ingenieur
wissenschaften
htw saar

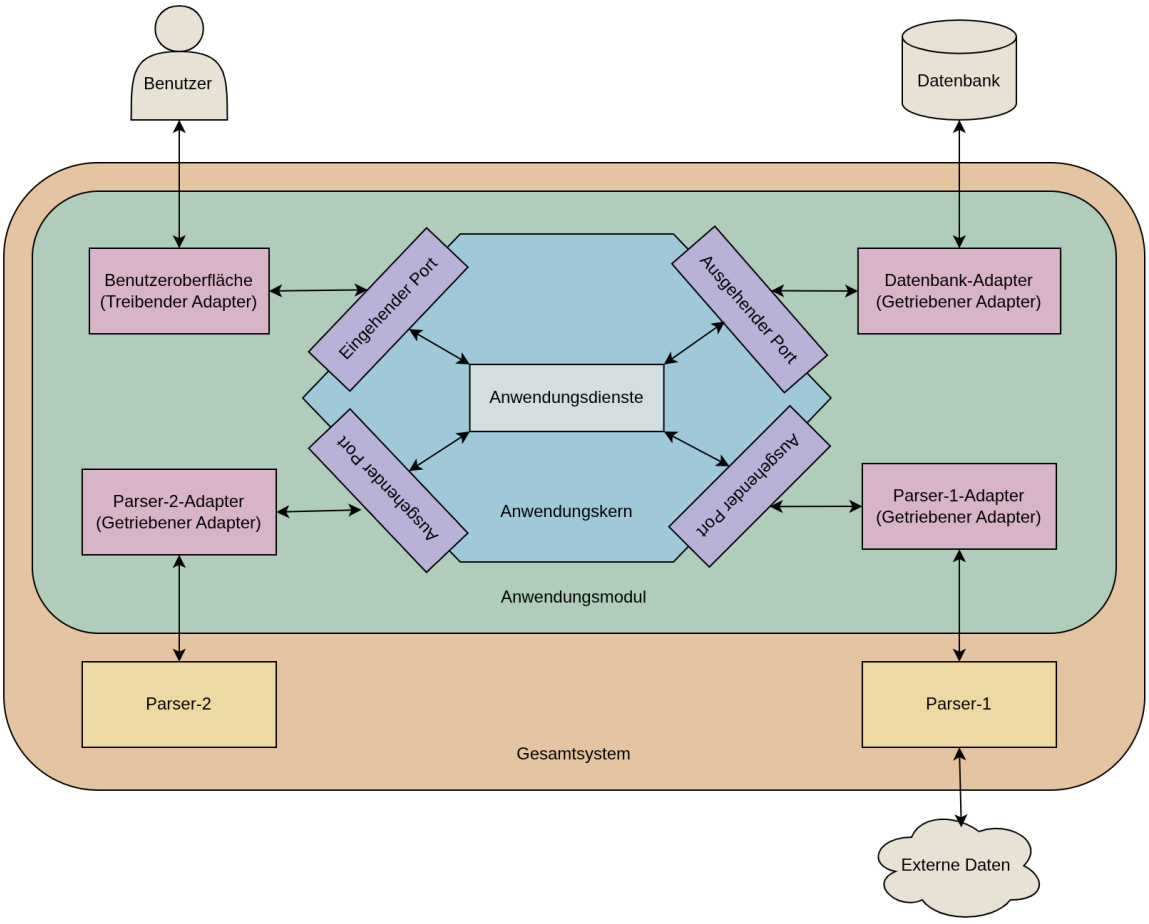
Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

STL SYSTEMTECHNIKLABOR

Konzeptionierung und Entwicklung eines webbasierten Systems zur digitalen Planung und Organisation des Studienverlaufs

Jan Straßburger

Technical Report – STL-TR-2026-02 – ISSN 2364-7167



stl.htwsaar.de

Technische Berichte des Systemtechniklabors (STL) der htw saar
Technical Reports of the System Technology Lab (STL) at htw saar
ISSN 2364-7167

Jan Straßburger: Konzeptionierung und Entwicklung eines webbasierten Systems zur digitalen Planung und Organisation des Studienverlaufs
Technical report id: STL-TR-2026-02

First published: October 2026
Last revision: September 2026
Internal review: Christopher Olbertz, André Miede

For the most recent version of this report see: https://stl.htwsaar.de/

Title image source: Jan Straßburger



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. http://creativecommons.org/licenses/by-nc-nd/4.0/

htw saar – Hochschule für Technik und Wirtschaft des Saarlandes (University of Applied Sciences)
Fakultät für Ingenieurwissenschaften (School of Engineering)
STL – Systemtechniklabor (System Technology Lab)
Prof. Dr.-Ing. André Miede (andre.miede@htwsaar.de)
Goebenstraße 40
66117 Saarbrücken, Germany
https://stl.htwsaar.de

ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Bachelor-Thesis

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)
an der Hochschule für Technik und Wirtschaft des Saarlandes
im Studiengang Praktische Informatik
der Fakultät für Ingenieurwissenschaften

Konzeptionierung und Entwicklung eines webbasierten Systems zur digitalen Planung und Organisation des Studienverlaufs

vorgelegt von
Jan Alexander Straßburger

betreut und begutachtet von
Prof. Dr.-Ing. André Miede
Herrn Christopher Olbertz

Saarbrücken, 29. September 2026

Selbständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit (bei einer Gruppenarbeit: den entsprechend gekennzeichneten Anteil der Arbeit) selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich erkläre hiermit weiterhin, dass die vorgelegte Arbeit zuvor weder von mir noch von einer anderen Person an dieser oder einer anderen Hochschule eingereicht wurde.

Darüber hinaus ist mir bekannt, dass die Unrichtigkeit dieser Erklärung eine Benotung der Arbeit mit der Note „nicht ausreichend“ zur Folge hat und einen Ausschluss von der Erbringung weiterer Prüfungsleistungen zur Folge haben kann.

Saarbrücken, 29. September 2026

Jan Alexander Straßburger

Vorwort

In dieser Arbeit wird überwiegend geschlechtsneutrale Sprache verwendet. Wo dies nicht sinnvoll möglich ist, wird das generische Maskulinum verwendet. Personenbezeichnungen im generischen Maskulinum beziehen sich dabei auf Personen aller Geschlechter.

Zusammenfassung

Die Planung eines Semesters ist für Studierende eine große Herausforderung, insbesondere an der Hochschule für Technik und Wirtschaft des Saarlandes (HTW), wo Informationen zu Modulen, Veranstaltungen und Prüfungen über mehrere verschiedene Plattformen verteilt sind und in unterschiedlichen Formaten vorliegen.

Diese Arbeit beschreibt die Konzeptionierung und Implementierung einer Webanwendung, die diese Informationen zentral bündelt und den Studierenden eine übersichtliche, auf den jeweiligen Studiengang zugeschnittene Darstellung bietet. Die Anwendung ermöglicht es den Studierenden, ihre Module, Veranstaltungen und Prüfungen zu verwalten und gibt Hinweise auf zeitliche Überschneidungen von Veranstaltungen, eine mögliche Überlastung durch zu viele Module in einem Semester sowie fehlende Voraussetzungen.

Nach einer Analyse der bestehenden Systeme und der Anforderungen wurde eine Anwendung auf Basis einer hexagonalen Architektur entwickelt, die mithilfe unabhängiger Parser-Module flexibel an unterschiedliche, sich ändernde Datenquellen angebunden werden kann. Die Anwendung verfügt über ein System zur selbstbestimmten Datenhaltung, das es den Studierenden ermöglicht, ihre Daten wahlweise lokal als Datei zu verwalten oder diese serverseitig mit ihrem Hochschul-IT-Zentrum (HIZ)-Login zu speichern.

Die Evaluation zeigt, dass die im Soll-Konzept definierten Anforderungen an die Anwendung erfüllt wurden und die Forschungsfragen zur Zusammenführung heterogener Quellen in einem flexiblen Datenmodell, zur Unabhängigkeit und Erweiterbarkeit der Architektur und zur selbstbestimmten Datenhaltung beantwortet werden konnten.

The most important property of a program is whether it accomplishes the intentions of its user.

— C.A.R. Hoare [11]

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich während der Anfertigung dieser Arbeit unterstützt haben.

Ich danke Herrn Prof. Dr. André Miede und Herrn Christopher Olbertz für die Betreuung meiner Bachelor-Thesis. Insbesondere möchte ich mich bei Herrn Olbertz für die vielen hilfreichen Hinweise und Anregungen bedanken, die mir während der Arbeit sehr geholfen haben.

Ganz besonders möchte ich mich auch bei meinen Eltern bedanken, die mir mein Studium durch ihre Unterstützung erst ermöglicht haben.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Problemstellung . . . 2
- 1.3 Zielsetzung . . . 2

2 Grundlagen **5**
- 2.1 Maven . . . 5
- 2.2 Spring und Spring Boot . . . 7
- 2.3 Java Persistence API . . . 8
- 2.4 Jackson . . . 8
- 2.5 UI-Technologien . . . 9
- 2.6 JSpecify . . . 10
- 2.7 PostgreSQL . . . 10
- 2.8 Docker . . . 11
- 2.9 Hexagonale Architektur . . . 13

3 Analyse **15**
- 3.1 Ist-Analyse . . . 15
- 3.2 Soll-Konzept . . . 19

4 Konzeptionierung **27**
- 4.1 Systemarchitektur . . . 27
- 4.2 Datenmodell . . . 29
- 4.3 Nutzerdatenspeicherung und Authentifizierung . . . 31
- 4.4 Auswahl der Technologien . . . 33

5 Implementierung **37**
- 5.1 Null-Safety durch JSpecify . . . 37
- 5.2 Umsetzung der Architektur . . . 37
- 5.3 Umsetzung des Datenmodells . . . 42
- 5.4 Migration von Benutzerdaten . . . 45
- 5.5 Benutzersitzungen und Datenspeicherung . . . 46

6 Deployment **49**

7 Evaluation **53**
- 7.1 Testvorgang . . . 53
- 7.2 Beantwortung der Forschungsfragen . . . 55

8 Fazit und Ausblick **57**
- 8.1 Fazit . . . 57
- 8.2 Ausblick . . . 57

Literatur **59**

Abbildungsverzeichnis **61**

Tabellenverzeichnis **61**

Listings **61**

Abkürzungsverzeichnis **63**

A Listings **67**
A.1 pom.xml der Hauptanwendung 67

1 Einleitung

1.1 Motivation

Während der Schulzeit wird einem Schüler klar vorgegeben, welche Fächer er belegen muss und wann diese stattfinden. Es muss nicht abgewogen werden, welche Fächer man belegen möchte und welche nicht. Erst in der Oberstufe muss teilweise selbst entschieden werden, welche Fächer man belegen möchte.

Wenn ein Schüler dann an eine Hochschule wechselt, wird er mit einer Vielzahl an Möglichkeiten und einem großen Maß an Eigenverantwortung konfrontiert. An einer Hochschule ist kein Stundenplan für einen Studierenden vorgegeben. Jeder Studierende muss sich selbst überlegen, welche Veranstaltungen er besuchen möchte und wie er diese in seinem Stundenplan unterbringen kann.

In vielen Studiengängen gibt es im Grundstudium meist nur Pflichtmodule, welche im Rahmen des Studiengangs verpflichtend sind, und nur wenige bis gar keine Wahlpflichtmodule. Jedoch müssen auch diese Pflichtmodule erst identifiziert werden und vor allem muss darauf geachtet werden, dass manche Module nur in bestimmten Semestern angeboten werden.

In späteren Semestern kommen dann viele Wahlpflichtmodule hinzu, die der Studierende besuchen kann. Oft ist es vorgegeben, dass eine bestimmte Anzahl an Wahlpflichtmodulen besucht werden muss, allerdings kann der Studierende, wie der Name schon sagt, selbst entscheiden, welche Wahlpflichtmodule er besuchen möchte. Dabei muss neben dem eigenen Interesse auf viele verschiedene Faktoren geachtet werden. Manche Veranstaltungen überschneiden sich zeitlich, manche Veranstaltungen haben andere Veranstaltungen als empfohlene Voraussetzungen und es muss auf die eigenen Kapazitäten geachtet werden.

Neben der Auswahl der Veranstaltungen selbst müssen Studierende gegen Ende des Semesters auch die zugehörigen Prüfungen planen. Studierende müssen sich aktiv für Prüfungen anmelden und dabei auf die Anmeldefristen achten. Außerdem müssen sie sich überlegen, welche Prüfungen sie in welchem Semester schreiben möchten. Dabei müssen sie auch darauf achten, dass manche Prüfungen nur in bestimmten Semestern angeboten werden. Außerdem hat man für bestimmte Prüfungen nur eine begrenzte Anzahl an Versuchen. Wenn man eine Prüfung nicht besteht, muss man abwägen, ob man diese im folgenden Semester direkt erneut ablegt oder die Wiederholung bewusst aufschiebt, um erst mehr Erfahrung zu sammeln.

Die Planung eines Semesters ist also eine komplexe Aufgabe, die viele Informationen und geeignete Werkzeuge erfordert. An der HTW kommen hierbei weitere, hochschulspezifische Herausforderungen hinzu, auf die im Folgenden näher eingegangen wird.

1.2 Problemstellung

An der HTW kommt zu diesen generellen Herausforderungen noch hinzu, dass für diese Planung notwendige Informationen nicht zentral verfügbar sind. Sie sind auf verschiedenen Webseiten und Portalen verteilt und teilweise versteckt. So müssen sich Studierende immer wieder zwischen verschiedenen Webseiten hin- und herklicken, um die benötigten Informationen und Dokumente zu finden.

Die Moduldatenbank ist ein Portal, das bereits viele grundlegende Informationen zentral bündelt. Sie listet zentral alle Module und Veranstaltungen auf, die an der Hochschule angeboten werden. Für die vollständige Semesterplanung fehlen dort jedoch einige Informationen. Es gibt keine Informationen darüber, ob Veranstaltungen im aktuellen Semester angeboten werden, wann und wo sie stattfinden, wann die Prüfungen stattfinden und welche oder wie viele Veranstaltungen für ein Studium besucht werden müssen. Hinzu kommt, dass die Moduldatenbank vor allem für neue Studierende unübersichtlich wirken kann, da die Moduldatenbank alle Studiengänge und Module der Hochschule auflistet und nicht nur die, die für den eigenen Studiengang relevant sind.

