldung von Benutzern für

einen geschützten Zugriff und Funktionalitäten für Termine, wie die Einsicht und Absage von bestehenden Terminen, ermöglichen.

Zudem kann neben in den vorigen Abschnitten beschriebenen Verbesserungsmöglichkeiten eine Verfeinerung der Bewertungsheuristiken umgesetzt werden, damit die Kombination von mehreren Anbieterwünschen zu einer möglichst passenden Gesamtbewertung führt.

Eine Synchronisation von Terminen mit externen Terminverwaltungssystemen würde Terminanbietern die Arbeit erleichtern, weil die Termine dann nicht manuell übertragen werden müssten. Mögliche Benachrichtigungen per E-Mail ließen es zu, dass ein Anbieter schneller informiert werden würde und gebuchte Termine nicht aktiv abfragen müsste.

Darüber hinaus können zusätzliche Funktionalitäten, wie das Verfassen von Termin-Kommentaren oder die Definition und Bereitstellung von Zusatzangaben (z.B. Dateien oder Textangaben), die Vor- und Nachbereitung erleichtern.
Die beschriebenen Verbesserungs- und Erweiterungsmöglichkeiten können Anregungen für zukünftige Arbeiten und Projekte bieten.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

ACID Atomarität, Konsistenz, Integrität und Dauerhaftigkeit.

DOM Document Object Model.

DRY Don't repeat yourself.

DTO Data Transfer Object.

HATEOAS Hypertext As The Engine Of Application State.

HAW Hochschule für Angewandte Wissenschaften Hamburg.

ICC Informatik Compute Cloud der HAW.

JAR Java Archive.

JPA Java Persistence API.

JSON JavaScript Object Notation.

JVM Java Virtual Machine.

URI Uniform Resource Identifier.

Glossar

Abstract Factory Pattern Erzeugungsmuster, das eine Schnittstelle für die Erzeugung „verwandter oder voneinander abhängiger Objektfamilien ohne die Benennung ihrer konkreten Klassen“[32, S. 128] anbietet (vgl. [32, S. 128]).

Angular Entwicklungsplattform, die ein Webanwendungs-Framework für die Entwicklung von Single-Page Applications umfasst (vgl. [39] und [37]).

Angular Material Bibliothek für Angular, die Angular-Components im Material Design bietet (vgl. [40]).

Angular-Component Repräsentation des Begriffs „component“ des Angular-Frameworks. Nach dem Glossar von Angular beschreibe der Begriff eine Klasse, die teilweise eine Ansicht definiere (vgl. [33]). Diese stelle Daten bereit und verarbeite die Anzeige und Interaktion mit der Benutzeroberfläche (vgl. [33]).

Angular-Router Ein Angular-Router ermöglicht nach dem Angular-Glossar die Navigation mit Zuständen und Sichten innerhalb einer einzigen Seite und in Verbindung mit einer URL (vgl. [34]).

Angular-Service Klasse in Angular, die Logik, die nicht zur Benutzeroberfläche gehört, sowie wiederverwendbaren Code enthält (vgl. [35]). Sie kann in Verbindung mit Dependency Injection genutzt werden (vgl. [35]).

CalDAV Erweiterung des WebDAV-Protokolls, um unter anderem auf Informationen für Kalender und Termine zuzugreifen und um diese verwalten und teilen zu können (vgl. [12]).

Data Binding „Automatische Weitergabe von Daten zwischen Objekten“[57].

Data Transfer Object Ein Data Transfer Object aggregiert Daten von Objekten und wird für Anfragen bei Remote-Schnittstellen als einziger Parmeter oder als Rückgabe eingesetzt (vgl. [31, S. 401]).

Dependency Injection Vorgehensweise, bei der eine Abhängigkeit einer Klasse, die in der Klasse genutzt wird, von einem anderen Objekt als konkrete Implementierung eingesetzt wird (vgl. [26]).

Document Object Model Schnittstelle, die im Zusammenhang mit HTML- und XML-Dokumenten eingesetzt wird (vgl. [52]). „Sie bildet die strukturelle Abbildung des Dokuments und ermöglicht Skripten die Veränderung des Inhalts und dessen Präsentation“[52].

Eventual Consistency Spezielle Form der schwachen Konsistenz mit der Garantie, dass nachfolgende Zugriffe auf ein Datum, ohne dass es kürzlich verändert wurde, letztendlich den neuesten Wert erhalten würden (vgl. [69, S. 42]).

Facade Pattern Strukturmuster, das der „Bereitstellung einer einheitlichen Schnittstelle zu einem Schnittstellensatz in einem Subsystem“[32, S. 237] dient (vgl. [32, S. 237]). Es „definiert eine Schnittstelle höherer Ebene, die die Nutzung des Subsystems vereinfacht“[32, S. 237].

Hibernate Technologie, die unter anderem Object-Relational Mapping und Validierung ermöglicht (vgl. [47]).

Inversion of Control Umkehrung des Kontrollflusses, sodass der eigene Anwendungscode (zum Beispiel von einem Framework) aufgerufen wird und sodass dieser Anwendungscode damit die Kontrolle der Ausführung abgibt (vgl. [27]).

Java Persistence API „Java API für die Verwaltung von Persistenz“[15](übersetzt) und Object-Relational Mapping (vgl. [15]).

Kubernetes Plattform für die „Verwaltung von containerisierten Arbeitslasten und Services“[62] (vgl. [62]).

Kubernetes-Pod Gruppe innerhalb von Kubernetes, die Container und eine zugehörige Ausführungsspezifikation umfasst (vgl. [61]).

Microservice-Architektur Architekturstil einer in kleine Services aufgeteilten Anwendung, deren Services jeweils Funktionalität zu einer Geschäftsfähigkeit bereitstellen (vgl. [30]).

Object-Relational Mapping „Programmiertechnik zur Konvertierung von Daten zwischen relationalen Datenbanken und objektorientierten Programmiersprachen“[13], mit der „Objekte auf die Strukturen einer relationalen Datenbank abgebildet werden“[13] können (vgl. [13]).

Observer Pattern Verhaltensmuster, das nach einer Zustandsänderung zu einer Benachrichtigung und Aktualisierung abhängiger Objekte führt (vgl. [32]).

Qualitätsattributsszenario Anforderung für ein Qualitätsmerkmal, die aus sechs Teilen besteht (vgl. [5, S. 75]). Anmerkung: Die Durch- bzw. Ausführung eines Qualitätsattributsszenarios steht in der Arbeit für die Überprüfung dieser Anforderung anhand eines darin beschriebenen Vorgangs.

React JavaScript-Bibliothek, mit der Benutzeroberflächen entwickelt werden können (vgl. [21]).

Repository Pattern Ein Repository kapselt die persistente Datenhaltung von Objekten und das Zusammensetzen dafür eingesetzter Operationen, um eine klare Trennung der Domänenschicht von Datenabbildungsschichten zu erreichen (vgl. [31, S. 322-323]). Es nutzt eine „sammlungsähnliche Schnittstelle, um auf Domänenobjekte zuzugreifen“[31, S. 322](übersetzt).

Richardson Maturity Model Das Richardson Maturity Model beschreibt die grundlegenden Eigenschaften von REST-APIs und teilt diese in 3 Level auf (vgl. [28]).

Single-Page Application Webanwendung, die mithilfe von JavaScript betrieben wird und nur einmalig als Seite geladen werden muss (vgl. [25, S. 497]).

SMTP Protokoll für den Transport und die Zustellung von Mails (vgl. [50, S. 0 (Abstract)]).

Spring Framework für die Programmiersprache Java, das flexibel erweiterbar sei und unter anderem Inversion of Control und Dependency Injection bietet (vgl. [64]).

Strategy Pattern Verhaltensmuster, das eine „Familie von einzeln gekapselten, austauschbaren Algorithmen“[32, S. 383] definiert und eine variable Nutzung der Algorithmen ermöglicht (vgl. [32, S. 383]).

Swagger UI Werkzeug für eine interaktive und visuelle API-Dokumentation (vgl. [58]).

Uniform Resource Identifier „Kompakte Zeichenkette, die eine abstrakte oder physikalische Ressource identifiziert“[6, S. 0 (Abstract), übersetzt].