Die für die Semesterplanung notwendigen Informationen sind über mehrere unterschiedliche Systeme und Formate verteilt, etwa Studien- und Prüfungsordnungen auf der Hochschul-Homepage, eine separate Raumplaner-Anwendung sowie Dokumente in unterschiedlichen Moodle-Kursen. Eine detaillierte Analyse der bestehenden Systeme erfolgt in Abschnitt 3.1.

Zusammenfassend lassen sich die zentralen Probleme wie folgt darstellen:

- Fehlende zentrale Informationsquelle
- Teilweise versteckte Informationen in Moodle-Kursen und auf der Homepage der Hochschule
- Unterschiedliche Formate der Informationen (PDF, Excel, Webseiten)

Diese Probleme führen dazu, dass Studierende bei der Planung ihres Semesters viel unnötige Zeit und Aufwand damit verbringen, die benötigten Informationen zu finden und zusammenzutragen. Dies kann zu Frustration führen und den Studienerfolg negativ beeinflussen.

1.3 Zielsetzung

Diese Arbeit beschäftigt sich mit der Konzeptionierung und Entwicklung einer Webanwendung, die Studierende bei der Planung ihres Semesters unterstützt.

Konkret sollen die in Abschnitt 1.2 beschriebenen Probleme adressiert werden. Die Anwendung soll eine zentrale Informationsquelle bereitstellen, die alle relevanten Informationen für die Semesterplanung eines Studierenden zusammenführt. Dabei sollen die Informationen aus den verschiedenen Quellen automatisch aggregiert und nur die für den jeweiligen Studierenden relevanten Informationen angezeigt werden. Die Anwendung soll außerdem eine benutzerfreundliche Oberfläche bieten, die es den Studierenden ermöglicht, ihre Semesterplanung effizient und intuitiv durchzuführen.

Im Rahmen der Arbeit sollen insbesondere die folgenden Fragen beantwortet werden:

FF1 Wie können die aus unterschiedlichen Quellen und in unterschiedlichen Formaten vorliegenden Informationen in einem einheitlichen, aber gegenüber studiengangsspezifischen Eigenheiten flexiblen Datenmodell zusammengeführt werden?

FF2 Wie kann die Systemarchitektur der Anwendung so gestaltet werden, dass diese möglichst unabhängig von den bestehenden Systemen ist und auch in Zukunft leicht erweitert und angepasst werden kann?

FF3 Wie kann den Studierenden ermöglicht werden, selbst zu entscheiden, ob und in welcher Form ihre Daten gespeichert werden, und wie kann dabei eine konsistente Datenhaltung sichergestellt werden?

2 Grundlagen

2.1 Maven

Maven ist ein Build- und Abhängigkeitsmanagement-Tool für Java-Projekte. Es wurde entwickelt, um den Build-Prozess zu automatisieren und die Verwaltung von Projektabhängigkeiten zu vereinfachen. Im Gegensatz zu Vorgängern wie Ant, die eine hohe Flexibilität bieten, aber eine detaillierte Konfiguration erfordern, verfolgt Maven einen konventionellen Ansatz, der auf Standardstrukturen und -prozessen basiert. Maven bietet viele vordefinierte Ziele (Goals) und Plugins, die den Build-Prozess standardisieren und vereinfachen. Der Vorteil dabei ist, dass sich Entwickler auf die Implementierung der eigentlichen Funktionalität konzentrieren können, anstatt sich mit der Konfiguration des Build-Prozesses zu beschäftigen. [2, 3]

Standardmäßig verwendet Maven eine bestimmte Verzeichnisstruktur für Projekte, die als `Standard Directory Layout` bezeichnet wird. Diese Struktur definiert, wo Quellcode, Ressourcen, Tests und andere Projektartefakte abgelegt werden sollten. Durch die Einhaltung dieser Struktur können sich Entwickler schneller in neue Projekte einarbeiten und Tools können effizienter arbeiten. [4]

Maven verwendet eine deklarative Konfiguration, die in einer XML-Datei namens `pom.xml` (Project Object Model) definiert ist. Diese Datei liegt im Stammverzeichnis des Projekts und enthält Informationen über das Projekt, seine Abhängigkeiten, Plugins und andere Konfigurationsdetails. [3]

Maven bietet auch zentrale Repositories, in denen Abhängigkeiten gespeichert werden können. Dies erleichtert die Verwendung von Bibliotheken und deren Integration in Maven-Projekte. [3]

Maven unterstützt außerdem die Aufteilung von Projekten in mehrere Module. Ein Modul ist ein eigenständiges Maven-Projekt, das innerhalb eines übergeordneten Projekts (Parent Project) organisiert ist. Jedes Modul hat seine eigene `pom.xml`-Datei, in der die spezifischen Abhängigkeiten und Konfigurationen für dieses Modul definiert sind. Durch die Verwendung von Modulen können Entwickler komplexe Projekte in kleinere, wiederverwendbare Komponenten aufteilen, was die Wartbarkeit und Skalierbarkeit verbessert. Die übergeordnete `pom.xml`-Datei des Parent Projects definiert die gemeinsamen Abhängigkeiten und Konfigurationen, die von allen Modulen geerbt werden. Dies ermöglicht eine zentrale Verwaltung von Abhängigkeiten und erleichtert die Konsistenz im gesamten Projekt. [3] Listing 2.1 zeigt ein Beispiel für eine Parent `pom.xml`-Datei, welche grundlegende Informationen über das Projekt wie `groupId`, `artifactId` und `version` sowie die enthaltenen Module spezifiziert. Listing 2.2 zeigt ein Beispiel für eine Modul `pom.xml`-Datei, die die spezifischen Abhängigkeiten und Konfigurationen für ein einzelnes Modul definiert. Dieses verweist auch wieder auf die Parent `pom.xml`-Datei, um die gemeinsamen Abhängigkeiten und Konfigurationen zu erben.

Listing 2.1: Beispiel einer Parent pom.xml

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>parent-project</artifactId>
    <version>1.0.0</version>
    <packaging>pom</packaging>

    <modules>
        <module>module-a</module>
        <module>module-b</module>
    </modules>
</project>
```

Listing 2.2: Beispiel einer Modul-pom.xml

```
<project>
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>com.example</groupId>
        <artifactId>parent-project</artifactId>
        <version>1.0.0</version>
    </parent>

    <artifactId>module-a</artifactId>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

2.2 Spring und Spring Boot

Maven bildet somit die Grundlage, um Abhängigkeiten und Module in Java-Projekten zu verwalten. Dies ist insbesondere bei der Entwicklung von Anwendungen mit dem Spring Framework von Bedeutung, da Spring eine Vielzahl von Modulen und Abhängigkeiten bereitstellt, die für die Entwicklung von Unternehmensanwendungen benötigt werden. In den folgenden Abschnitten werden das Spring Framework und das darauf aufbauende Spring Boot näher erläutert.

2.2.1 Spring

Spring ist in der Java-Welt eines der am weitesten verbreiteten Frameworks für die Entwicklung von Unternehmensanwendungen. Christian Ullenboom bezeichnet Spring sogar als „Quasistandard“ in der Java-Community [22, S. 23].

Um die Entstehung und das Ziel von Spring zu verstehen, ist es wichtig, zunächst einen Blick auf die Java 2 Platform, Enterprise Edition (J2EE) zu werfen. J2EE bezeichnete eine Spezifikation für die Entwicklung und Ausführung von Enterprise-Anwendungen, deren APIs und Dienste durch Application Server bereitgestellt bzw. implementiert wurden. Die J2EE Spezifikation definiert eine Reihe von Schnittstellen, die von den Application Servern implementiert werden müssen. Diese Schnittstellen definieren unter anderem, wie Geschäftslogik, Datenzugriff, Transaktionsmanagement und Sicherheit in Unternehmensanwendungen umgesetzt werden können. Die Probleme mit J2EE waren unter anderem, dass die ersten Application Server sehr speicherintensiv waren und eine lange Startzeit hatten. Außerdem war die Geschäftslogik stark an die J2EE Schnittstellen gekoppelt. Dies erschwerte die Testbarkeit und Wartbarkeit. Des Weiteren war die Spezifikation zwar umfangreich, deckte aber nicht alle Anwendungsfälle ab. Dies führte dazu, dass viele Application Server Hersteller eigene proprietäre Erweiterungen anboten, die nicht standardisiert waren und zu einer stärkeren Abhängigkeit von spezifischen Application Servern führten.

Diese Einschränkungen führten dazu, dass Spring als leichtgewichtiges Framework entwickelt wurde, welches die Entwicklung von Unternehmensanwendungen in Java vereinfachen und gleichzeitig die Kopplung zwischen Geschäftslogik und Infrastruktur reduzieren sollte. Im Gegensatz zu J2EE verfolgte Spring einen leichtgewichtigeren Ansatz. Anstatt die Geschäftslogik eng an die durch einen Application Server definierte Infrastruktur zu binden, bietet Spring die Möglichkeit, unabhängige Komponenten zu entwickeln, die durch das Framework verwaltet werden. Ein zentrales Konzept hierbei ist die Dependency Injection (DI), bei der die Abhängigkeiten einer Komponente nicht von dieser selbst erzeugt, sondern durch einen externen Container bereitgestellt werden. Sie ermöglicht eine lose Kopplung zwischen den Komponenten einer Anwendung und erleichtert die Testbarkeit und Wartbarkeit des Codes. [22]

2.2.2 Spring Boot

Obwohl Spring vieles bei der Entwicklung von Unternehmensanwendungen vereinfachte, war die Konfiguration von größeren Spring-Anwendungen immer noch komplex, da viele Komponenten und Abhängigkeiten konfiguriert werden mussten, insbesondere in den frühen Versionen, welche häufig noch über XML-Dateien konfiguriert werden mussten.

Um dieses Problem zu lösen, wurde Spring Boot entwickelt. Spring Boot ist kein eigenständiges Framework, sondern setzt direkt auf dem bestehenden Spring Framework auf und erweitert es um Mechanismen zur automatischen Konfiguration. Die tatsächlichen Funktionen des Spring Frameworks wie DI und die Verwaltung von Komponenten bleiben unverändert. Spring Boot übernimmt lediglich einen Großteil der manuellen Konfiguration, indem es sinnvolle Standardwerte vorgibt und viele Komponenten automatisch anhand der vorhandenen Abhängigkeiten konfiguriert (Auto-Konfiguration). Eine Spring-Boot-Anwendung kann durch eine `main`-Methode gestartet werden und kann durch wenige Properties oder Klassen konfiguriert werden. Spring Boot bietet außerdem eine Reihe von Starter-Abhängigkeiten für häufige Funktionalitäten wie Datenbankzugriff, Sicherheit oder Webentwicklung. Diese enthalten bereits die benötigten Bibliotheken und Konfigurationen. Spring Boot ermöglicht es also Entwicklern, sich auf die Implementierung der Geschäftslogik zu konzentrieren, indem es den Konfigurationsaufwand reduziert. [21, 22]

2.3 Java Persistence API

Die Java Persistence API (JPA) ist eine Spezifikation für objektrelationale Frameworks (ORM-Frameworks) in Java, die im Jakarta Enterprise Edition (Jakarta EE) Standard enthalten ist. Als ORM-Frameworks werden solche bezeichnet, die das objektrelationale Mapping zwischen Datenbanktabellen und Java-Klassen ermöglichen. Die JPA definiert eine einheitliche Schnittstelle, um mit ORM-Frameworks, die heutzutage als Persistence Provider bezeichnet werden, zu interagieren. Die JPA ist nicht an ein bestimmtes ORM-Framework gebunden, sondern kann mit verschiedenen Implementierungen, wie EclipseLink oder Hibernate, verwendet werden. [20]

Spring Boot unterstützt die JPA durch den Spring Boot Starter `spring-boot-starter-data-jpa`. Dieser Starter enthält Hibernate, einen der am häufigsten verwendeten Persistence Provider, sowie Spring Data JPA. Spring Data JPA erleichtert die Erstellung von Repositories und die Durchführung von Datenbankabfragen. Mit diesem Starter müssen Entitäten nicht in einer XML-Datei konfiguriert werden, sondern können direkt in Java-Klassen definiert werden. Spring Data JPA bietet außerdem die Möglichkeit, Repositories zu erstellen, die automatisch CRUD-Operationen (Create, Read, Update, Delete) unterstützen. Durch die Definition von Methoden in den Repository-Interfaces können benutzerdefinierte Abfragen erstellt werden. [26]

2.4 Jackson

Jackson ist eine Java-Bibliothek, die hauptsächlich für die Verarbeitung von JSON-Daten verwendet wird. Sie enthält Funktionen zum Parsen und Generieren von JSON sowie zur Serialisierung und Deserialisierung von Java-Objekten. Jackson unterstützt neben JSON auch andere Datenformate wie XML, YAML und CSV. [10]

Spring Boot unterstützt Jackson als eine der Bibliotheken zur Verarbeitung von JSON-Daten. In Spring Boot 4 ist Jackson die Standardbibliothek für die Verarbeitung von JSON-Daten. Wenn sich Jackson auf dem Klassenpfad befindet, erzeugt Spring Boot automatisch eine `JsonMapper`-Bean, die anschließend in der Anwendung zur Verfügung steht. [24]

2.5 UI-Technologien

2.5.1 Jakarta Faces

Während Spring und Spring Boot die Entwicklung von Unternehmensanwendungen in Java vereinfachen, sind sie nicht für die Entwicklung von Benutzeroberflächen zuständig. Für die Entwicklung von Benutzeroberflächen gibt es verschiedene Technologien, die in Kombination mit Spring und Spring Boot verwendet werden können. Eine dieser Technologien ist Jakarta Faces, ursprünglich JavaServer Faces (JSF), ein komponentenbasiertes Framework zur Entwicklung serverseitig gerenderter Benutzeroberflächen für Java-Web-Anwendungen. JSF 1.2 wurde 2006 als Teil von Java Enterprise Edition (Java EE) standardisiert.