Wireframe „Hilfsmittel für die Konzeption von Websites“[51] mit einer reduzierten Darstellungsweise (vgl. [51]).

Literaturverzeichnis

[1] Apple Inc.: *Antworten auf Einladungen in der App „Kalender“ auf dem Mac - Apple Support*. Webseite. – URL https://support.apple.com/de-de/guide/calendar/icl1019/11.0/mac/10.15. – Zugriffsdatum: 14.4.2021

[2] Apple Inc.: *Einladen von Personen zu Ereignissen in der App „Kalender“ auf dem Mac - Apple Support*. Webseite. – URL https://support.apple.com/de-de/guide/calendar/icl1016/11.0/mac/10.15. – Zugriffsdatum: 14.4.2021

[3] Apple Inc.: *Festlegen von Benachrichtigungen für Ereignisse und Empfangen von Benachrichtigungen in der App „Kalender“ auf dem Mac - Apple Support*. Webseite. – URL https://support.apple.com/de-de/guide/calendar/icl1012/11.0/mac/10.15. – Zugriffsdatum: 14.4.2021

[4] Apple Inc.: *Hinzufügen, Ändern oder Löschen von Ereignissen auf dem Mac - Apple Support*. Webseite. – URL https://support.apple.com/de-de/guide/calendar/icalwr13-events/11.0/mac/10.15. – Zugriffsdatum: 14.4.2021

[5] Bass, Len ; Kazman, Rick ; Clements, Paul: *Software Architecture in practice*. 2. ed. Addison-Wesley Professional, 2003 (SEI series in software engineering). – ISBN 0321154959

[6] Berners-Lee, T. ; Fielding, R. ; Masinter, L.: *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. Januar 2005. – URL https://www.rfc-editor.org/rfc/rfc3986.txt. – Zugriffsdatum: 21.5.2021

[7] Calendly: *Product Features - Calendly*. Webseite. – URL https://calendly.com/de/features. – Zugriffsdatum: 15.4.2021

[8] Braincept AG: *Einzeltermine für deine Kunden einfach online buchbar machen | calenso*. Webseite. – URL https://calenso.com/einzelbuchung/. – Zugriffsdatum: 8.6.2021

[9] Clockwise Inc.: *Calendar Management | Schedule Optimization | Clockwise*. Webseite. – URL https://www.getclockwise.com/autopilot. – Zugriffsdatum: 15.4.2021

[10] Clockwise Inc.: *Scheduling Assistant | Calendar Management | Clockwise*. Webseite. – URL https://www.getclockwise.com/overview. – Zugriffsdatum: 14.4.2021

[11] Cohn, Mike: *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional, März 2004. – ISBN 0321205685

[12] Daboo, Cyrus ; Dusseault, Lisa M. ; Desruisseaux, Bernard: *Calendaring Extensions to WebDAV (CalDAV)*. RFC 4791. März 2007. – URL https://rfc-editor.org/rfc/rfc4791.txt. – Zugriffsdatum: 14.6.2021

[13] DATACOM Buchverlag GmbH: *ORM (object relational mapping) / ITWissen.info*. Webseite. – URL https://www.itwissen.info/ORM-object-relational-mapping.html. – Zugriffsdatum: 14.6.2021

[14] Doodle AG: *How do I set up my Bookable Calendar? - Doodle*. Webseite. Januar 2021. – URL https://help.doodle.com/hc/en-us/articles/360037441034. – Zugriffsdatum: 14.4.2021

[15] Eclipse Foundation: *Eclipse Announces EclipseLink Project to Deliver JPA 2.0 Reference Implementation | The Eclipse Foundation*. Presseinformation. März 2008. – URL http://www.eclipse.org/org/press-release/20080317_Eclipselink.php. – Zugriffsdatum: 14.6.2021

[16] Eigner, Thomas: *Web-Anwendungen vs Desktop-Anwendungen | Der Vergleich*. Webseite. März 2019. – URL https://www.cyber-solutions.at/blog/web-vs-desktop. – Zugriffsdatum: 26.5.2021

[17] Evans, Eric: *Domain-Driven Design: Tackling Complexity in the Heart of Software*. 4. print. Addison-Wesley, 2004

[18] Facebook Inc.: *AJAX and APIs - React*. Webseite. – URL https://reactjs.org/docs/faq-ajax.html. – Zugriffsdatum: 22.4.2021

[19] Facebook Inc.: *Design Principles – React*. Webseite. – URL https://reactjs.org/docs/design-principles.html. – Zugriffsdatum: 3.6.2021

[20] Facebook Inc.: *Glossary of React Terms - React*. Webseite. – URL https://reactjs.org/docs/glossary.html#single-page-application. – Zugriffsdatum: 22.4.2021

[21] Facebook Inc.: *React - A JavaScript library for building user interfaces*. Webseite. – URL https://reactjs.org. – Zugriffsdatum: 22.4.2021

[22] Facebook Inc.: *Rendering Elements - React*. Webseite. – URL https://reactjs.org/docs/rendering-elements.html. – Zugriffsdatum: 23.4.2021

[23] Facebook Inc.: *Testing Overview - React*. Webseite. – URL https://reactjs.org/docs/testing.html. – Zugriffsdatum: 22.4.2021

[24] Facebook Inc.: *Thinking in React - React*. Webseite. – URL https://reactjs.org/docs/thinking-in-react.html. – Zugriffsdatum: 22.4.2021

[25] Flanagan, David: *JavaScript: The Definitive Guide: The Definitive Guide*. 5. O'Reilly Media, 2006. – ISBN 9780596554477

[26] Fowler, Martin: *Inversion of Control Containers and the Dependency Injection pattern*. Webseite. Januar 2004. – URL https://martinfowler.com/articles/injection.html. – Zugriffsdatum: 21.4.2021

[27] Fowler, Martin: *InversionOfControl*. Webseite. Juni 2005. – URL https://martinfowler.com/bliki/InversionOfControl.html. – Zugriffsdatum: 21.4.2021

[28] Fowler, Martin: *Richardson Maturity Model*. Webseite. März 2010. – URL https://martinfowler.com/articles/richardsonMaturityModel.html. – Zugriffsdatum: 3.5.2021

[29] Fowler, Martin: *MonolithFirst*. Webseite. Juni 2015. – URL https://martinfowler.com/bliki/MonolithFirst.html. – Zugriffsdatum: 10.5.2021

[30] Fowler, Martin ; Lewis, James: *Microservices*. Webseite. März 2014. – URL https://martinfowler.com/articles/microservices.html. – Zugriffsdatum: 10.5.2021

[31] Fowler, Martin ; Rice, David ; Foemmel, Matthew ; Hieatt, Edward ; Mee, Robert ; Stafford, Randy: *Patterns of enterprise application architecture*. 8. print. Addison-Wesley, 2005 (The Addison-Wesley signature series). – ISBN 0321127420

[32] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John M.: *Design Patterns: Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software*. 1. Aufl. mitp, 2015. – ISBN 9783826697005

[33] Google: *Angular - Glossary*. Webseite. – URL https://angular.io/guide/glossary#component. – Zugriffsdatum: 23.4.2021

[34] Google: *Angular - Glossary*. Webseite. – URL https://angular.io/guide/glossary#router. – Zugriffsdatum: 23.4.2021

[35] Google: *Angular - Glossary*. Webseite. – URL https://angular.io/guide/glossary#service. – Zugriffsdatum: 8.6.2021

[36] Google: *Angular - In-app navigation: routing to views*. Webseite. – URL https://angular.io/guide/router. – Zugriffsdatum: 22.4.2021

[37] Google: *Angular - Introduction to Angular concepts*. Webseite. – URL https://angular.io/guide/architecture. – Zugriffsdatum: 22.4.2021

[38] Google: *Angular - Two-way binding*. Webseite. – URL https://angular.io/guide/two-way-binding. – Zugriffsdatum: 3.6.2021

[39] Google: *Angular - What is Angular?* Webseite. – URL https://angular.io/guide/what-is-angular. – Zugriffsdatum: 22.4.2021

[40] Google: *Angular Material UI component library*. Webseite. – URL https://material.angular.io. – Zugriffsdatum: 10.5.2021

[41] Google: *Gäste zu Google Kalender-Terminen einladen - Computer - Google Kalender-Hilfe*. Webseite. – URL https://support.google.com/calendar/answer/37161?hl=de&ref_topic=3417926. – Zugriffsdatum: 15.4.2021

[42] Google: *Google Kalender-Benachrichtigungen ändern oder deaktivieren - Computer - Google Kalender-Hilfe*. Webseite. – URL https://support.google.com/calendar/answer/37242. – Zugriffsdatum: 14.4.2021

[43] Google: *Termine erstellen - Computer - Google Kalender-Hilfe*. Webseite. – URL https://support.google.com/calendar/answer/72143?hl=de&ref_topic=6270042. – Zugriffsdatum: 14.4.2021

[44] Google: *Terminen Anhänge hinzufügen - Computer - Google Kalender-Hilfe*. Webseite. – URL https://support.google.com/calendar/answer/6192039?hl=de&ref_topic=6270042. – Zugriffsdatum: 14.4.2021

[45] H2: *Features (H2 Database Engine)*. Webseite. – URL https://www.h2database.com/html/features.html. – Zugriffsdatum: 21.4.2021

[46] Haywood, Dan: *In Defence of the Monolith, Part 1*. Webseite. März 2017. – URL https://www.infoq.com/articles/monolith-defense-part-1/. – Zugriffsdatum: 8.6.2021

[47] Hibernate: *Hibernate. Everything data. - Hibernate*. Webseite. – URL https://hibernate.org. – Zugriffsdatum: 4.5.2021

[48] Hrudschka, Peter ; Starke, Gernot: *Template - acr42*. Webseite. – URL https://arc42.de/template. – Zugriffsdatum: 18.5.2021

[49] iso25000.com: *ISO 25010*. Webseite. – URL https://iso25000.com/index.php/en/iso-25000-standards/iso-25010. – Zugriffsdatum: 8.6.2021

[50] Klensin, John C.: *Simple Mail Transfer Protocol*. RFC 5321. Oktober 2008. – URL https://rfc-editor.org/rfc/rfc5321.txt. – Zugriffsdatum: 14.6.2021

[51] kulturbanause®: *Was ist ein Wireframe? | kulturbanause®*. Webseite. – URL https://kulturbanause.de/faq/wireframe/. – Zugriffsdatum: 7.5.2021

[52] Mozilla and individual contributors: *DOM - Web API Referenz | MDN*. Webseite. – URL https://developer.mozilla.org/de/docs/Web/API/Document_Object_Model. – Zugriffsdatum: 23.4.2021

[53] Parlakov, Georgi: *The Page Pattern: Separating the business and UI logic problem in Angular*. Webseite. September 2020. – URL https://medium.com/angular-in-depth/the-page-pattern-9f437ec99d7b. – Zugriffsdatum: 27.4.2021

[54] Pietrucha, Bartosz: *Angular Architecture Patterns and Best Practices (that help to scale)*. Webseite. Juli 2019. – URL https://dev-academy.com/angular-architecture-best-practices/. – Zugriffsdatum: 27.4.2021

[55] Pivotal, Inc.: *Testing*. Webseite. – URL https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/testing.html. – Zugriffsdatum: 21.4.2021

[56] Rölle, Chantal ; Serwitzki, Fabian ; Studienkreis GmbH: *Übersicht: Funktionstypen und ihre Eigenschaften*. Webseite. – URL https://www.studienkreis.de/mathematik/funktionstypen-uebersicht/. – Zugriffsdatum: 12.5.2021

[57] Schwichtenberg, Holger: *Data Binding - Begriffserklärung im Entwickler-Lexikon/Glossar auf www.IT-Visions.de*. Webseite. – URL https://www.it-visions.de/glossar/alle/6875/Data_Binding.aspx. – Zugriffsdatum: 21.5.2021

[58] SmartBear Software: *REST API Documentation Tool | Swagger UI*. Webseite. – URL https://swagger.io/tools/swagger-ui/. – Zugriffsdatum: 3.5.2021

[59] Sommerville, Ian: *Modernes Software-Engineering: Entwurf und Entwicklung von Softwareprodukten*. Pearson Studium, 2020. – ISBN 9783868943962

[60] Starke, Gernot: *Effektive Softwarearchitekturen: Ein praktischer Leitfaden*. 9. Aufl. Carl Hanser Verlag GmbH & Co KG, August 2020. – ISBN 9783446466906

[61] The Kubernetes Authors ; The Linux Foundation ®: *Pods | Kubernetes*. Webseite. – URL https://kubernetes.io/de/docs/concepts/workloads/pods/. – Zugriffsdatum: 21.5.2021

[62] The Kubernetes Authors ; The Linux Foundation ®: *Was ist Kubernetes? | Kubernetes*. Webseite. – URL https://kubernetes.io/de/docs/concepts/overview/what-is-kubernetes/. – Zugriffsdatum: 21.5.2021

[63] The PostgreSQL Global Development Group: *PostgreSQL: About*. Webseite. – URL https://www.postgresql.org/about/. – Zugriffsdatum: 21.4.2021

[64] VMware, Inc.: *Spring | Why Spring?* Webseite. – URL https://spring.io/why-spring. – Zugriffsdatum: 21.4.2021

[65] VMware, Inc.: *Spring Boot*. Webseite. – URL https://spring.io/projects/spring-boot. – Zugriffsdatum: 21.4.2021

[66] VMware, Inc.: *Spring Data*. Webseite. – URL https://spring.io/projects/spring-data. – Zugriffsdatum: 21.4.2021

[67] VMware, Inc.: *Spring Data JPA*. Webseite. – URL https://spring.io/projects/spring-data-jpa. – Zugriffsdatum: 12.4.2021

[68] VMware, Inc.: *Spring Security*. Webseite. – URL https://spring.io/projects/spring-security. – Zugriffsdatum: 21.4.2021

[69] Vogels, Werner: Eventually Consistent. In: *Commun. ACM* 52 (2009), Januar, Nr. 1, S. 40–44. – URL https://dl.acm.org/doi/pdf/10.1145/1435417.1435432. – Zugriffsdatum: 13.6.2021. – ISSN 0001-0782

[70] YouCanBook.me Ltd: *Control and manage your booking flow - YouCanBook.me.* Webseite. – URL https://youcanbook.me/feature/booking-flow/. – Zugriffsdatum: 14.4.2021

A Anhang

Der Quellcode des Frontends und Backends inklusive Dokumentation für die Ausführung ist auf der CD zu finden.

A.1 Zusammenfassung der Interviews

Die Interviews erfolgten mit zwei Professoren und drei Studenten.

Ausgangslage

Art der Verwaltung von Terminen und genutzten Technologien

- digital: Apple Kalender, Google Kalender, Outlook
- manuell: Kalender-Notizbuch mit flexiblen Markierungsmöglichkeiten

Art des Anbietens von Terminen und genutzten Technologien (als Terminanbieter)
Per E-Mail, mündlich oder für mehrere Personen: Doodle (mit Werbung) oder Deutsches Forschungsnetz (ohne Werbung)

Ablauf des Anbietens von Terminen bei eingegangener Terminanfrage

1. Ermittlung der Terminart und des Anliegens
2. Zeitpunkt finden: Im Kalender eine Lücke finden, die möglichst wenig neue Lücken entstehen lässt, Zeitwünsche von Personen berücksichtigen, mindestens eine Terminmöglichkeit anbieten und ggf. zusätzlichen Aufwand berücksichtigen
3. Anmerken und Nachfordern von notwendigen Informationen

Art des Vereinbarens von Terminen und genutzten Technologien (als Terminnehmer)
Überwiegend per E-Mail im Kontext der HAW, wenn vorhanden Online-Dienste (z.B. Doodle bei Gruppen), persönlich, telefonisch

Ablauf des Vereinbarens von Terminen als Terminnehmer
Annahme: Vereinbarung per E-Mail:

1. Kontakt per E-Mail mit Anliegen aufnehmen, ggf. Wünsche
2. Abstimmung per E-Mail mit dem Termingeber für die Vereinbarung