JSF ermöglicht die Entwicklung von Benutzeroberflächen durch die Verwendung von wiederverwendbaren Komponenten. Diese Komponenten können sowohl Standardkomponenten wie Eingabefelder, Schaltflächen, Tabellen und Listen als auch benutzerdefinierte Komponenten sein, um spezifische Anforderungen zu erfüllen. Wie in anderen UI-Frameworks können auch in JSF Komponenten miteinander verschachtelt werden, um komplexe Benutzeroberflächen zu erstellen.

Komponenten können in JSF durch die Verwendung von Facelets, einer deklarativen View-Technologie, definiert werden. Views werden in XHTML-Dateien erstellt, die die Struktur und das Layout der Benutzeroberfläche beschreiben. Facelets ermöglichen die Wiederverwendung von Komponenten und Templates, um konsistente Benutzeroberflächen zu erstellen.

Die Interaktion zwischen Benutzeroberfläche und Anwendungslogik orientiert sich am Model-View-Controller (MVC)-Muster. Die View wird mithilfe von Facelets definiert, während Managed Beans beziehungsweise CDI-Beans die Brücke zwischen der Benutzeroberfläche und der dahinterliegenden Anwendungslogik bilden. Sie stellen Zustände und Methoden bereit, die von den Komponenten der Benutzeroberfläche aufgerufen werden und ermöglichen so die Verarbeitung von Benutzereingaben und die Aktualisierung der Benutzeroberfläche.

Dadurch ist die Geschäftslogik völlig von der Benutzeroberfläche entkoppelt, was die Wartbarkeit und Testbarkeit der Anwendung verbessert. [15]

2.5.2 PrimeFaces

JSF selbst bietet nur eine begrenzte Anzahl von Standardkomponenten, die für die Entwicklung von Benutzeroberflächen verwendet werden können. Für komplexere Benutzeroberflächen gibt es verschiedene Komponentenbibliotheken, die zusätzliche UI-Komponenten bereitstellen. Eine der bekanntesten und am weitesten verbreiteten Komponentenbibliotheken ist PrimeFaces mit einer Vielzahl von häufig verwendeten UI-Komponenten, wie Buttons, Tabellen, Dialogen oder auch komplexen Komponenten wie Kalendern oder Diagrammen. Dadurch kann für häufige UI-Komponenten auf eine standardisierte, breit getestete Lösung zurückgegriffen werden. Des Weiteren bietet PrimeFaces eine einfache Möglichkeit, alle Komponenten mit einem einheitlichen Design zu versehen, indem es verschiedene Themes zur Verfügung stellt. Somit kann ein einheitliches Design für die gesamte Anwendung gewährleistet werden. [18]

2.5.3 JoinFaces

Sowohl Jakarta Faces als auch PrimeFaces sind eigentlich für den Einsatz in einer Jakarta EE Umgebung konzipiert und nicht für die Verwendung mit Spring Boot. Um beide Technologien dennoch in einer Spring-Boot-Anwendung zu verwenden, gibt es das Projekt JoinFaces. JoinFaces ist ein Open-Source-Projekt, das die Integration von JSF in Spring-Boot-Anwendungen erleichtert. Durch das Hinzufügen entsprechender Spring-Boot-Starter werden automatisch die notwendigen Beans und Konfigurationen bereitgestellt, um JSF-Seiten innerhalb einer Spring-Boot-Anwendung zu verwenden. JoinFaces unterstützt verschiedene JSF-Implementierungen wie Mojarra und MyFaces sowie unterschiedliche Servlet-Container. Darüber hinaus ermöglicht es die Integration verschiedener JSF-Komponentenbibliotheken, darunter PrimeFaces, RichFaces und IceFaces. Des Weiteren können sowohl JSF- als auch CDI-Annotationen, beispielsweise `@RequestScoped`, verwendet werden. [14]

2.6 JSpecify

JSpecify ist ein Standard, der Annotationen für die statische Code-Analyse in Java definiert. Er versucht, ein grundlegendes Problem von Java zu lösen: Die Nullability von Referenzen wird nicht im Typsystem abgebildet. Zwar haben viele Tools und Bibliotheken dieses Problem bereits mit eigenen Annotationen versucht zu lösen, jedoch gab es bislang keinen einheitlichen Standard, auf den sich alle Beteiligten einigen konnten. JSpecify versucht daher einen solchen Standard zu etablieren. Zu den Mitgliedern des Projekts gehören unter anderem Google, JetBrains, Meta, Microsoft und Oracle. [12]

JSpecify definiert hierfür Annotationen, die angeben, ob eine Referenz `null` sein kann oder nicht. Diese Annotationen können dann von statischen Code-Analyse-Tools wie IntelliJ IDEA verwendet werden, um mögliche `NullPointerExceptions` bereits zur Kompilierzeit zu erkennen. [12, 13]

Außerhalb eines `@NullMarked`-Kontexts geht JSpecify davon aus, dass Referenzen `null` sein dürfen, wenn keine entsprechende Annotation vorhanden ist. Um dieses Verhalten zu ändern, kann die Annotation `@NullMarked` auf ein Package oder eine Klasse angewendet werden. Sie legt fest, dass alle Referenzen innerhalb des Gültigkeitsbereichs als `non-null` betrachtet werden, sofern keine andere Annotation vorhanden ist. Eine Referenz kann anschließend mit der Annotation `@Nullable` versehen werden, um anzugeben, dass sie null sein darf. Dies ermöglicht es auch, eine bestehende Codebasis schrittweise auf die Verwendung von JSpecify zu migrieren, indem zunächst nur bestimmte Packages oder Klassen annotiert werden. [12]

2.7 PostgreSQL

PostgreSQL ist ein Open-Source-objektrelationales Datenbankmanagementsystem, das seit fast 40 Jahren aktiv entwickelt wird. Es ist für seine Stabilität, Zuverlässigkeit und hohe Performance bekannt. PostgreSQL unterstützt relationale Datenbankmodelle sowie Structured Query Language (SQL) und bietet eine Vielzahl von Funktionen, wie Transaktionen, Indizes, Views, Trigger und Stored Procedures. PostgreSQL ist plattformunabhängig und kann auf verschiedenen Betriebssystemen wie Linux, Windows und macOS

eingesetzt werden. Darüber hinaus ist PostgreSQL für seine hohe Erweiterbarkeit bekannt, da es die Möglichkeit bietet, benutzerdefinierte Datentypen, Operatoren und Funktionen zu erstellen. [16]

PostgreSQL bietet außerdem native Datentypen, um JavaScript Object Notation (JSON)-Daten zu speichern und zu verarbeiten. Im Gegensatz zur Verwendung des `TEXT`-Datentyps, der die Speicherung von JSON-Daten als einfache Zeichenkette ermöglicht, bieten die Datentypen `JSON` und `JSONB` eine strukturierte Speicherung von JSON-Daten. PostgreSQL prüft beim Einfügen der Daten auf gültige JSON-Syntax und ermöglicht das Ausführen von JSON-spezifischen Operationen wie das Abfragen von verschachtelten Objekten und Arrays. Der Unterschied zwischen `JSON` und `JSONB` liegt in der Art der Speicherung. Während `JSON` die Daten als Text speichert, wobei auch Whitespace und Reihenfolge von Schlüsseln erhalten bleiben, speichert `JSONB` die Daten in einem binären Format. `JSONB` unterstützt die Erstellung von Indizes, was die Effizienz von Abfragen verbessert. Das Speichern von `JSONB` kann jedoch mit höherem Arbeitsaufwand verbunden sein, da die Daten erst in ein binäres Format konvertiert werden müssen. [17]

2.8 Docker

Docker ist eine Plattform, die es ermöglicht, Anwendungen in Containern zu verpacken und auszuführen. Dadurch müssen sich Entwickler nicht um Abhängigkeiten und Konfiguration kümmern und die Anwendung läuft auf jedem System gleich. [8]

Das erste Konzept, welches man kennen muss, um Docker zu verstehen, ist das Konzept des Images. Ein Image ist eine unveränderliche Vorlage mit den Anweisungen, wie ein Container erstellt werden soll. Ein Image kann auf einem anderen Image basieren und enthält alle notwendigen Dateien, Bibliotheken und Abhängigkeiten, die für die Ausführung einer Anwendung erforderlich sind. Es enthält außerdem die Anweisungen zum Starten der Anwendung. Ein Image wird in einer `Dockerfile`-Datei definiert, welche eine Reihe von Anweisungen enthält, die Docker verwendet, um das Image zu erstellen. Ein Dockerfile kann beispielsweise Anweisungen zum Installieren von Software, zum Kopieren von Dateien oder zum Ausführen von Befehlen enthalten. [8]

Sobald ein Image erstellt wurde, kann es verwendet werden, um einen Container zu starten. Ein Container ist eine laufende Instanz eines Images. Er wird auf dem Host-System ausgeführt, ist aber vom restlichen System und anderen Containern isoliert. Im Gegensatz zu virtuellen Maschinen teilen sich Container den Kernel des Host-Systems, was sie leichter und schneller macht. Eine Gegenüberstellung der Architektur von Docker-Containern und virtuellen Maschinen ist in Abbildung 2.1 dargestellt. Container können auf jedem System ausgeführt werden, das Docker unterstützt, unabhängig von der zugrunde liegenden Hardware oder dem Betriebssystem. [8, 9]

Docker folgt einer Client-Server-Architektur, bei der der Docker-Client Befehle an den Docker-Daemon sendet, welcher die Container verwaltet. Der Docker-Daemon ist für das Erstellen, Starten und Stoppen von Containern verantwortlich und kann auf einem lokalen System oder auf einem Remote-Server ausgeführt werden. Der Docker-Client kann über die Befehlszeile oder über eine API verwendet werden, um mit dem Docker-Daemon zu kommunizieren. [8]

Der Docker-Daemon kann Images aus einer Docker-Registry herunterladen, welche ein Speicherort für private und öffentliche Images ist. Die bekannteste Docker-Registry ist

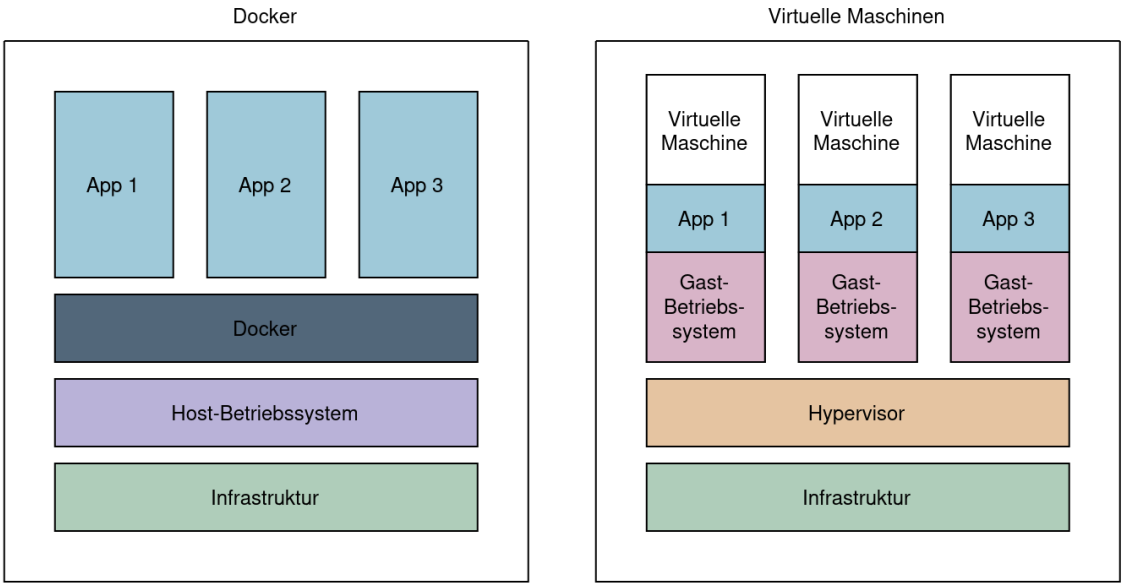


Abbildung 2.1: Docker vs. Virtual Machine (eigene Darstellung, nach [9])

Docker Hub, aber es gibt auch andere Registries wie Google Container Registry oder Amazon Elastic Container Registry. [8]

Eine Anwendung setzt sich in der Regel aus mehreren Containern zusammen, zum Beispiel eine Webanwendung, die aus einem Frontend, einem Backend und einer Datenbank besteht, welche alle in separaten Containern laufen. Um diese Container zu orchestrieren, kann man Docker Compose verwenden. Docker Compose ist ein Tool, mit dem man Multi-Container-Anwendungen definieren und ausführen kann. Es verwendet eine YAML-Datei, um die Konfiguration der Container zu definieren, einschließlich der Images, Netzwerke, Volumes und Umgebungsvariablen, die sie verwenden. Mit einem einzigen Befehl kann man dann alle Container starten, stoppen oder neu starten. [7] Listing 2.3 zeigt ein Beispiel für eine Docker-Compose-Datei, die eine Anwendung mit einem Frontend, einem Backend und einer Datenbank definiert.

Listing 2.3: Beispiel einer Docker-Compose-Datei

```
services:
  frontend:
    image: my-frontend-image:latest
    ports:
      - "80:80"
    depends_on:
      - backend
  backend:
    image: my-backend-image:latest
    ports:
      - "8080:8080"
    depends_on:
      - db
  db:
    image: postgres:latest
    environment:
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypassword
```

```
    POSTGRES_DB: mydatabase
  ports:
    - "5432:5432"
```

2.9 Hexagonale Architektur

Die hexagonale Architektur, auch bekannt als Ports-and-Adapters-Architektur, ist ein Softwarearchitekturmuster, das darauf abzielt, die Kernlogik einer Anwendung von den äußeren Einflüssen wie Benutzerschnittstellen, Datenbanken und externen Diensten zu trennen. Dieses Muster wurde 2005 von Alistair Cockburn entwickelt und fördert die Modularität und Testbarkeit von Software.

Eines der größten Probleme bei der Strukturierung von Softwareanwendungen ist die Abhängigkeit von externen Systemen und Technologien. In traditionellen Architekturen passiert es oft, dass die Kernlogik der Anwendung direkt mit der Datenbank, der Benutzeroberfläche oder anderen externen Systemen verknüpft ist. Dies erschwert die Testbarkeit und Wartbarkeit der Anwendung, da zum Beispiel beim Testen der Kernlogik auch die externen Systeme verfügbar sein müssen oder wenn Änderungen an diesen Systemen vorgenommen werden, die Kernlogik angepasst werden muss. Des Weiteren ist es oftmals schwierig, die Anwendung auf andere Plattformen oder Technologien zu portieren, da die Kernlogik stark an die spezifischen Implementierungen der externen Systeme gebunden ist.

Bei der hexagonalen Architektur wird die Kernlogik der Anwendung in einem zentralen Hexagon platziert mit sogenannten Ports, die die Schnittstellen zu den externen Systemen darstellen. Diese Ports definieren, wie die Kernlogik mit der Außenwelt interagiert, ohne dass sie direkt von den Implementierungen der externen Systeme abhängig ist. Dies ermöglicht es, die Kernlogik unabhängig von den äußeren Einflüssen zu entwickeln und zu testen. Die externen Systeme werden durch Adapter implementiert, die die Ports der Kernlogik nutzen, um mit den spezifischen Implementierungen der externen Systeme zu kommunizieren. Die Sechseckform ist dabei nicht als feste Anzahl von sechs Schnittstellen zu verstehen, sondern wurde gewählt, um Platz für eine beliebige Anzahl von Ports zu schaffen, ohne die bei klassischen Schichtenarchitekturen übliche Vorstellung von „oben“ und „unten“ zu erzeugen [5]. Abbildung 2.2 zeigt eine schematische Darstellung der hexagonalen Architektur.

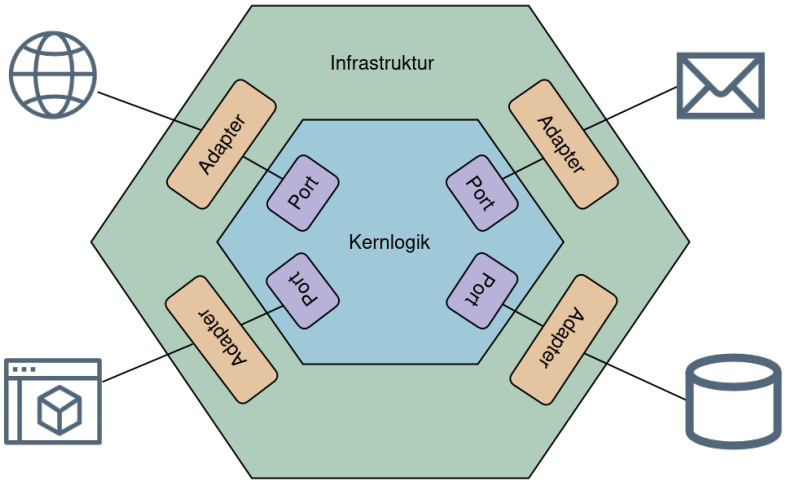


Abbildung 2.2: Hexagonale Architektur (eigene Darstellung)

3 Analyse

3.1 Ist-Analyse

Bevor eine Anwendung geplant und entwickelt werden kann, ist es wichtig, die Anforderungen und den aktuellen Stand der Technik zu analysieren. In dieser Ist-Analyse werden die bestehenden Systeme, Technologien und Prozesse untersucht, um ein besseres Verständnis für die Ausgangssituation zu gewinnen. Ziel ist es, die Stärken und Schwächen der aktuellen Lösung zu identifizieren und mögliche Verbesserungsansätze zu erarbeiten.

Die Ist-Analyse basiert auf den eigenen Erfahrungen und Beobachtungen als Studierender des Studiengangs Praktische Informatik in der Fakultät Ingenieurswissenschaften an der HTW. Es wurden die verschiedenen Systeme und Plattformen untersucht, die für einen Studierenden für die Planung seines Semesters oder Gesamtstudiums relevant sind. Außerdem wurden die Anforderungen und Bedürfnisse der Studierenden analysiert, um ein besseres Verständnis für die Ausgangssituation zu gewinnen. Die Ergebnisse der Ist-Analyse werden in den folgenden Abschnitten dargestellt.

3.1.1 Ausgangssituation und Kontext

Als Studierender an einer Hochschule ist es notwendig, einzelne Semester und den gesamten Studienverlauf zu planen. Für den Studienverlauf ist es wichtig, dass alle Pflichtmodule abgeschlossen werden und dass die Mindestzahl an European Credit Transfer and Accumulation System (ECTS)-Punkten erreicht wird. Außerdem gibt es in jedem Studiengang Extraregelungen, wie zum Beispiel, dass eine bestimmte Anzahl an ECTS-Punkten in Wahlmodulen oder nicht-technischen Modulen erreicht werden muss. Die Planung eines einzelnen Semesters ist ebenfalls nicht trivial, da es viele Faktoren gibt, die berücksichtigt werden müssen. Zum Beispiel müssen die Module, die im Semester angeboten werden, mit den Modulen, die der Studierende noch absolvieren muss, abgeglichen werden. Außerdem müssen die Module so gewählt werden, dass sich die Termine der Module nicht überschneiden und dass die Arbeitslast des Semesters für den Studierenden stemmbar ist. Außerdem gibt es Module, die aufeinander aufbauen und daher in einer bestimmten Reihenfolge belegt werden sollten. Die Planung eines Semesters ist daher eine komplexe Aufgabe, die viel Zeit und Aufwand erfordert.

3.1.2 Beteiligte Systeme im Überblick

Das größte Problem bei der Planung eines Semesters ist, dass es keine zentrale Plattform gibt, die alle relevanten Informationen bereitstellt. Stattdessen müssen Studierende verschiedene Systeme und Plattformen nutzen, um die notwendigen Informationen zu erhalten. Tabelle 3.1 gibt einen Überblick über die beteiligten Systeme und deren Zweck, Zugriffsweg und Reibungspunkte.

Tabelle 3.1: Übersicht der beteiligten Systeme

System	**Zweck**	**Zugriffsweg**	**Reibungspunkte**
Homepage	Semesterzeiten und Studien- und Prüfungsordnungen	Website	Unübersichtlichkeit der Homepage
Moduldatenbank	Modulbeschreibungen	Website	• Unzureichende Suchfunktion • Keine Angabe zur aktuellen Verfügbarkeit • Keine Kennzeichnung als technisch/nicht-technisch
SIM Portal	Notenübersicht, ECTS-Stand	Website	Unübersichtliche Benutzeroberfläche
Raumplanungssystem	Stundenplan	Website	Unübersichtlichkeit, da alle Module eines Semesters in einem Kalender dargestellt werden
Prüfungsplan	Prüfungstermine und -orte	Excel-Datei im Moodle-Kurs	• Schwer auffindbar • Unübersichtlich, da alle Prüfungen der Fakultät in einer Datei erfasst sind • Benötigt Excel oder vergleichbares Programm
Wahlpflichtkatalog	Angebotene Wahlmodule für das aktuelle Semester	PDF-Datei im Moodle-Kurs	Schwer auffindbar und erst kurz vor oder kurz nach Semesterbeginn verfügbar

3.1.3 Prozessbeschreibung der Semesterplanung

Die Planung eines Semesters für einen Studierenden ist ein iterativer Prozess, der mehrere Schritte umfasst.

Zunächst muss der Studierende herausfinden, wie viele ECTS-Punkte für das Studium benötigt werden und wie viele bereits erreicht wurden. Die benötigten ECTS-Punkte können in der Studien- und Prüfungsordnung und die bereits erreichten ECTS-Punkte in der Notenübersicht oder im SIM Portal gefunden werden. Er muss außerdem in der Studien- und Prüfungsordnung nach den Vorgaben bezüglich Mindest- und Maximalanzahl an ECTS-Punkten in bestimmten Modulgruppen suchen.

Anschließend müssen die Pflichtmodule für das aktuelle Semester identifiziert werden. Diese können in der Moduldatenbank oder in der Studien- und Prüfungsordnung gefunden werden. Wenn das Studium Wahlmodule enthält und der Studierende diese im aktuellen Semester absolvieren möchte, müssen diese ebenfalls in der Moduldatenbank recherchiert werden. Hier muss jedoch auch darauf geachtet werden, welche Wahlmodule im aktuellen Semester angeboten werden. Diese Information findet sich im Wahlpflichtkatalog. Sowohl bei Pflicht- als auch bei Wahlmodulen muss darauf geachtet werden, ob die Module aufeinander aufbauen und in der richtigen Reihenfolge belegt werden sollten. Diese Informationen können in der Moduldatenbank gefunden werden.

Nachdem die Module identifiziert wurden, müssen die Termine der Module recherchiert werden. Diese können im Raumplanungssystem gefunden werden. Vor allem bei Wahlmodulen kommt es oft dazu, dass sich Vorlesungstermine überschneiden. In diesem Fall muss der Studierende entscheiden, welche Module er belegen möchte und welche er nicht belegen kann.

Nachdem die Module ausgewählt wurden, muss der Studierende die Arbeitslast des Semesters abschätzen. Diese kann in der Moduldatenbank aus den ECTS-Punkten der Module abgeleitet werden. Wenn die Arbeitslast zu hoch ist, muss der Studierende seine Auswahl an Modulen überdenken und gegebenenfalls ändern.

Nachdem die Module gewählt wurden und sich das Semester dem Ende nähert, muss der Studierende die Prüfungstermine recherchieren. Diese können im Prüfungsplan gefunden werden. Es muss geplant werden, wie viel Zeit für die Vorbereitung auf die Prüfungen benötigt wird und welche Prüfungen er überhaupt ablegen möchte. Wenn er sich entschieden hat, welche Prüfungen er ablegen möchte, muss er sich für diese Prüfungen anmelden. Die Anmeldung erfolgt über das SIM Portal und muss bis spätestens zwei Wochen vor der Prüfung erfolgen. Wenn der Studierende die Prüfungen erfolgreich abgelegt hat, muss er seine Noten im SIM Portal einsehen und seinen ECTS-Stand aktualisieren.

Abbildung 3.1 zeigt ein Swimlane-Diagramm, das den Prozess der Semesterplanung aus Sicht eines Studierenden darstellt. Die einzelnen Schritte des Prozesses sind in den verschiedenen Spalten dargestellt, die jeweils einem beteiligten System zugeordnet sind. Die Pfeile zeigen die Abhängigkeiten zwischen den einzelnen Schritten und die Reihenfolge, in der sie durchgeführt werden müssen. Das Diagramm verdeutlicht die Komplexität des Prozesses und die Vielzahl an beteiligten Systemen und Plattformen.

3.1.4 Identifizierte Probleme und Herausforderungen

Konkret lassen sich die folgenden Probleme und Herausforderungen bei der Planung eines Semesters für einen Studierenden identifizieren:

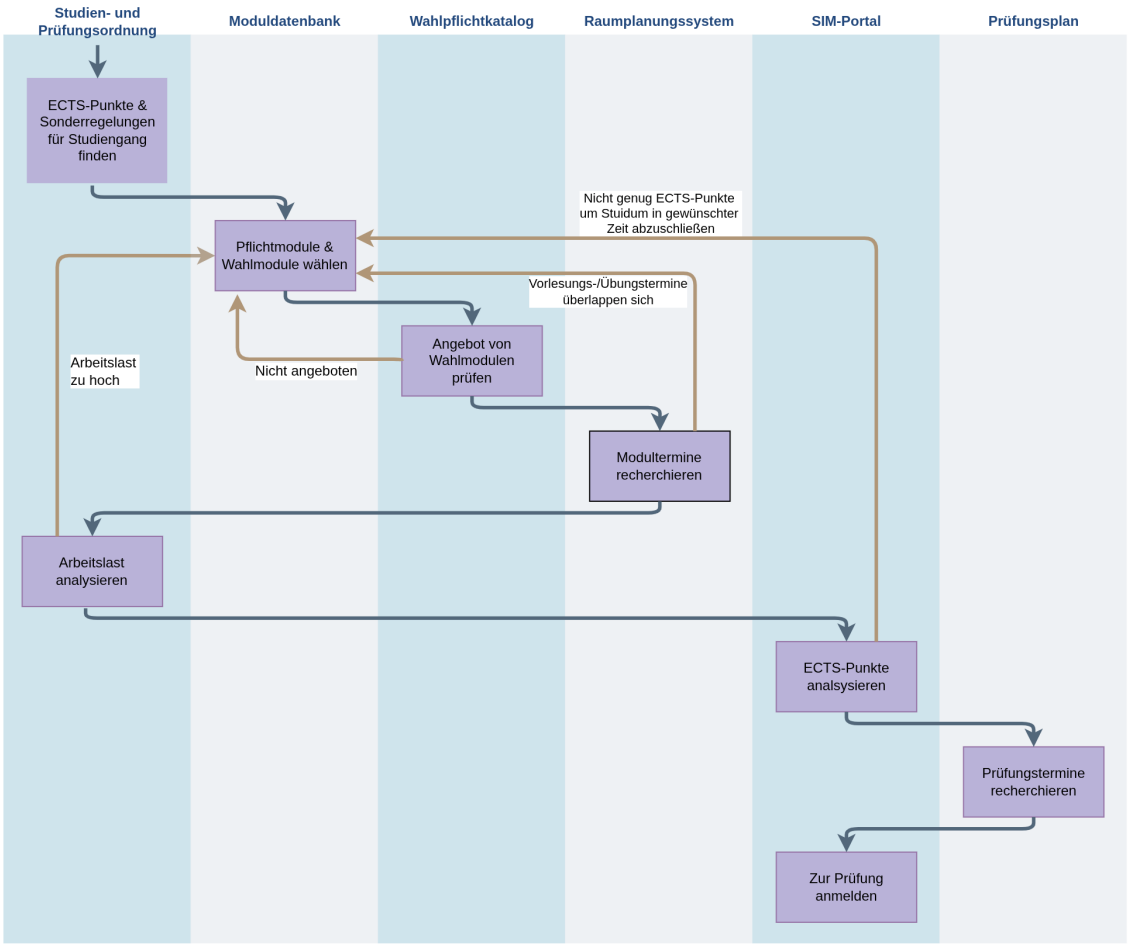


Abbildung 3.1: Swimlane-Diagramm der Semesterplanung (eigene Darstellung)