Subjektive Vor- und Nachteile der Art und Weise der genutzten Terminvereinbarung
Für Terminnehmer (in Bezug auf E-Mail-Vereinbarung, nur Nachteile genannt):
Überwiegender Nachteil: große Dauer/Verzögerung der E-Mail-Kommunikation
Nachteile: Hoher Aufwand durch viele Nachrichten, schlechte Erreichbarkeit des Terminanbieters, Unübersichtlichkeit durch Nutzung mehrerer Programme
Für Terminanbieter:
Überwiegender Nachteil: hoher Aufwand durch Schreiben vieler E-Mails, auch Nachforderung von Informationen
Nachteile: Eigene Zeitslotverwaltung inklusive der Verwaltung von Terminvorschlägen erforderlich, schlechte Wiederverwendbarkeit angebotener, anschließend abgelehnter Terminvorschläge
Vorteile: Flexibilität des Anbietens von Terminen bei Prioritätsfällen, eigene Kontrolle und Unabhängigkeit von einem Terminvereinbarungssystem

Art der Erinnerung an Termine und Hilfsmittel
Mit Post-Its, eigenem Erinnerungsvermögen, oder eine Erinnerungsfunktion eines Terminverwaltungssystems

Termine

Termineigenschaften
Überwiegend: Startzeitpunkt, Dauer/Ende, Ort/Medium, Anliegen/Agenda, Teilnehmer mit Kontaktdaten
Vereinzelt: Anhänge/Dokumente, Diskussion mit Fragen/Antworten für benötigte Informationen, Kategoriemarkierung, private/interne Notizen des Terminanbieters

Granularität für Terminzeitpunkte und der Dauer
Überwiegend 5-minütlich, 15-minütlich
Vereinzelt: 10-minütlich oder minütlich (falls ein bereits festgelegter Termin auf die Minute genau besteht)

Art der Nutzung von terminbegleitenden Materialien
Falls erforderlich, als Datei, interner Terminkommentar oder physisches Material
Vereinzelt: meist im Voraus, seltener im Nachhinein

Gemeinsamkeiten von Terminen und mögliche Gruppierung aus Terminanbietersicht
Art der Termine: Zum Beispiel für Bafög, Hochschulwechsel, Studienfachberatung oder Themenfindung von Bachelor-/Masterarbeiten
Dauer der Termine: Beispielsweise oftmals eine Stunde lang

Auftretende Terminänderungen
Überwiegend: Absagen (beidseitig), Zeitpunktverschiebungen
Vereinzelt: Änderung des Themas und der Teilnehmeranzahl

Art der Terminabsage
Über dieselbe Art, wie die Vereinbarung stattfand oder per E-Mail, inhaltlich mit einem Absagegrund und ggf. Vorschlag einer Alternative

Möglichkeiten der Terminübertragung ins selbst genutzte Terminverwaltungssystem
Abschreiben (physisch oder digital), Import als Kalendereintragsdatei, CalDAV-Anbindung, automatische Extraktion aus einer E-Mail mittels eines E-Mail-Programms

Terminvergabe (nur für Terminanbieter relevant)

Wichtige Kriterien für die Auswahl eines Terminzeitpunkts (abgesehen von bestehenden Terminen)
Nähe zu bestehenden Terminen, Vermeidung von Lücken/Fragmenten, Füllung von vorne/hinten, Freihaltung von Zeitblöcken (z.B. Pausen), Vermeidung von Extrema (z.B. frühmorgens), Berücksichtigung zu bevorzugender/meidender Zeitpunkte/Zeiträume, Begrenzung der Anzahl von Terminarten pro Slot nach Art

Inwiefern werden zeitliche Wünsche eines Terminnehmers berücksichtigt? Überwiegende Berücksichtigung mit Ausnahmen, Berücksichtigung von zu vermeidenden Zeiten, mit Begründung, warum eine Vermeidung erwünscht wird

Erwartete Informationen über einen Terminteilnehmer Abhängig von der Terminart, Angaben zur Person, E-Mail, eher keine Telefonnummer

Wie oft werden Termine innerhalb einer Woche vergeben? Unterschiedlich, teils 3-4 oder 0-30 Mal pro Woche, Abhängigkeit von der Zeit (Anfang/Mitte/Ende des Semesters)

Wie spontan werden Termine vergeben? Was ist die Grenze? Kurzfristig für dringende Fälle, folgende Woche für weniger dringende Fälle, mündlich ggf. am selben Tag, per E-Mail spätestens ein Tag vorher

Arten von Personen, mit denen Termine vereinbart werden Studenten (überwiegend), Studieninteressenten als externe Personen, und sehr selten mit internen Mitarbeitern

Terminannahme

Art der Suche nach Termine anbietenden Personen (in Bezug zum Kontext der HAW)
Personensuche auf der HAW-Webseite oder mit einer Suchmaschine (nach Name, Titel, Fachbereich), Einsicht der Personen-Homepage und Verlinkung auf Buchungsmöglichkeiten

Welche preiszugebenden bzw. anzugebenden Daten werden für eine Buchung bei einer terminanbietenden Person erwartet? Vorname, Nachname, E-Mail, Anliegen, ungern Telefon
Abhängig von Terminart/Rolle: Matrikelnummer, Studiengang, Semester, Geburtsdatum, anonym, passender Zeitraum/spezifische Uhrzeit

Inwiefern ist eine Registrierung für einen Account für die Buchung eines Termins erwünscht? Überwiegend: Keine Registrierung erwünscht
Vereinzelt: nur mit Wiederverwendung des HAW-Accounts, keine Verwendung von Angaben zur Terminbuchung für eine Registrierung

Wünsche für die zeitliche Vergabe eines Termins als Terminnehmer Erforderlich: Angabe eines Datums, Eingrenzung des Zeitraums, an dem man (nicht) kann
Optional: Zeitpunktangabe, nächstgelegener Termin, Nachfragen für individuelle Termine, Vermeidung extremer Uhrzeiten, Angabe mehrerer Tage/Uhrzeiten

Nachträgliche Zugriffs- und Verwaltungswünsche eines Terminnehmers für einen Termin
Überwiegend: Absage
Vereinzelt: per E-Mail-Link, Zugriffsmöglichkeit mit Verhinderung von Missbrauch durch Dritte, Einsicht angegebener Daten, kurzfristige Änderungen eigener Angaben, Verschiebung, Hinterlegen weiterer Informationen

Benachrichtigungen

Erwünschte Benachrichtigungszwecke
Buchungsbestätigung, Änderung, Absage, Erinnerung

Erwünschte Kanäle für Benachrichtigungen E-Mail (überwiegend), Messenger, Push-Benachrichtigung via Smartphone-App

Account (Befragung nur von internen Mitarbeitern als Terminanbieter)

Arten der Anmeldung
Verschiedene Ansichten: Benutzername und Passwort, hausintern mit interner E-Mail-Adresse, LDAP der HAW, ohne Erforderlichkeit eines neuen Passworts

Welche Informationen dürfen fremde Personen über Sie sehen, wenn sie nach Ihnen suchen?
Kontaktdaten, freie Terminoptionen, keine bestehenden Termine, Informationen über die Art des Treffens (z.B. Zoom)

Sonstiges

Welche sonstigen Aspekte sind Ihnen für die Terminverwaltung und -vergabe wichtig?
Übersicht kommender Termine, Diskussionen für Termine, Checkliste für Angaben, eingefärbte Kalenderansicht für die Buchung, explizite Bestätigung einer Buchung vom Anbieter, Absage aller Termine innerhalb eines Zeitraums, Angabe einer alternativen Kontaktmöglichkeit bei unpassenden Terminen
Bedienung: Übersichtlichkeit, Einfachheit der Bedienung, Verständlichkeit
Protokollierung: Protokollierung fachlicher Tätigkeiten, um diese später beweisen zu können, persistente Speicherung in einer Datei oder Datenbank
Betrieb: verschlüsselte Übertragung im Webbrowser, Speicherung des Passworts als Hash, nicht zu viel Konfiguration, leichte Betreibbarkeit und Möglichkeit eines Backups bzw. der Wiederherstellung
Schutz vor Missbrauch: Nicht zu geringer Aufwand pro Buchung, keine Vergabe mehrerer nahestehender Termine an die gleiche Person (freischaltbare Ausnahmen)

A.2 Funktionale Anforderungen (User Stories)

Administration

US-001 Als Administrator kann ich als Artefakte konfigurierbare Docker-Images erzeugen, um das System betreiben zu können.

US-002 Als Administrator kann ich Systemereignisse und Fehlermeldungen persistent protokollieren lassen, um getätigte Aktionen im Zweifel nachweisen zu können (bezieht sich nur auf Systemteile, die fachliche Daten verwalten und auch fachliche Funktionalität bieten).

Account

US-016 Als interner Mitarbeiter kann ich mich für einen Account registrieren, um zukünftig Termine für andere Personen anbieten zu können.

- Als interner Mitarbeiter kann ich meine Accountinformationen einsehen, um diese zu überprüfen.
- Als interner Mitarbeiter kann ich Zugangsdaten meines Accounts ersetzen, um bei Vergessen den Account weiterhin nutzen zu können oder Missbrauch zu verhindern.
- Als interner Mitarbeiter kann ich meinen Account löschen, um alle meine Daten zu löschen.

US-017 Als interner Mitarbeiter kann ich mich anmelden, um meinen Account zu nutzen.

US-018 Als interner Mitarbeiter kann ich mich abmelden, um meinen Account nicht mehr zu nutzen.

- Als Student oder externe Person oder interner Mitarbeiter kann ich mithilfe eines Links direkt die Accountübersicht einer Person aufrufen, um über eine externe Verlinkung direkt auf Informationen eines Accounts, insbesondere Terminangebote, zugreifen zu können.
- Als Student oder externe Person kann ich nach dem Account einer Person nach Accountname suchen, um eine spezielle Person zu finden.
- Als Student oder externe Person kann ich Accountinformationen einer Person einsehen, um mich über diese zu informieren.

Einzeltermin

US-004 Als Student oder interner Mitarbeiter oder externe Person kann ich Informationen eines Einzeltermins einsehen, um mich auf den Einzeltermin vorbereiten zu können.

- Als interner Mitarbeiter kann ich einen Einzeltermin löschen, um nicht mehr benötigte Einzeltermine dauerhaft zu entfernen.

US-019 Als Student oder interner Mitarbeiter oder externe Person kann ich einen Einzeltermin absagen, um zu signalisieren, dass der Einzeltermin nicht mehr benötigt wird oder nicht wahrgenommen werden kann.

- Als interner Mitarbeiter kann ich alle Einzeltermine innerhalb eines angegebenen Zeitraums absagen, um bei Massenabsagen Zeit zu sparen.

US-003 Als interner Mitarbeiter kann ich einen Einzeltermin erstellen, damit außerhalb des Systems bestehende Termine auch berücksichtigt werden.

- Als Student oder externe Person kann ich an eine Person eine direkte Einzelterminanfrage stellen, um einen Einzeltermin individuell zu vereinbaren.
- Als interner Mitarbeiter kann ich meine Einzeltermine nach Name suchen, um mich über einen speziellen Einzeltermin zu informieren.
- Als interner Mitarbeiter kann ich meine Einzeltermine nach Datum/Zeit suchen, um mich über einen speziellen Einzeltermin zu informieren.
- Als interner Mitarbeiter kann ich meine Einzeltermine nach einem Teilnehmer suchen, um mich über einen speziellen Einzeltermin zu informieren.
- Als interner Mitarbeiter kann ich kommende Einzeltermine einsehen, um mich über nahende Termine zu informieren.
- Als interner Mitarbeiter kann ich Einzeltermine in einer Tagesansicht einsehen, um mich an Einzeltermine desselben Tages zu erinnern oder um den Zeitplan eines speziellen Tages nachvollziehen zu können.

US-005 Als interner Mitarbeiter kann ich Einzeltermine in einer Wochenansicht einsehen, um den Zeitplan der aktuellen/einer speziellen Woche nachvollziehen zu können.

- Als interner Mitarbeiter kann ich Einzeltermine in einer Monatsansicht einsehen, um den Zeitplan des aktuellen/eines speziellen Monats nachvollziehen zu können.
- Als Student oder interner Mitarbeiter oder externe Person kann ich Einzelterminformationen ändern, um fehlerhafte Informationen zu korrigieren.

US-020 Als Student oder externe Person kann ich mithilfe eines Links auf einen von mir gebuchten Einzeltermin zugreifen, um ihn einsehen, ändern oder absagen zu können.

Kommentar

- Als Student oder externe Person oder interner Mitarbeiter kann ich einen Kommentar für einen Einzeltermin erstellen, um erforderliche Informationen anzugeben.
- Als Student oder externe Person oder interner Mitarbeiter kann ich einen eigenen bestehenden Kommentar zu einem Einzeltermin löschen, um fehlerhafte Informationen zu entfernen.
- Als Student oder externe Person oder interner Mitarbeiter kann ich einen für mich bestimmten Kommentar eines Einzeltermins einsehen, um mich zu informieren.

Angaben

- Als interner Mitarbeiter kann ich eine Angabenanforderung für einen Einzeltermin erstellen, um einen Teilnehmer an benötigte Informationen für den Termin zu erinnern.
- Als Student oder externe Person oder interner Mitarbeiter kann ich alle Angabenanforderungen eines Einzeltermins einsehen, um mich über benötigte Angaben zu informieren.
- Als interner Mitarbeiter kann ich eine Angabenanforderung eines Einzeltermins ändern, um diese zu korrigieren.
- Als interner Mitarbeiter kann ich eine Angabenanforderung eines Einzeltermins löschen, um auf eine nicht mehr benötigte Anforderung zu reagieren.

- Als Student oder externe Person oder interner Mitarbeiter kann ich eine einfache Wertangabe für einen Einzeltermin erstellen, um benötigte Informationen zu hinterlegen.
- Als Student oder externe Person oder interner Mitarbeiter kann ich eine einfache Wertangabe für einen Einzeltermin ändern, um diese zu korrigieren.
- Als Student oder externe Person oder interner Mitarbeiter kann ich eine einfache Wertangabe für einen Einzeltermin einsehen, um zusätzliche Informationen für den Termin einzusehen.
- Als interner Mitarbeiter kann ich eine einfache Wertangabe für einen Einzeltermin löschen, um unangebrachte Angaben zu entfernen.
- Als Student oder externe Person oder interner Mitarbeiter kann ich eine Dateiangabe durch Hochladen einer Datei für einen Einzeltermin erstellen, um benötigte Informationen zu hinterlegen.
- Als Student oder externe Person oder interner Mitarbeiter kann ich eine Dateiangabe für einen Einzeltermin ersetzen, um diese zu korrigieren.
- Als Student oder externe Person oder interner Mitarbeiter kann ich eine Dateiangabe für einen Einzeltermin einsehen und herunterladen, um zusätzliche Informationen für den Termin einzusehen.
- Als interner Mitarbeiter kann ich eine Dateiangabe für einen Einzeltermin löschen, um unangebrachte Angaben zu entfernen.
- Als Student oder externe Person oder interner Mitarbeiter kann ich den Fortschritt einsehen, inwieweit Angabenanforderungen durch getätigte Angaben erfüllt wurden.
- Als interner Mitarbeiter kann ich eine Angabenanforderung für eine Terminserie erstellen, um diese als Vorlage für Einzeltermine nutzen zu können.
- Als interner Mitarbeiter kann ich alle Angabenanforderungen für eine Terminserie einsehen, um diese zu überprüfen.
- Als interner Mitarbeiter kann ich eine Angabenanforderung für eine Terminserie ändern, um diese zu korrigieren.

- Als interner Mitarbeiter kann ich eine Angabenanforderung für eine Terminserie löschen, weil sie nicht mehr benötigt wird.

Terminserie

US-006 Als interner Mitarbeiter kann ich eine Terminserie erstellen, um diese für zukünftig zu vereinbarende Einzeltermine zu nutzen.

US-007 Als interner Mitarbeiter kann ich alle meine Terminserien einsehen, um einen Überblick zu erhalten.