- **Informationsverteilung:** Wie in Abbildung 3.1 ersichtlich, sind für die Planung eines Semesters mindestens sechs verschiedene Systeme und Plattformen relevant, zwischen denen ohne technische Verknüpfung gewechselt werden muss. Dies führt dazu, dass Studierende viel Zeit damit verbringen müssen, die benötigten Informationen zu finden, anstatt sich auf die tatsächliche Planung des Semesters zu konzentrieren. Es kann außerdem dazu führen, dass wichtige Informationen übersehen werden und dass die Planung des Semesters fehlerhaft ist.
- **Versteckte Informationen:** Informationen sind teilweise tief in den Systemen und Plattformen versteckt, wie zum Beispiel die Informationen zu den Wahlmodulen im Wahlpflichtkatalog, welcher in einem bestimmten Abschnitt eines speziellen Moodle-Kurses zu finden ist. Dies erschwert es Studierenden selbst innerhalb eines Systems, die notwendigen Informationen zu finden und kann ebenfalls dazu führen, dass wichtige Informationen übersehen werden.
- **Formatvielfalt:** Informationen sind in verschiedenen Formaten verfügbar, wie zum Beispiel in PDF-Dateien, Excel-Dateien oder auf Webseiten. Dies erschwert es den Studierenden, die Informationen zu vergleichen und zu analysieren und erfordert teilweise die Nutzung von zusätzlichen Programmen oder Tools, um die Informationen zu verarbeiten.
- **Späte Verfügbarkeit:** Informationen sind teilweise erst sehr spät verfügbar, wie zum

Beispiel der Wahlpflichtkatalog, der erst kurz vor oder kurz nach Semesterbeginn veröffentlicht wird. Dies erschwert es den Studierenden, ihre Planung rechtzeitig abzuschließen und kann dazu führen, dass sie wichtige Fristen verpassen.

Vor allem die Fragmentierung der Informationen auf verschiedene Systeme und Plattformen sowie deren teils versteckte Ablage und späte Verfügbarkeit stellen eine große Herausforderung dar, da diese die Planung zeitaufwendig, komplex und fehleranfällig machen. Die Ist-Analyse hat gezeigt, dass es notwendig ist, Informationen zentral zu bündeln und die Planung eines Semesters zu vereinfachen. Die hier identifizierten Schwachstellen bilden die Grundlage für die im nächsten Kapitel vorgestellten Lösungsansätze und die Entwicklung einer Anwendung, die Studierenden bei der Planung ihres Semesters unterstützt.

3.2 Soll-Konzept

Nachdem in der Ist-Analyse der aktuelle Stand und die Probleme des Systems analysiert wurden, werden in diesem Kapitel die Anforderungen an das neue System beschrieben. Dabei werden sowohl funktionale als auch nicht-funktionale Anforderungen betrachtet. Jede Anforderung wird mit einer eindeutigen ID versehen, die im weiteren Verlauf der Arbeit verwendet wird, um auf die Anforderung zu verweisen. Diese ID trifft keine Aussage über die Priorität der Anforderung, sondern dient lediglich der eindeutigen Identifikation.

3.2.1 Funktionale Anforderungen

3.2.1.1 Basisfunktionen

Diese Anforderungen beschreiben die grundlegenden Funktionen, auf denen die eigentliche Semesterplanung aufbaut.

FA-1: Datenpersistenz Die Anwendung speichert die Daten, die ein Studierender in der Anwendung eingibt, im Browser. Die Anwendung erlaubt es dem Studierenden, diese Daten als Datei zu exportieren. Besagte Datei kann zu einem späteren Zeitpunkt wieder in die Anwendung importiert werden, sodass der Studierende seine Daten nicht erneut eingeben muss. Alternativ kann der Studierende sich optional über das HIZ-Login in der Anwendung anmelden und seine Daten auf dem Server speichern. Der Studierende kann sich zu einem späteren Zeitpunkt wieder in der Anwendung anmelden und seine Daten werden automatisch geladen.

FA-2: Studiengangwahl Die Anwendung ermöglicht es dem Studierenden, seinen Studiengang auszuwählen. Die Auswahl des Studiengangs beeinflusst die Auswahl der Module, die der Studierende in der Anwendung auswählen kann. Ein Studierender kann nur einen Studiengang gleichzeitig studieren.

FA-3: Semesterverwaltung Die Anwendung ermöglicht es dem Studierenden, mehrere Semester anzulegen, sodass der Studierende seine Module und Prüfungsergebnisse für jedes Semester separat verwalten kann. Diese Semester sollen eindeutig einer tatsächlichen Semesterzeit zugeordnet werden (z.B. Wintersemester 2025/2026).

3.2.1.2 Modulplanung

Die Anforderungen in diesem Abschnitt beschreiben Funktionen, die dem Studierenden bei der Auswahl und Planung seiner Module helfen sollen.

FA-4: Modulwahl Die Anwendung ermöglicht es dem Studierenden, Module für ein Semester auszuwählen. Die Auswahl der Module ist abhängig von dem gewählten Studiengang und den bereits gewählten Modulen. Ein Studierender kann nur Module auswählen, die in seinem Studiengang im aktuellen Semester angeboten werden.

FA-5: Prüfungsstatus eines Moduls verwalten Die Anwendung ermöglicht es dem Studierenden, den Prüfungsstatus eines Moduls in einem Semester zu ändern. Ein Studierender kann den Status der Prüfung auf „Erfolgreich abgeschlossen“, „Gescheitert“ oder „Nicht versucht“ setzen. Der Status eines Moduls beeinflusst die Berechnung der Gesamt-ECTS-Punkte. Es werden keine konkreten Noten gespeichert, sondern nur der Status der Prüfung.

FA-6: Abhängigkeitsprüfung Wenn ein Studierender ein Modul auswählt, welches ein anderes Modul als Voraussetzung empfiehlt und der Studierende dieses Modul noch nicht erfolgreich abgeschlossen hat, wird dem Studierenden eine Warnung angezeigt, dass er das Modul nicht auswählen sollte. Der Studierende kann das Modul trotzdem auswählen. Dadurch soll dem Studierenden ein besserer Überblick über die Abhängigkeiten der Module gegeben werden, sodass dieser besser entscheiden kann, welche Module er in einem Semester wählen sollte.

FA-7: Module mit Fehlversuchen hervorheben Wenn ein Studierender ein Modul auswählt, welches er bereits ein- oder mehrmalig versucht und nicht bestanden hat, wird dem Studierenden eine Warnung angezeigt. Der Studierende kann das Modul trotzdem auswählen. Dadurch soll dem Studierenden dabei geholfen werden, zu priorisieren, für welche Module er mehr Zeit und Energie aufwenden sollte.

3.2.1.3 Auswertung und Übersicht

Hier werden Anforderungen an Funktionen beschrieben, die dem Studierenden dabei helfen sollen, seine Module und sein Studium besser zu überblicken und auszuwerten.

FA-8: Gesamt-ECTS für Semester Die Anwendung berechnet die Gesamt-ECTS-Punkte und die Wahlpflicht-ECTS-Punkte für ein Semester basierend auf den gewählten Modulen, was dem Studierenden helfen soll, einzuschätzen, wie viele und welche Module er in einem Semester wählen sollte.

FA-9: Arbeitsaufwand für Semester Die Anwendung berechnet basierend auf den gewählten Modulen und deren ECTS-Punkten den Arbeitsaufwand für ein Semester. Es werden die Veranstaltungsstunden sowie die Selbstlernzeit berechnet und in Stunden angezeigt. Es wird dem Studierenden klar gezeigt, wie diese berechnet wurden. Es wird außerdem eine Warnung bei einem zu hohen Arbeitsaufwand angezeigt. Dies soll dem Studierenden helfen, sich nicht in einem Semester zu überlasten und die Module besser zu priorisieren.

FA-10: Studiumsübersicht Die Anwendung zeigt dem Studierenden eine Übersicht über sein gesamtes Studium. In dieser Übersicht wird dem Studierenden angezeigt, wie viele ECTS-Punkte er bereits gesammelt hat und wie viele er noch benötigt, um sein

Studium erfolgreich abzuschließen. Außerdem wird dem Studierenden angezeigt, wie viele Wahlpflicht-ECTS-Punkte er bereits gesammelt hat und wie viele er noch benötigt. Das soll dem Studierenden einen besseren Überblick über den Fortschritt seines Gesamtstudiums geben.

3.2.1.4 Zeitplanung

Diese Anforderungen beschreiben die Funktionen, die dem Studierenden dabei helfen sollen, seine Zeit besser zu planen und zu organisieren und einen besseren Überblick über seine Veranstaltungen und Prüfungen zu bekommen.

FA-11: Stundenplan Die Anwendung zeigt dem Studierenden den Stundenplan für ein Semester basierend auf den gewählten Modulen an. Der Stundenplan zeigt dem Studierenden in einer Wochenübersicht, wann und wo die Veranstaltungen stattfinden. Veranstaltungen, die zur gleichen Zeit stattfinden, werden besonders markiert. Dies soll dem Studierenden helfen, schnell zu erkennen, ob sich gewählte Module überschneiden und er sich für ein anderes Modul entscheiden sollte. Außerdem hilft es dem Studierenden, den Überblick über seine Veranstaltungen zu behalten und seine Zeit besser zu planen.

FA-12: Prüfungsplan Die Anwendung zeigt dem Studierenden den Prüfungsplan für ein Semester basierend auf den gewählten Modulen an. Der Prüfungsplan zeigt dem Studierenden in einer Kalenderübersicht, wann und wo die Prüfungen stattfinden. Prüfungen, die innerhalb von drei Tagen stattfinden, werden mit einem Warnhinweis markiert. Es soll dem Studierenden helfen, schnell zu erkennen, ob er sich auf mehrere Prüfungen gleichzeitig vorbereiten muss und soll helfen, seine Zeit besser zu planen. Außerdem hilft es dem Studierenden, den Überblick über seine Prüfungen zu behalten.

3.2.2 Nicht-funktionale Anforderungen

NFA-1: Anwendung als Webanwendung Die Anwendung ist eine Webanwendung, die in einem Webbrowser ausgeführt wird. Die Anwendung ist auf allen gängigen Webbrowsern lauffähig und benötigt keine Installation auf dem Gerät des Studierenden. Dadurch soll dem Studierenden die Nutzung der Anwendung erleichtert werden, da er keine Software installieren und aktuell halten muss und die Anwendung auf allen Geräten nutzen kann, die einen Webbrowser haben.

NFA-2: Datenschutz Die Anwendung speichert keine sensiblen Daten des Studierenden, wie z.B. Noten. Die Anwendung gibt dem Studierenden die Möglichkeit, seine Daten lokal im Browser zu speichern oder auf dem Server zu speichern. Die Daten werden nur für den Studierenden sichtbar gespeichert und nicht an Dritte weitergegeben. Dadurch soll der Studierende die Kontrolle über seine Daten behalten und seine Privatsphäre geschützt werden.

NFA-3: Wartbarkeit Die Anwendung ist nach der hexagonalen Architektur aufgebaut, sodass die einzelnen Komponenten der Anwendung leicht ausgetauscht oder angepasst werden können. Vor allem die Datenquellen, aus denen die Anwendung ihre Daten (z.B. Module, Prüfungen, Veranstaltungen) bezieht, sind leicht auszutauschen oder anzupassen. Dies ist besonders wichtig, da sich die in der Ist-Analyse

identifizierten Datenquellen stark unterscheiden und sich wahrscheinlich in Zukunft ändern werden. Durch diese Strukturierung kann die Anwendung leicht an neue Anforderungen oder Änderungen angepasst werden, ohne dass der Code der Anwendung stark verändert werden muss. Dies soll die Wartbarkeit und Erweiterbarkeit der Anwendung erhöhen.

NFA-4: Java als Programmiersprache Die Anwendung wird im Rahmen eines bestehenden Teams erstellt und soll auch nach Abschluss der Arbeit weiterentwickelt werden. Da alle aktuellen und voraussichtlich auch zukünftigen Teammitglieder Java als Programmiersprache beherrschen, wird die Anwendung in Java entwickelt. Dadurch wird einerseits die Einarbeitung neuer Teammitglieder erleichtert, da diese bereits mit der Programmiersprache vertraut sind. Andererseits wird die Wartbarkeit der Anwendung sichergestellt, da alle Teammitglieder den Code verstehen und anpassen können.

3.2.3 Administrative Anforderungen

Damit den Studierenden die in den vorherigen Abschnitten beschriebenen Funktionen zur Verfügung stehen, müssen die zugrunde liegenden Daten durch einen Administrator gepflegt werden. Die folgenden Anforderungen beschreiben die dafür notwendigen Funktionen.

Während Semesterzeiten, Studiengänge und Module direkt von den entsprechenden Webseiten der HTW ausgelesen werden können (AA-1 bis AA-3), liegen Prüfungs- und Stundenpläne nur als Datei vor und müssen daher manuell durch einen Administrator importiert werden (AA-4, AA-5).

AA-1: Semesterzeiten verwalten Die Anwendung ruft die Semesterzeiten automatisiert von der Website der HTW ab, sobald ein Administrator dies auslöst.

AA-2: Studiengänge verwalten Ein Administrator kann das Auslesen der Studiengänge aus dem Modulkatalog auslösen. Die Anwendung speichert die ausgelesenen Studiengänge anschließend.