US-008 Als interner Mitarbeiter kann ich Informationen einer Terminserie einsehen, um diese überprüfen zu können.

- Als interner Mitarbeiter kann ich eine Terminserie anpassen, um diese zu aktualisieren/korrigieren.
- Als interner Mitarbeiter kann ich eine Terminserie löschen, um nicht benötigte Terminserien zu entfernen.

Terminangebotszeitrum

US-009 Als interner Mitarbeiter kann ich einen Terminangebotszeitraum für Terminserien erstellen, damit andere Nutzer einfach Einzeltermine buchen können.

US-010 Als interner Mitarbeiter kann ich Informationen eines Terminangebotszeitraums einsehen, um diesen zu überprüfen.

US-011 Als interner Mitarbeiter kann ich einen Überblick meiner anstehenden Terminangebotszeiträume einsehen, um einen Überblick über die Terminangebotszeiträume zu bekommen.

- Als interner Mitarbeiter kann ich einen Terminangebotszeitraum ändern, um auf Änderungen reagieren zu können.
- Als interner Mitarbeiter kann ich einen Terminangebotszeitraum sperren oder entsperren, um diesen bei Buchungen nicht bzw. wieder zu berücksichtigen.
- Als interner Mitarbeiter kann ich einen Terminangebotszeitraum löschen, damit keine Terminserien mehr für den Zeitraum angeboten werden.

- Als interner Mitarbeiter kann ich eine Terminangebotszeitraumvorlage für Terminserien definieren, um bei der Erstellung von Terminangebotszeiträumen Zeit zu sparen.
- Als interner Mitarbeiter kann ich meine Terminangebotszeitraumvorlage(n) einsehen, um sie zu kontrollieren.
- Als interner Mitarbeiter kann ich eine Terminangebotszeitraumvorlage ändern, um Korrekturen vorzunehmen.
- Als interner Mitarbeiter kann ich eine Terminangebotszeitraumvorlage löschen, um nicht mehr benötigte Vorlagen zu entfernen.

US-012 Als Student oder externe Person kann ich angebotene Terminserien einer Person einsehen, um eine Vorlage für einen Einzeltermin auszusuchen.

US-013 Als Student oder externe Person kann ich buchbare Terminmöglichkeiten einer Terminserie einsehen, um einen für mich passenden Zeitpunkt für einen Einzeltermin zu finden.

US-014 Als Student oder externe Person kann ich buchbare Terminmöglichkeiten unter Berücksichtigung von Zeiträumen, an denen ich (nicht) kann, bei gegebener Terminserie einsehen, um einen Termin zu finden, an dem ich auch kann.

US-015 Als Student oder externe Person kann ich eine Terminmöglichkeit buchen, um einen Einzeltermin festzulegen.

Kommunikation/Synchronisation

- Als interner Mitarbeiter kann ich meine Email bestätigen, um Missbrauch vorzubeugen.

US-021 Als Student oder interner Mitarbeiter oder externe Person kann ich per Email bei Ereignissen benachrichtigt werden, um schnell auf Anfragen oder Änderungen reagieren zu können.

- Als Student oder interner Mitarbeiter oder externe Person kann ich per Email an einen kommenden Einzeltermin erinnert werden, um ihn nicht zu vergessen.
- Als Student oder interner Mitarbeiter oder externe Person kann ich einen Einzeltermin als Datei exportieren, um ihn in einem anderen Dienst nutzen zu können.

- Als Student oder interner Mitarbeiter kann ich meinen Kalender als Datei exportieren, um ihn in einem anderen Dienst nutzen zu können.
- Als Student oder interner Mitarbeiter kann ich den Terminkalender mit einem Kalenderdienst (z.B. Google Kalender) synchronisieren, um einen einheitlichen Terminkalender zu haben.

A.3 Weitere Anwendungsfälle

Für allgemeine Fehlerfälle siehe Abschnitt 3.8.

UC-001: Angebotene Terminserien einer Person einsehen

Akteur: Student oder externe Person (alias SEP)
Ziel: SEP ist über angebotene Terminserien einer Person informiert.
Auslöser: SEP möchte sich über angebotene Terminserien informieren.
Vorbedingungen: Ein interner Mitarbeiter, der die Terminserien anbieten kann, besitzt einen Account und hat einen gültigen Link für den Zugriff freigegeben.
Nachbedingungen: Der Student oder die externe Person kennt mögliche angebotene Terminserien.
Erfolgsszenario:

1 SEP ruft die Webseite mittels des Links auf.

2 SEP navigiert zur Anzeige angebotener Terminserien des anbietenden internen Mitarbeiters.

3 Das System filtert Terminangebotszeiträume des Anbieters nach Gültigkeit (nicht in Vergangenheit, aktiv, genügend Vorlaufzeit) und sammelt darin enthaltene Terminserien.

4 Das System zeigt entsprechende angebotene Terminserien der anbietenden Person aus Schritt 3 an.

Fehlerfälle:

1-a) Siehe allgemeiner Fehlerfall F-1.

3-a) Siehe allgemeiner Fehlerfall F-2.

Erweiterungen:

4-a) SEP möchte Informationen zu buchbaren Möglichkeiten für die Terminserie erfahren.

4-a-1 Ausführung des Anwendungsfalls UC-002

Zugehörige Anforderungen: US-012

UC-003: Einsehen buchbarer Terminmöglichkeiten einer Terminserie einer Person mit einschränkenden Zeiträumen

Akteur: Student oder externe Person (alias SEP)
Ziel: SEP kennt angebotene Terminmöglichkeiten.
Auslöser: SEP passen die angebotenen Terminbuchungsmöglichkeiten nicht. SEP möchte, dass Terminbuchungsmöglichkeiten anhand von angegebenen, zu vermeidenden Zeiträumen ausgeschlossen werden, die ihr/ihm nicht passen.
Vorbedingungen: Durchführung des Anwendungsfalls UC-002. SEP hat sich über Terminmöglichkeiten einer ausgewählten Terminserie einer Person an einem Tag informiert.
Nachbedingungen: SEP wird/werden eine/mehrere, nicht mit den unerwünschten Zeiträumen kollidierende Terminmöglichkeit(en) angeboten, falls möglich.
Erfolgsszenario:

0 Siehe Schritte des Anwendungsfalls UC-002.

1 SEP wählt aus, dass die angebotene Terminmöglichkeit(en) nicht passen.

2 Das System zeigt ein Formular an, für das man mehrere Zeiträume angeben kann.

3 SEP gibt im Formular unpassende Zeiträume mittels Start- und Enduhrzeit an.

4 SEP wählt die Funktion „Zeiträume berücksichtigen“ aus

5 Das System prüft die Formulareingaben: Sind die Uhrzeiten im richtigen Format angegeben? Sind alle erforderlichen Eingaben angegeben? Liegt der jeweilige Start vor dem Ende?, Sind angegebene Uhrzeiten/Zeitspannen 5-Minuten-granulär?

6 Das System ermittelt Terminmöglichkeiten unter zusätzlicher Berücksichtigung der angegebenen Zeiten des SEP, bewertet diese und ermittelt die Möglichkeit(en) mit der höchsten Bewertung.

7 Das System zeigt die aktualisiert(en) ermittelte(n) Terminmöglichkeit(en) an, falls verfügbar.

Fehlerfälle:

5-a) Siehe allgemeiner Fehlerfall F-3.

6-a) Siehe allgemeiner Fehlerfall F-2.