AA-3: Module pro Studiengang verwalten Ein Administrator kann einen Studiengang auswählen und die Anwendung liest automatisch die Liste an Modulen für diesen Studiengang aus dem Modulkatalog aus und speichert sie in der Anwendung.

AA-4: Prüfungsplan importieren Ein Administrator kann für jedes Semester eine Excel-Datei mit dem Prüfungsplan aus dem entsprechenden Moodle-Kurs hochladen, aus der die Anwendung die Prüfungstermine automatisiert extrahiert und in der Anwendung speichert.

AA-5: Stundenplan importieren Ein Administrator kann für jeden Studiengang und für jedes Semester die Stundenplan-Website (Teil des Raumplanungssystems) als HTML-Datei hochladen. Die Anwendung extrahiert automatisch die Veranstaltungstermine und speichert sie in der Anwendung.

AA-6: Module als angeboten markieren Ein Administrator kann für jedes Semester und jeden Studiengang die Module, die in diesem Semester angeboten werden, manuell als „angeboten“ markieren. Dadurch kann der Studierende diese Module in der Anwendung auswählen. Dies ist notwendig, da der Wahlpflichtkatalog erst kurz

Tabelle 3.2: Priorisierung der Anforderungen

ID	Anforderung	Priorität
FA-1	Datenpersistenz	Must have
FA-2	Studiengangwahl	Must have
FA-3	Semesterverwaltung	Must have
FA-4	Modulwahl	Must have
FA-8	Gesamt-ECTS für Semester	Must have
FA-10	Studiumsübersicht	Must have
AA-1	Semesterzeiten verwalten	Must have
AA-2	Studiengänge verwalten	Must have
AA-3	Module pro Studiengang verwalten	Must have
FA-5	Prüfungsstatus eines Moduls verwalten	Should have
FA-6	Abhängigkeitsprüfung	Should have
FA-9	Arbeitsaufwand für Semester	Should have
FA-11	Stundenplan	Should have
FA-12	Prüfungsplan	Should have
AA-4	Prüfungsplan importieren	Should have
AA-5	Stundenplan importieren	Should have
FA-7	Module mit Fehlversuchen hervorheben	Could have

vor Beginn des Semesters veröffentlicht wird und die Anwendung daher nicht automatisch erkennen kann, welche Wahlpflichtmodule in einem Semester angeboten werden.

AA-7: Administratoren verwalten Ein Administrator kann andere Benutzer als Administratoren markieren, sodass diese ebenfalls die administrativen Funktionen der Anwendung nutzen können. Ein Administrator kann außerdem andere Administratoren wieder entmarkieren, sodass diese die administrativen Funktionen nicht mehr nutzen können. Es sollen außerdem lokale Benutzer hinzugefügt werden können, die nicht über das HIZ-Login verfügen. Diese lokalen Benutzer können ebenfalls als Administratoren markiert werden.

3.2.4 Priorisierung der Anforderungen

Da die Zeit zum Entwickeln der Anwendung begrenzt ist, müssen die Anforderungen priorisiert werden. Die Priorisierung erfolgt nach dem MoSCoW-Prinzip, bei dem die Anforderungen in vier Kategorien eingeteilt werden: Must have, Should have, Could have und Won't have. [1] Tabelle 3.2 zeigt die Priorisierung der Anforderungen. Nichtfunktionale Anforderungen werden nicht priorisiert, da sie als Qualitätsanforderungen für die Anwendung als Ganzes gelten und daher nicht einzeln priorisiert werden können.

Die Anforderungen, die in der Tabelle als „Must have" priorisiert wurden, sind die wichtigsten Anforderungen, die unbedingt umgesetzt werden müssen, damit die Anwendung ihren Zweck erfüllt. Die Anforderungen, die als „Should have" priorisiert wurden, sind ebenfalls wichtig, aber nicht unbedingt notwendig für den grundlegenden Betrieb der Anwendung. Die Anforderungen, die als „Could have" priorisiert wurden, sind Nice-to-have-Funktionen, die die Anwendung verbessern können, aber nicht unbedingt notwen-

dig sind. Es gibt keine Anforderungen, die als „Won't have" priorisiert wurden, da alle identifizierten Anforderungen für die Anwendung relevant sind.

Als Must have wurden neben den Basisfunktionen (FA-1 bis FA-3) auch die Modulwahl (FA-4) sowie die darauf aufbauende ECTS-Berechnung (FA-8) und die Studiumsübersicht (FA-10) eingestuft, da diese gemeinsam den Kernprozess der Semesterplanung abbilden, den die Anwendung mindestens abdecken muss, um ihren Zweck zu erfüllen. Sie bieten dem Studierenden zumindest einen Überblick über die angebotenen Module, die Auswahl seiner Module für ein Semester sowie einen Überblick über den Fortschritt seines Studiums und adressieren damit zumindest teilweise die in der Ist-Analyse identifizierten Probleme der Fragmentierung von Informationen und der fehlenden Übersicht. AA-1 bis AA-3 sind als Must have priorisiert, da die Anwendung ohne die zugrunde liegenden Daten nicht funktionsfähig ist. Die Daten müssen von einem Administrator gepflegt werden, damit die Anwendung den Studierenden die notwendigen Informationen bereitstellen kann.

Als Should have gelten die Verwaltung des Prüfungsstatus (FA-5), die Abhängigkeitsprüfung (FA-6), die Berechnung des Arbeitsaufwands (FA-9) sowie Stundenplan (FA-11) und Prüfungsplan (FA-12). Diese Funktionen erweitern den Kernprozess um zusätzliche Übersicht und Komfort, sind für die grundlegende Semesterplanung aber nicht zwingend erforderlich. Sie geben dem Studierenden zusätzliche Informationen und Hilfestellungen, um seine Module besser zu priorisieren und seine Zeit besser zu planen. Vor allem lösen Stundenplan (FA-11) und Prüfungsplan (FA-12) das in der Ist-Analyse gefundene Problem der Formatvielfalt, da Studierende zum Beispiel nicht mehr eine Excel-Datei für den Prüfungsplan öffnen müssen, sondern alle Informationen in der Anwendung gebündelt angezeigt bekommen. Diese tragen außerdem weiter zur Reduzierung der Fragmentierung von Informationen bei, da die Anwendung die Informationen aus den verschiedenen Quellen zusammenführt und dem Studierenden in einer einheitlichen Ansicht präsentiert. Prüfungsstatus-Verwaltung (FA-5), Abhängigkeitsprüfung (FA-6) und Arbeitsaufwandsberechnung (FA-9) unterstützen den Studierenden zusätzlich bei der Priorisierung und Einschätzung seiner Modulwahl, sind für die reine Grundfunktion der Anwendung aber nicht zwingend notwendig. AA-4 und AA-5 sind als Should have priorisiert, da sie die notwendige Datengrundlage für die ebenfalls als Should have eingestuften FA-11 und FA-12 bereitstellen, ohne die diese nicht funktionsfähig wären.

Als Could have wurde nur das Hervorheben von Modulen mit Fehlversuchen (FA-7) eingestuft, da diese Funktion zwar bei der Priorisierung einzelner Module unterstützt, aber keinen Einfluss auf die grundlegende Planungsfähigkeit der Anwendung hat.

3.2.5 Abgrenzung der Anforderungen

Die bisher beschriebenen Anforderungen bilden die Grundlage für die Entwicklung der Anwendung. Es gibt jedoch auch Anforderungen, die nicht in den Umfang dieser Arbeit fallen und daher nicht umgesetzt werden. Diese Anforderungen werden in diesem Abschnitt beschrieben.

NA-1: Beschränkung auf Informatik-Studiengänge Die Anwendung beschränkt sich auf die Informatik-Studiengänge der HTW, also Praktische Informatik, Kommunikationsinformatik und Produktionsinformatik. Unterschiedliche Studiengänge haben unterschiedliche Anforderungen an die Modulwahl und die Berechnung der ECTS-Punkte, eigene Modulnummern und weitere Eigenheiten. Um die Anwendung in der gegebenen Zeit entwickeln zu können, wird die Anwendung im Rahmen dieser

Arbeit auf die Informatik-Studiengänge beschränkt. Da alle drei Studiengänge dieselben zentralen Hochschulsysteme nutzen, wird angenommen, dass die in der Ist-Analyse identifizierten Probleme in vergleichbarer Form auch für Kommunikations- und Produktionsinformatik gelten, auch wenn die Analyse selbst auf Praktischer Informatik basiert.

NA-2: Mindestanzahl an technischen ECTS-Punkten Die Anwendung zeigt dem Studierenden nur die Gesamtanzahl an bereits erworbenen ECTS-Punkten sowie die noch benötigte Anzahl zum erfolgreichen Abschluss des Studiums an. Eine Überprüfung, ob die studiengangsspezifischen Mindestanforderungen an technischen, nicht-technischen oder beispielsweise informatikbezogenen ECTS-Punkten erfüllt sind, findet nicht statt. Dies liegt vor allem daran, dass sich die Anforderungen und Kategorisierungen zwischen den Studiengängen unterscheiden. Es können unterschiedliche Studiengänge unterschiedliche Mindestanforderungen an bestimmte ECTS-Kategorien stellen und Module je nach Studiengang unterschiedlichen Kategorien wie „technisch“, „nicht-technisch“ oder „informatikspezifisch“ zuordnen. Eine einheitliche Kategorisierung der Module ist daher nicht möglich.

NA-3: Keine Verwaltung von Noten Aus Datenschutzgründen werden in der Anwendung keine Noten gespeichert. Die Anwendung speichert nur den Status der Prüfung eines Moduls, also ob der Studierende die Prüfung erfolgreich abgeschlossen hat, die Prüfung nicht bestanden hat oder die Prüfung noch nicht versucht hat.

Zusätzlich ergibt sich aus der Ist-Analyse eine organisatorische Rahmenbedingung, die unabhängig von der Anwendung besteht: Der Wahlpflichtkatalog wird erst kurz vor Beginn des Semesters veröffentlicht. Dies bedeutet, dass die Anwendung nicht in der Lage sein wird, die Wahlpflichtmodule für das aktuelle Semester anzuzeigen, bevor diese veröffentlicht wurden. Dies wird dadurch gelöst, dass Wahlmodule von System-Administratoren manuell als „angeboten“ markiert werden können, sodass der Studierende diese in der Anwendung auswählen kann.

4 Konzeptionierung

Nachdem die Ist-Analyse abgeschlossen wurde und die Anforderungen an das System definiert wurden, kann nun mit der Konzeptionierung des Systems begonnen werden. Zunächst wird die Architektur des Systems entworfen und die einzelnen Komponenten des Systems definiert. Anschließend wird das Datenmodell des Systems erstellt und das System für die Nutzerdatenspeicherung und Authentifizierung konzipiert. Zuletzt werden die Technologien ausgewählt, die für die Umsetzung des Systems verwendet werden sollen.

4.1 Systemarchitektur

Die Anwendung wurde in klar abgegrenzte Module unterteilt: mehrere Module zum Abrufen und Parsen von benötigten Daten wie Stundenplan oder Prüfungsplan sowie ein Modul, welches die eigentliche Anwendung darstellt. Dies fördert die Wiederverwendbarkeit der Module und trennt die eigentliche Anwendung klar von der Logik zum Parsen und Abrufen der Daten, deren Quellen sich häufig ändern können. Abbildung 4.1 zeigt die Module der Anwendung und deren Beziehungen zueinander. Die gerichteten Pfeile zeigen an, dass ein Modul ein anderes Modul als Abhängigkeit referenziert. Hier ist der Excel-Parser zusätzlich als eigenes Modul extrahiert, da er allgemeine Logik zum Parsen von Excel-Dateien enthält, die auch von potenziellen zukünftigen Parser-Modulen verwendet werden kann.

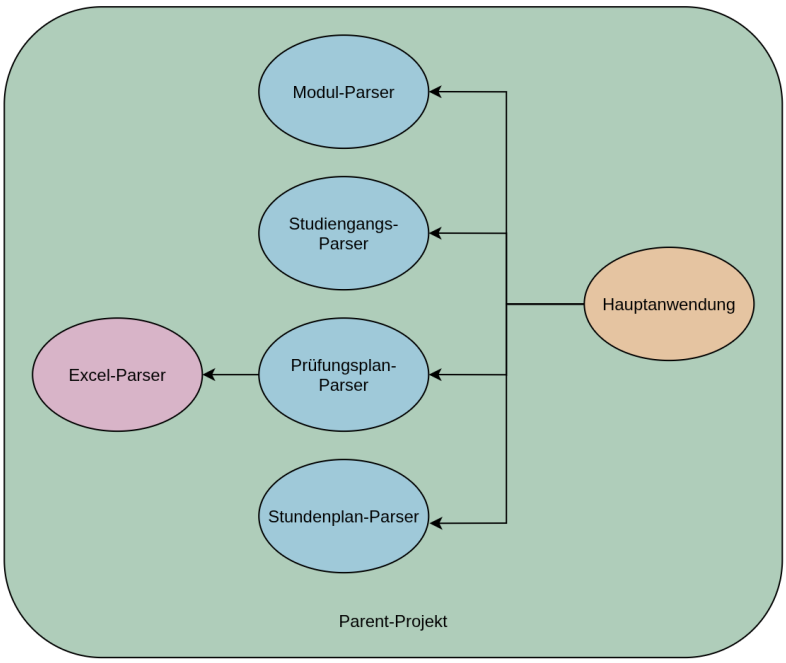


Abbildung 4.1: Übersicht der Module der Anwendung (eigene Darstellung)

Die Module zum Abrufen und Parsen sind alle ähnlich aufgebaut. Sie bestehen aus einer Application Programming Interface (API) und einer Implementierung dieser API. Die API definiert die Methoden, die zum Abrufen und Parsen der Daten benötigt werden und die Struktur der Daten, die zurückgegeben werden. Die Implementierung der API ist für das tatsächliche Abrufen und Parsen zuständig. Die Implementierung kann dabei je nach Bedarf angepasst werden, um die Daten von verschiedenen Quellen abzurufen und zu parsen, ohne die eigentliche Anwendung zu beeinflussen.