Erweiterungen:

7-a) SEP möchte eine Terminmöglichkeit buchen

7-a-1 Ausführung des Anwendungsfalls UC-004

Zugehörige Anforderungen: US-014
Zugehörige Anwendungsfälle: UC-002, UC-004

UC-005: Erstellung eines Einzeltermins

Akteur: Interner Mitarbeiter
Ziel: Bestehende Termine im System verzeichnen oder neue Termine eintragen.
Auslöser: Der interne Mitarbeiter möchte einen Einzeltermin im System eintragen.
Vorbedingungen: Der interne Mitarbeiter besitzt einen Account und hat Zugang dazu.
Nachbedingungen: Für den internen Mitarbeiter besteht ein neuer Einzeltermin.
Erfolgsszenario:

1 Interner Mitarbeiter ruft die Webseite auf und meldet sich mit Benutzername und Passwort an.

2 Interner Mitarbeiter navigiert zum Formular für die Einzelterminerstellung.

3 Interner Mitarbeiter füllt das Formular mit den Daten: Titel, Beschreibung (optional), Ort (optional), Start, Ende, Überziehungsschätzung (optional), Vorbereitungszeit (optional), Nachbereitungszeit (optional), Gäste (optional, mit Vorname, Nachname, Titel (optional), E-Mail, ob E-Mail-Benachrichtigung erwünscht)

4 Das System prüft die Angaben: Sind alle erforderlichen Felder angegeben? Ist das Format der Eingaben korrekt? Ist der Start vor dem Ende? Sind Zeitspannen positiv oder null?

5 Das System erstellt den Einzeltermin. Gäste werden ggf. per E-Mail benachrichtigt.

6 Das System zeigt dem internen Mitarbeiter eine Bestätigung der Erstellung des Einzeltermins mit Informationen des Termins an.

Fehlerfälle:

1-a) Die Zugangsdaten sind fehlerhaft: Hinweis auf Korrektur anzeigen.

1-b) Interner Mitarbeiter besitzt keinen Account: Neuen Account erstellen

4-a) Siehe allgemeiner Fehlerfall F-3.

5-a) Eine E-Mail-Benachrichtigung schlägt fehl: Fortfahren.

5-b) Siehe allgemeiner Fehlerfall F-2.

Zugehörige Anforderungen: US-003

UC-006: Erstellung eines Terminangebotszeitraums für Terminserien

Akteur: Interner Mitarbeiter
Ziel: Terminbuchungen ermöglichen
Auslöser: Interner Mitarbeiter möchte Termine anbieten.
Vorbedingungen: Der interne Mitarbeiter hat einen Account und gültige Zugangsdaten. Falls Terminserien angeboten werden sollen, müssen diese bestehen.
Nachbedingungen: Ein Terminangebotszeitraum für den internen Mitarbeiter besteht.
Erfolgsszenario:

1 Interner Mitarbeiter ruft die Webseite auf und meldet sich mit dem eigenen Benutzernamen und Passwort an.

2 Interner Mitarbeiter navigiert zum Formular der Erstellung eines Terminangebotszeitraums.

3 Interner Mitarbeiter füllt das Formular mit den Daten: Datum, Startuhrzeit, Enduhrzeit, Puffer zwischen Terminen (optional), Vorlaufzeit in Tagen (optional), Pausen (optional, jeweils mit frühester Startuhrzeit, Dauer, Maximalverschiebungsdauer (optional)), Terminserienpräferenzen (optional, Terminserie, Zeiträume (optional, jeweils mit Startuhrzeit und Enduhrzeit), allgemeine Präferenzen (optional, Art der Präferenzanmerkung), Zeitraumpräferenzen (optional, jeweils Startuhrzeit, Enduhrzeit, Zeitraumpräferenzart), Zeitpunktpräferenzen (optional, jeweils mit Uhrzeit, Zeitpunktpräferenzart))

4 Interner Mitarbeiter wählt die Funktion „Terminangebotszeitraum erstellen“ aus.

5 Das System prüft die Eingaben: Bestehen alle erforderlichen Angaben? Sind alle Angaben im richtigen Format? Sind alle angegebenen Uhrzeiten/Zeitspannen

5-Minuten-granulär? Ist jeder Start vor dem Ende? Gibt es die angegebene Terminserie? Gibt es keinen bestehenden Terminangebotszeitraum am gleichen angegebenen Datum? Beziehen sich alle Terminserienpräferenzen auf unterschiedliche Terminserien?

6 Das System erstellt den Terminangebotszeitraum.

7 Das System zeigt eine Bestätigung der Erstellung an.

Fehlerfälle:

1-a) Die Zugangsdaten sind fehlerhaft: Hinweis auf Korrektur anzeigen.

5-a) Siehe allgemeiner Fehlerfall F-3.

6-a) Siehe allgemeiner Fehlerfall F-2. Zusätzlich bleiben alle eingegebenen Daten bestehen.

Zugehörige Anforderungen: US-009

A.4 Weitere Wireframes

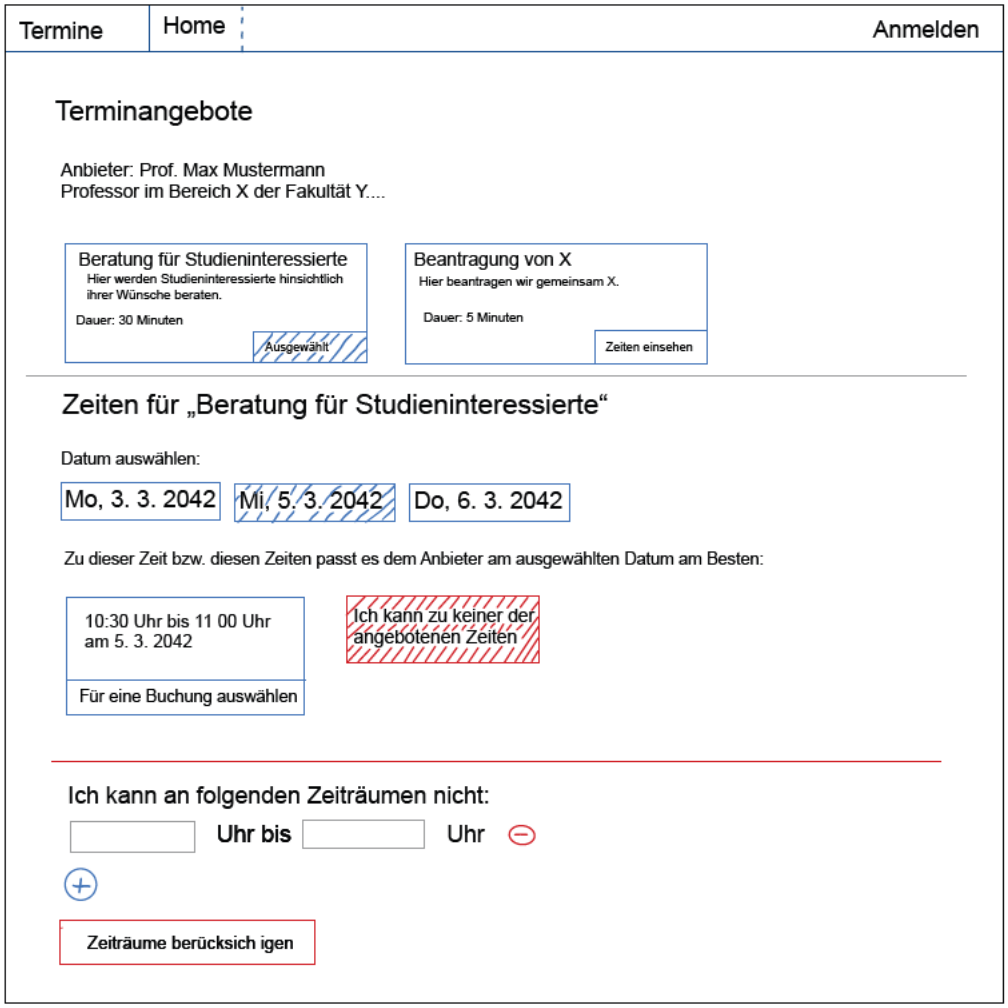

Abbildung A.1: Wireframe für die Einsicht buchbarer Terminmöglichkeiten mit einschränkenden Zeiträumen (Anwendungsfall UC-003)