Die Anwendung selbst ist wie in NFA-3 gefordert nach der hexagonalen Architektur aufgebaut. Der Anwendungskern besteht aus den Domänenmodellen, welche die Datenstrukturen der Anwendung definieren, und den Anwendungsdiensten, welche die Geschäftslogik der Anwendung implementieren.

Die Anwendungsdienste sind über Ports mit der Außenwelt verbunden. Hier wird zwischen eingehenden und ausgehenden Ports unterschieden. Alistair Cockburn beschreibt diese in [5] als „primary“ und „secondary“ ports. Die eingehenden Ports definieren die Schnittstellen, über die die Anwendung von außen aufgerufen werden kann. Die ausgehenden Ports definieren die Schnittstellen, über die die Anwendung auf externe Systeme zugreifen kann. Die eingehenden Ports werden durch die Anwendungsdienste im Anwendungskern implementiert. Gleichzeitig nutzen die Anwendungsdienste die ausgehenden Ports, um auf externe Systeme zuzugreifen.

Adapter nutzen die Ports, um mit der Anwendung zu kommunizieren. Hier wird auch wieder zwischen zwei Arten unterschieden: treibende und getriebene Adapter. Die treibenden Adapter nutzen die eingehenden Ports, um die Anwendung von außen aufzurufen (z.B. Weboberfläche). Die getriebenen Adapter implementieren die ausgehenden Ports, um die Anwendung mit externen Systemen zu verbinden, auf die die Anwendung selbst zugreift, wie einer Datenbank oder den Parser-Modulen. Die getriebenen Adapter sind dabei für die Implementierung der Schnittstellen zu den externen Systemen zuständig und können je nach Bedarf angepasst werden, um die Anwendung mit verschiedenen externen Systemen zu verbinden.

Abbildung 4.2 zeigt die Systemarchitektur der Anwendung als Diagramm. Orange hinterlegt ist das Gesamtsystem mit den Parser-Modulen und der eigentlichen Anwendung. Die Parser-Module sind als gelbe Rechtecke dargestellt und das Anwendungsmodul ist grün hinterlegt. Ein Benutzer kann über die Benutzeroberfläche, also einen treibenden Adapter (pink hinterlegt), auf die Anwendung zugreifen. Die Benutzeroberfläche nutzt die eingehenden Ports (lila hinterlegt), um die Anwendungsdienste (grau hinterlegt) im Anwendungskern (blau hinterlegt) aufzurufen. Die Anwendungsdienste nutzen die ausgehenden Ports, um auf die getriebenen Adapter zuzugreifen, welche die Schnittstellen zu den externen Systemen implementieren. Die getriebenen Adapter greifen auf die Parser-Module und die Datenbank zu, um die benötigten Daten abzurufen und zu speichern. Die Parser-Module greifen entweder direkt auf externe Systeme zu, um die Daten abzurufen (Parser 1), oder parsen Daten, die vom Benutzer über die Benutzeroberfläche hochgeladen und durch den Anwendungskern an die Parser-Module weitergeleitet werden (Parser 2).

Durch diese Architektur hängt der Anwendungskern also nur von den Port-Interfaces ab und nicht von den Implementierungen der Adapter. Dies ermöglicht eine hohe Flexibilität und Testbarkeit der Anwendung, da die Implementierungen der Adapter leicht ausgetauscht werden können, ohne den Anwendungskern zu beeinflussen.

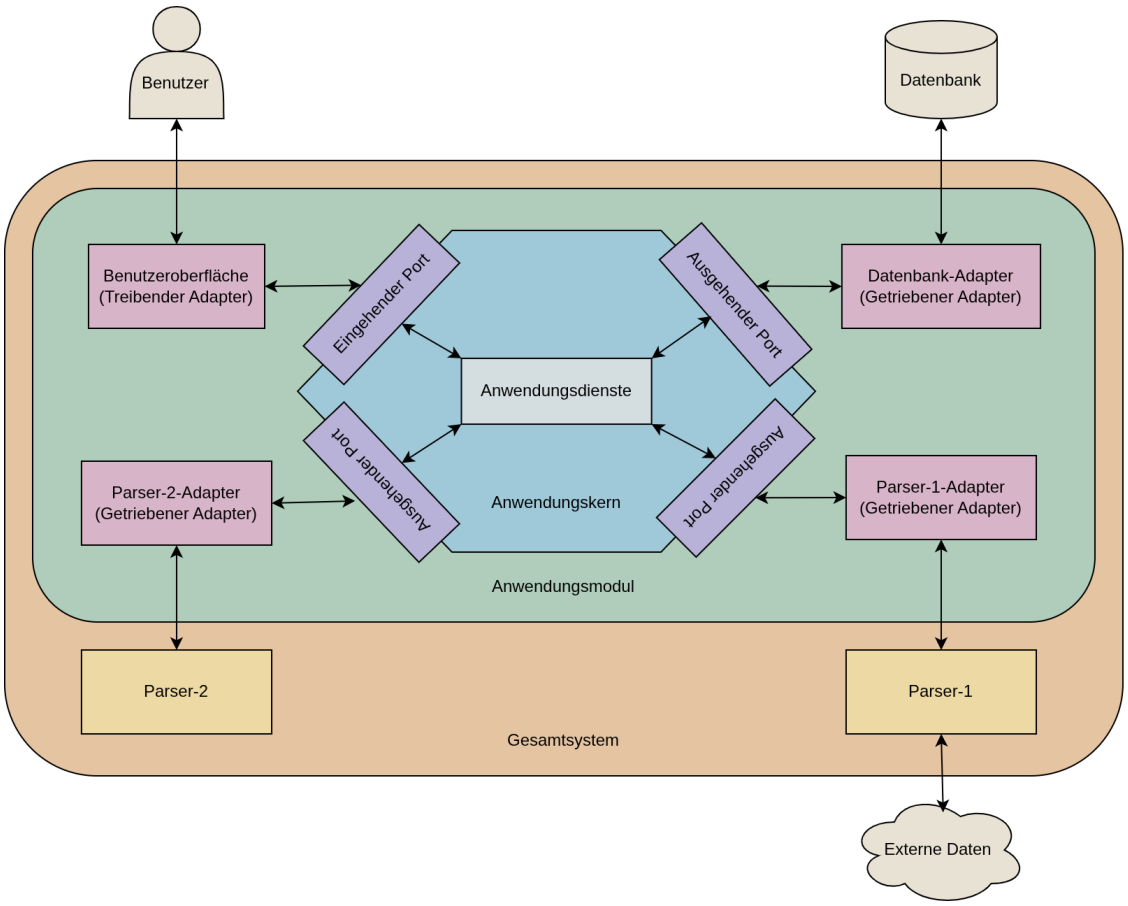


Abbildung 4.2: Systemarchitektur der Anwendung (eigene Darstellung)

4.2 Datenmodell

Im Folgenden wird das Datenmodell der Anwendung vorgestellt. Das Datenmodell beschreibt die Struktur der Daten, die in der Anwendung verwendet werden, und die Beziehungen zwischen den verschiedenen Entitäten. Das Datenmodell basiert auf den im Soll-Konzept definierten Anforderungen.

4.2.1 Entitäten

Da Studierende einen Studiengang auswählen können, müssen Studiengänge in der Datenbank gespeichert werden. Da ein Studiengang mehrere Variationen mit unterschiedlichen Studienordnungen haben kann, wird in der Datenbank pro Variation ein eigener Datensatz in der Studiengang-Entität angelegt. Jede dieser Entitäten speichert neben Metadaten den Studiengangscode, der den Studiengang im Hochschulsystem identifiziert, sowie den von der Moduldatenbank je Variation eindeutig vergebenen „b-Key“.

Im Soll-Konzept wird in FA-3 gefordert, dass Studierende Semester anlegen können, die einer Semesterzeit zugeordnet sind. Daher wird in der Datenbank eine Entität für Semesterzeiten benötigt. Ein Semester ist eindeutig durch eine von der Anwendung vergebene ID identifiziert und speichert Metadaten wie Start- und Enddatum.

Studierende sollen außerdem in der Lage sein, die Module, die sie in einem Semester

belegen, zu verwalten. Daher wird in der Datenbank eine Entität für Module benötigt. Ein Modul ist ebenfalls eindeutig durch eine von der Anwendung vergebene ID identifiziert. Ein Modul kann zwar zu mehreren Studiengängen gehören, allerdings könnten sich die Metadaten eines Moduls je nach Studiengang unterscheiden. Daher ist für ein Modul in der Datenbank die Kombination aus b-Key des Studiengangs und einem sogenannten „c-Key“, welcher von der Moduldatenbank für jedes Modul vergeben wird, einzigartig. Es werden außerdem die Metadaten des Moduls gespeichert, wie z.B. der Name, die Anzahl der ECTS-Punkte und die Dozenten. Hier wird auch gespeichert, ob ein Modul aktuell angeboten wird oder nicht.

Laut FA-11 sollen Studierende auch ihren Stundenplan für ein Semester sehen können. Daher wird eine Entität für Veranstaltungen benötigt. Eine Veranstaltung findet immer in einer bestimmten Semesterzeit statt und ist eindeutig durch eine von der Anwendung vergebene ID identifiziert. Eine Veranstaltung speichert außerdem Metadaten wie das zugehörige Modul, den Wochentag, die Uhrzeit, den Raum und die Dozenten.

Neben dem Stundenplan soll laut FA-12 auch der Prüfungsplan für ein Semester angezeigt werden. Es wird daher eine Entität für Prüfungen benötigt. Eine Prüfung findet ebenfalls immer in einer bestimmten Semesterzeit statt und ist ebenfalls eindeutig durch eine von der Anwendung vergebene ID identifiziert. Eine Prüfung speichert außerdem Metadaten wie das zugehörige Modul, das Datum, die Uhrzeit, den Raum und die Dozenten.

Da die Anwendung es Studierenden auch erlauben soll, sich über das HIZ-Login anzumelden, wird eine Entität für Benutzer benötigt. Dieser Benutzer ist eindeutig durch eine von der Anwendung vergebene ID identifiziert und speichert Daten vom HIZ-Login, wie z.B. die Benutzerkennung, Name und die E-Mail-Adresse. Es können hier allerdings auch lokale Benutzer angelegt werden, die sich nicht über das HIZ-Login anmelden. Diese lokalen Benutzer speichern dieselben Daten und zusätzlich ein gehashtes Passwort, welches für die Authentifizierung verwendet wird, und ein Flag, das den Benutzer als lokal kennzeichnet. Hier werden auch die Daten über das Studium des Benutzers als JSON-Objekt gespeichert. Mehr dazu im Abschnitt 4.3. Ein Benutzer kann außerdem als Administrator markiert werden, wodurch er Zugriff auf die in den administrativen Anforderungen definierten Funktionen erhält.

4.2.2 Beziehungen

Abbildung 4.3 zeigt das Datenmodell der Anwendung. Um die Übersicht zu bewahren, werden hier nur die wichtigsten Attribute der Entitäten dargestellt. Wie in der Abbildung zu sehen ist, gibt es kaum Beziehungen zwischen den Entitäten. Die einzigen Beziehungen sind N:1-Beziehungen zwischen den Entitäten Prüfung und Veranstaltung zur Entität Semesterzeit. Alle anderen Entitäten sind nicht auf Datenbankebene miteinander verbunden. Der Grund hierfür ist, dass viele Studiengänge und Module ihre Eigenheiten haben und es daher schwierig ist, ein allgemeingültiges Datenmodell zu erstellen, das alle Studiengänge und Module abbildet. Daher werden Regeln wie „Ein Modul kann nur zu einem Studiengang gehören“ nicht auf Datenbankebene, sondern in der Anwendung implementiert. Dadurch können einfacher Ausnahmen und Sonderfälle behandelt werden, die in den Studiengängen und Modulen auftreten können.

Zusätzlich speichert die Entität Benutzer die Daten über das Studium des Benutzers als JSON-Objekt. Dies ermöglicht es, die Daten flexibel zu speichern und zu erweitern, ohne das Datenmodell ändern zu müssen. Die Daten werden in der Anwendung validiert,

um sicherzustellen, dass sie den Anforderungen entsprechen. Dies wird im Abschnitt 4.3 näher erläutert.

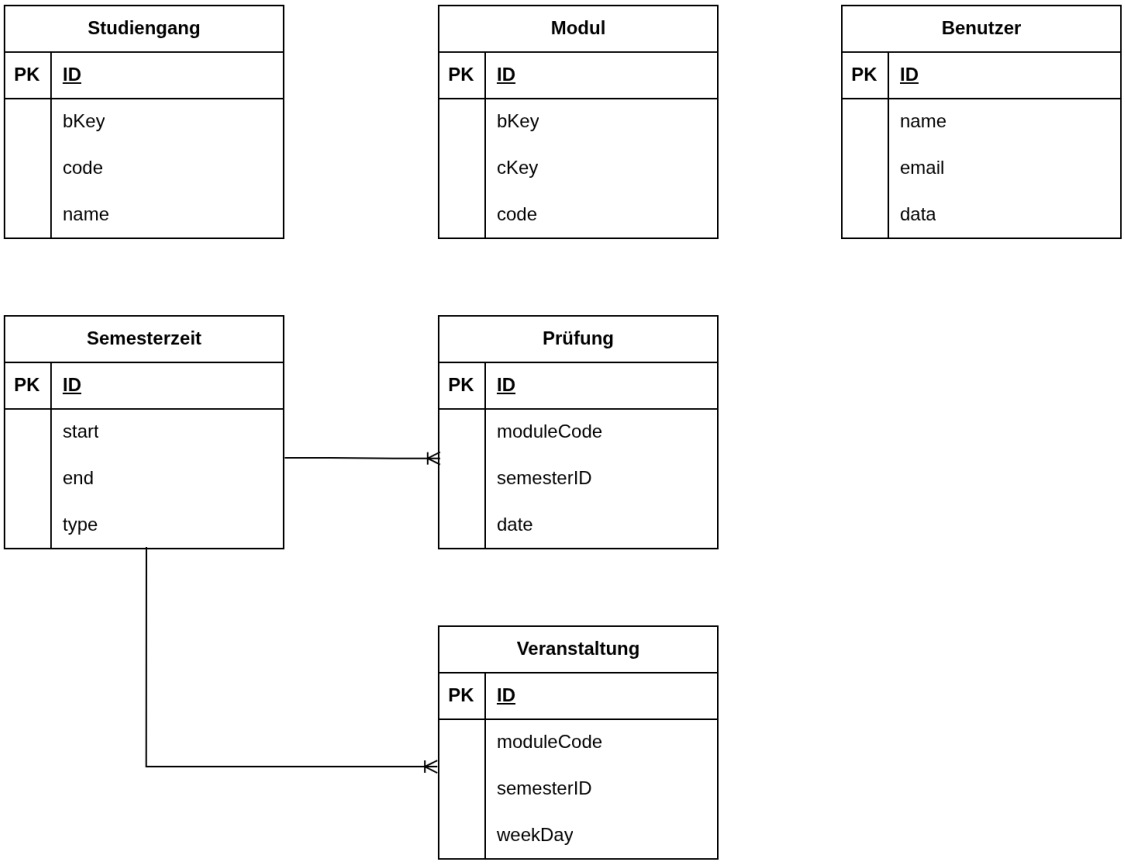


Abbildung 4.3: Datenmodell der Anwendung (eigene Darstellung)

4.3 Nutzerdatenspeicherung und Authentifizierung

Im folgenden Abschnitt wird die Speicherung der Nutzerdaten in einem strukturierten Format und die Authentifizierung der Benutzer in der Anwendung beschrieben.