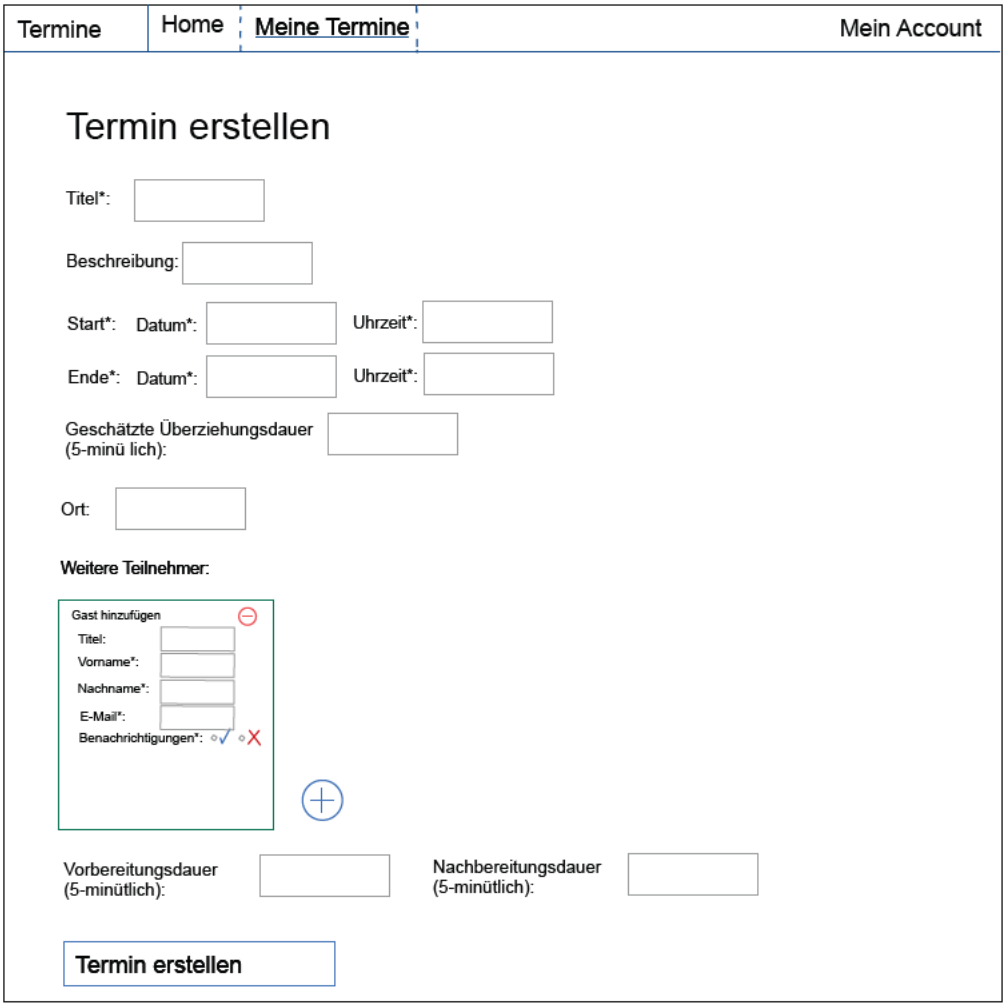

Abbildung A.2: Wireframe für die Erstellung eines Einzeltermins (Anwendungsfall UC-005)

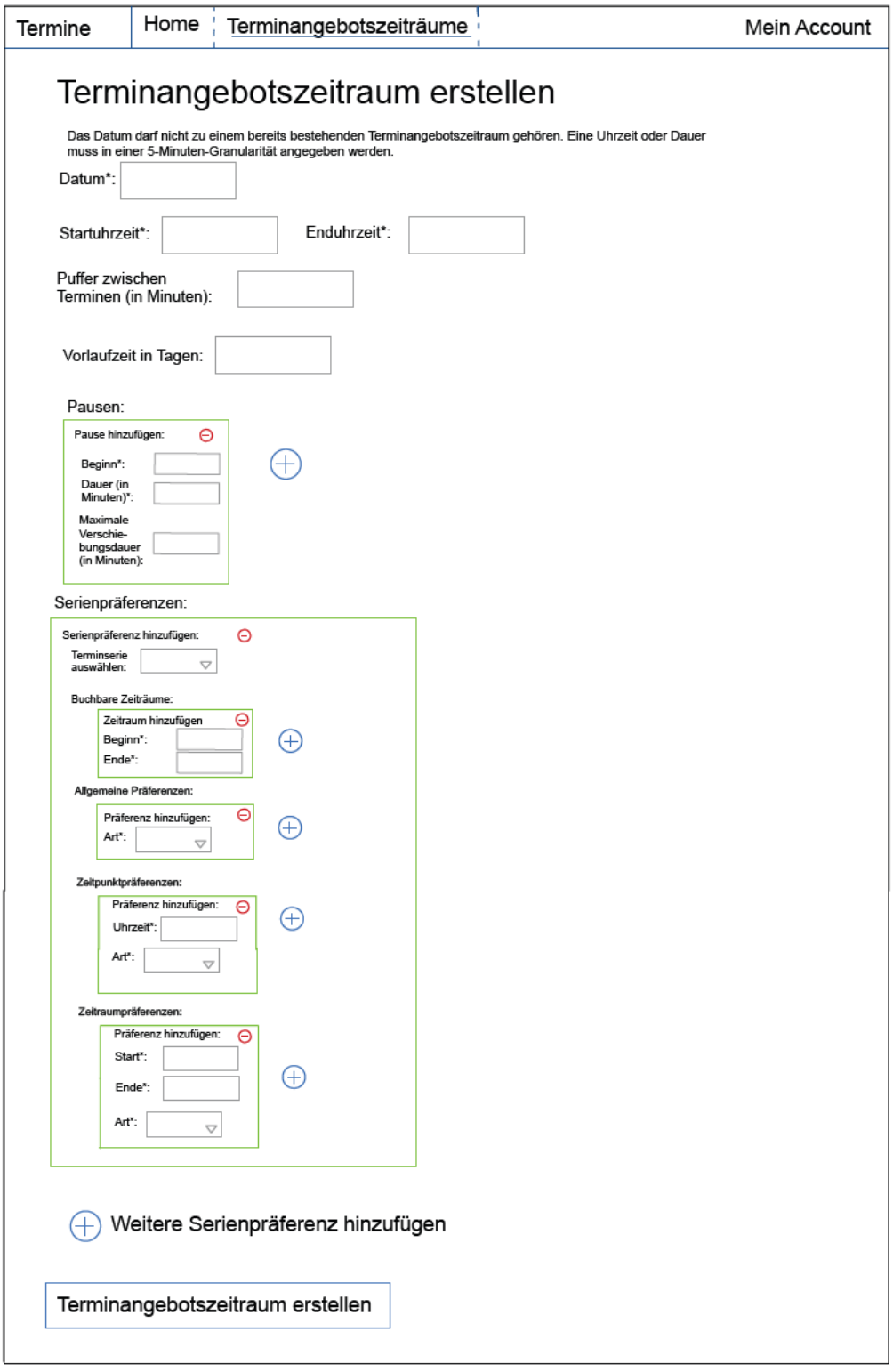

Abbildung A.3: Wireframe für die Erstellung eines Terminangebotszeitraums (Anwendungsfall UC-006)

A.5 Messergebnisse bei der Durchführung von Qualitätsattributsszenarien

Nr. der Messung	Dauer in Sekunden für den Seitenaufruf zur Erstellung eines Terminangebotszeitraums (UC-006)	Dauer in Sekunden für den Seitenaufruf zur Buchung eines Termins (UC-004)
1	2,18	2,06
2	2,67	2,92
3	3,07	2,24
4	3,24	2,16
5	2,32	2,36
6	2,93	2,16
7	2,53	2,02
8	2,94	2,27
9	2,68	1,97
10	2,34	1,92

Tabelle A.1: Messergebnisse der Durchführung des Qualitätsattributsszenarios QAS-01

Nr. der Messung	Dauer in Sekunden für die Erstellung eines Terminangebotszeitraums (UC-006)	Dauer in Sekunden für die Buchung eines Termins (UC-004)
1	0,99	0,86
2	0,97	0,86
3	0,97	0,87
4	0,98	0,86
5	0,96	0,87
6	0,97	0,86
7	1,05	0,88
8	0,96	0,86
9	0,97	0,86
10	0,96	0,86

Tabelle A.2: Messergebnisse der Durchführung des Qualitätsattributsszenarios QAS-02

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Entwurf und prototypische Umsetzung eines Systems zur Vermittlung von Terminen

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original