4.3.1 Nutzerdatenspeicherung

In FA-1 wird gefordert, dass Studierende ihre Daten, wie z.B. den gewählten Studiengang oder die belegten Module, auf dem Server speichern oder als Datei exportieren und wieder importieren können. Um nicht zwei verschiedene Datenmodelle für die Speicherung der Daten auf dem Server und in der Datei zu haben, wird das gleiche Datenmodell für beide Speicherorte verwendet. Die Daten werden als Objekte in der Anwendung gespeichert und können, wie in Abschnitt 4.2 erläutert, in serialisierter Form in der Datenbank in der Benutzer-Entität oder in einer Datei gespeichert werden.

In der Anwendung werden die Daten nach einem versionierten Schema erstellt. Wenn die Daten gespeichert oder exportiert werden, wird die Version des Schemas neben den Daten selbst gespeichert. Wenn die Daten wieder importiert oder geladen werden, wird die Version des Schemas überprüft und die Daten werden validiert. Wenn die Version des

Schemas nicht mit der Version der Anwendung übereinstimmt, werden die Daten nach einem Migrationsschema in das aktuelle Schema migriert. Hierbei wird inkrementell vorgegangen. Wenn zum Beispiel die zu importierende Datei Version 1 des Schemas enthält und die Anwendung Version 3 des Schemas verwendet, wird zuerst Migration 1->2 und dann Migration 2->3 durchgeführt. Die Migrationsschemata werden in der Anwendung definiert und können bei Bedarf erweitert werden. Dies ermöglicht es, die Datenstruktur in der Anwendung zu ändern, ohne dass die Daten der Benutzer verloren gehen. Durch die inkrementelle Migration wird im Gegensatz zu einer direkten Migration der Wartungsaufwand reduziert, da für jede neue Version des Schemas nur die Migration von der vorherigen Version zur neuen Version implementiert werden muss und nicht von jeder möglichen Version zur neuen Version. Abbildung 4.4 zeigt diesen Migrationsablauf beim Import von Nutzerdaten als Flowchart.

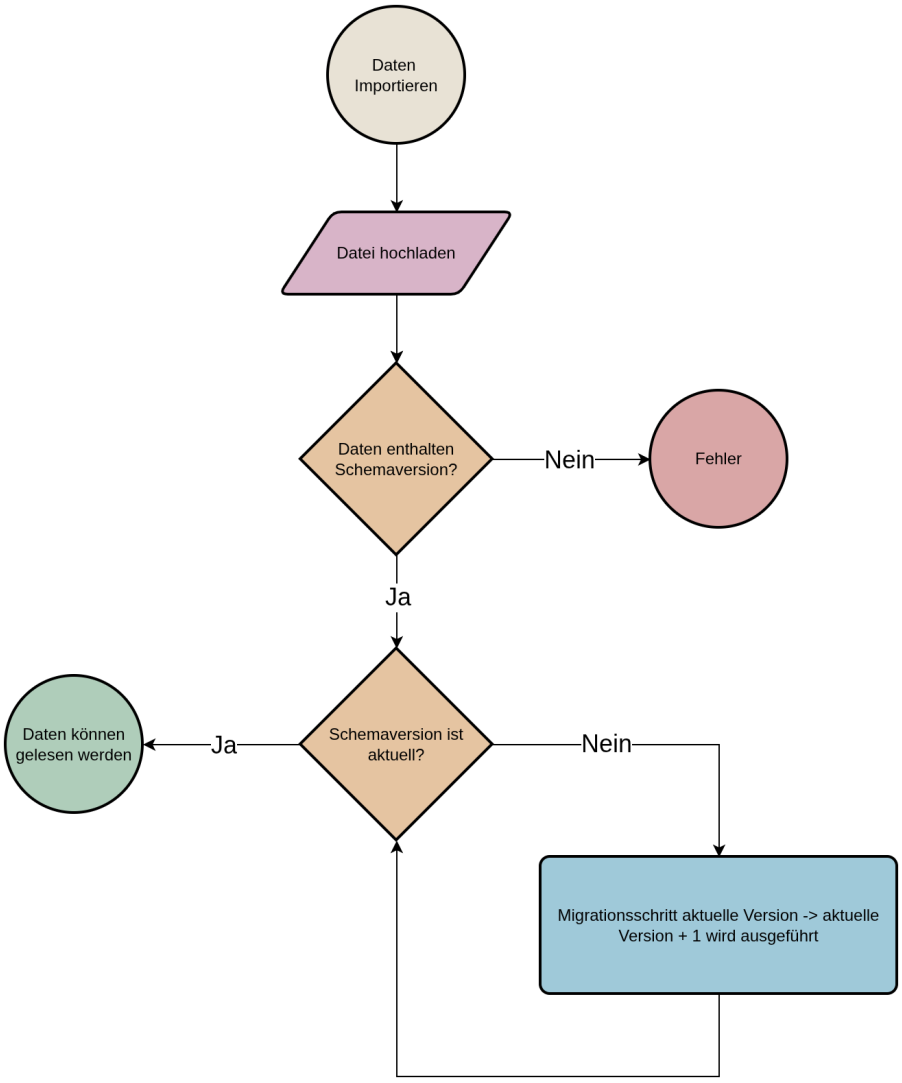


Abbildung 4.4: Migrationsablauf von Nutzerdaten (eigene Darstellung)

Es bedarf außerdem eines Systems, um Konflikte zu erkennen. Stellen wir uns folgendes Szenario vor: Ein Benutzer hatte in vorherigen Sessions seine Daten mithilfe seines HIZ-Logins auf dem Server gespeichert. In einer neuen Session möchte der Benutzer seine Daten wieder laden. Allerdings hat der Benutzer in der Zwischenzeit seine Daten lokal geändert. Wenn er sich anmeldet, muss er entscheiden, welche Daten er behalten möchte.

Die lokalen Daten oder die Daten auf dem Server dürfen nicht einfach überschrieben werden. Daher wird ein Konflikt erkannt und der Benutzer muss entscheiden, welche Daten er behalten möchte. Die Anwendung bietet dem Benutzer die Möglichkeit, die Daten zu vergleichen und die Änderungen zu übernehmen oder zu verwerfen.

4.3.2 Authentifizierung

Da in FA-1 gefordert wird, dass sich Studierende über das HIZ-Login anmelden können und in AA-7 gefordert wird, dass sich Administratoren entweder über das HIZ-Login oder über ein lokales Login anmelden können, wird ein hybrides System implementiert. Um die Anmeldung möglichst simpel zu gestalten, wird eine Anmelde-Seite implementiert, in der Benutzer ihren Benutzernamen und ihr Passwort eingeben können. Die Anwendung überprüft erst, ob der Benutzername zu einem lokalen Benutzer in der Datenbank gehört, also einem, bei dem, wie in Abschnitt 4.2 beschrieben, das local-Flag gesetzt ist. Wenn ja, wird das Passwort überprüft und der Benutzer angemeldet. Wenn nein, wird der Benutzername an das HIZ-Login weitergeleitet und die Anmeldung darüber durchgeführt. Wenn die Anmeldung erfolgreich war, wird in der Datenbank überprüft, ob der Benutzer bereits existiert. Wenn ja, wird der Benutzer angemeldet. Wenn nein, wird der Benutzer in der Datenbank angelegt und anschließend angemeldet. Sollte der Benutzername weder zu einem lokalen Benutzer noch zu einem HIZ-Benutzer gehören, wird eine Fehlermeldung angezeigt. Abbildung 4.5 zeigt genau diesen Authentifizierungsablauf der Anwendung als Flowchart.

4.4 Auswahl der Technologien

Im Folgenden werden die für die Umsetzung der Anwendung ausgewählten Technologien vorgestellt. Die Auswahl der Technologien erfolgte auf Basis der im Soll-Konzept formulierten Anforderungen sowie technologiespezifischer Kriterien, die in den jeweiligen Unterkapiteln erläutert werden.

4.4.1 Backend

Da im Soll-Konzept die Anforderung NFA-4 formuliert ist, dass die Anwendung in Java entwickelt werden soll, standen als Backend-Frameworks Jakarta EE und das Spring Framework mit Spring-Boot zur Auswahl. Beide Frameworks sind in der Java-Entwicklung weit verbreitet und bieten umfangreiche Funktionalitäten für die Erstellung von Webanwendungen. Allerdings ist Spring leichtgewichtiger und nicht an einen Application Server gebunden. Durch Spring-Boot können Anwendungen zudem als eigenständige JAR-Dateien bereitgestellt werden, was die Bereitstellung und den Betrieb vereinfacht. Daher fiel die Entscheidung auf Spring und Spring-Boot.

Als Build-Tool standen Maven und Gradle zur Auswahl. Beide Build-Tools sind in der Java-Entwicklung weit verbreitet und bieten eine gute Integration mit Spring-Boot. [23, 25] Maven wurde ausgewählt, da der Autor dieser Arbeit bereits Erfahrung mit Maven hat.

JSpecify wurde verwendet, da es eine standardisierte Möglichkeit bietet, Nullability-Annotationen in Java zu verwenden und auch von Spring-Boot seit Version 4 unterstützt

wird. [6] Durch die Verwendung von Nullability-Annotationen können potenzielle NullPointerExceptions frühzeitig erkannt und vermieden werden, was die Stabilität und Zuverlässigkeit der Anwendung erhöht. Da es keine anderen von Spring Boot unterstützten, standardisierten Möglichkeiten für Nullability-Annotationen in Java gibt, wurde JSpecify für diese Arbeit ausgewählt.

4.4.2 UI-Framework

Für die Entwicklung der Benutzeroberfläche standen folgende UI-Frameworks zur Auswahl:

- **Vaadin:** Vaadin ist ein Java-basiertes Framework, das die Entwicklung von Webanwendungen mit einer serverseitigen Architektur ermöglicht. Es bietet eine Vielzahl von UI-Komponenten und ermöglicht die Erstellung von responsiven Benutzeroberflächen. Die Benutzeroberfläche wird dabei direkt in Java-Klassen definiert, was die Entwicklung für Java-Entwickler erleichtert. [27]
- **React:** React ist ein weit verbreitetes JavaScript-Framework, das von Meta entwickelt wird. Es ermöglicht die Erstellung von wiederverwendbaren UI-Komponenten. React-Seiten werden im Gegensatz zu Vaadin auf dem Client gerendert. [19]
- **JSF:** JSF ist ein serverseitiges Java-Framework, das die Entwicklung von Webanwendungen erleichtert. Es basiert ebenfalls auf wiederverwendbaren UI-Komponenten, die in XML-Dateien definiert werden und auf dem Server gerendert werden. [15]

React wurde in Betracht gezogen, da es eine moderne und weit verbreitete Lösung für die Entwicklung von Benutzeroberflächen darstellt, wurde jedoch wegen NFA-4 nicht ausgewählt. Es wurde die Entscheidung getroffen, JSF als UI-Framework zu verwenden, da bei Vaadin die Benutzeroberfläche direkt in Java-Code geschrieben wird, was die Trennung von Logik und Darstellung erschwert. JSF hingegen ermöglicht eine klare Trennung von Logik und Darstellung durch die Verwendung von XML-Dateien für die Definition der Benutzeroberfläche. Dies erleichtert die Wartung und Weiterentwicklung der Anwendung. Um JSF in Spring-Boot zu integrieren, wurde die JoinFaces-Bibliothek verwendet.

Um die Benutzeroberfläche ansprechend zu gestalten, wurde das PrimeFaces-UI-Framework ausgewählt. PrimeFaces bietet eine Vielzahl von vorgefertigten UI-Komponenten, die auf JSF basieren und eine moderne und ansprechende Benutzeroberfläche ermöglichen. PrimeFaces wurde wegen seiner großen Zahl an Komponenten, dem guten Design und der hervorragenden Dokumentation ausgewählt.

4.4.3 Datenbank

Für die Datenbank waren die Kompatibilität mit Spring, die Unterstützung von SQL, ein relationales Datenmodell sowie die Möglichkeit zur Speicherung von JSON-Daten relevant. Diese Kriterien werden sowohl von PostgreSQL als auch von MySQL erfüllt, da beide Systeme relationale, SQL-basierte Datenbanken sind, nahtlos über Spring Data JPA angebunden werden können und native Unterstützung für JSON-Datentypen bieten. Da beide Systeme die Anforderungen erfüllen, fiel die Entscheidung aufgrund der weiten Verbreitung, der guten Dokumentation und der Erfahrung im Umgang mit PostgreSQL auf dieses System.

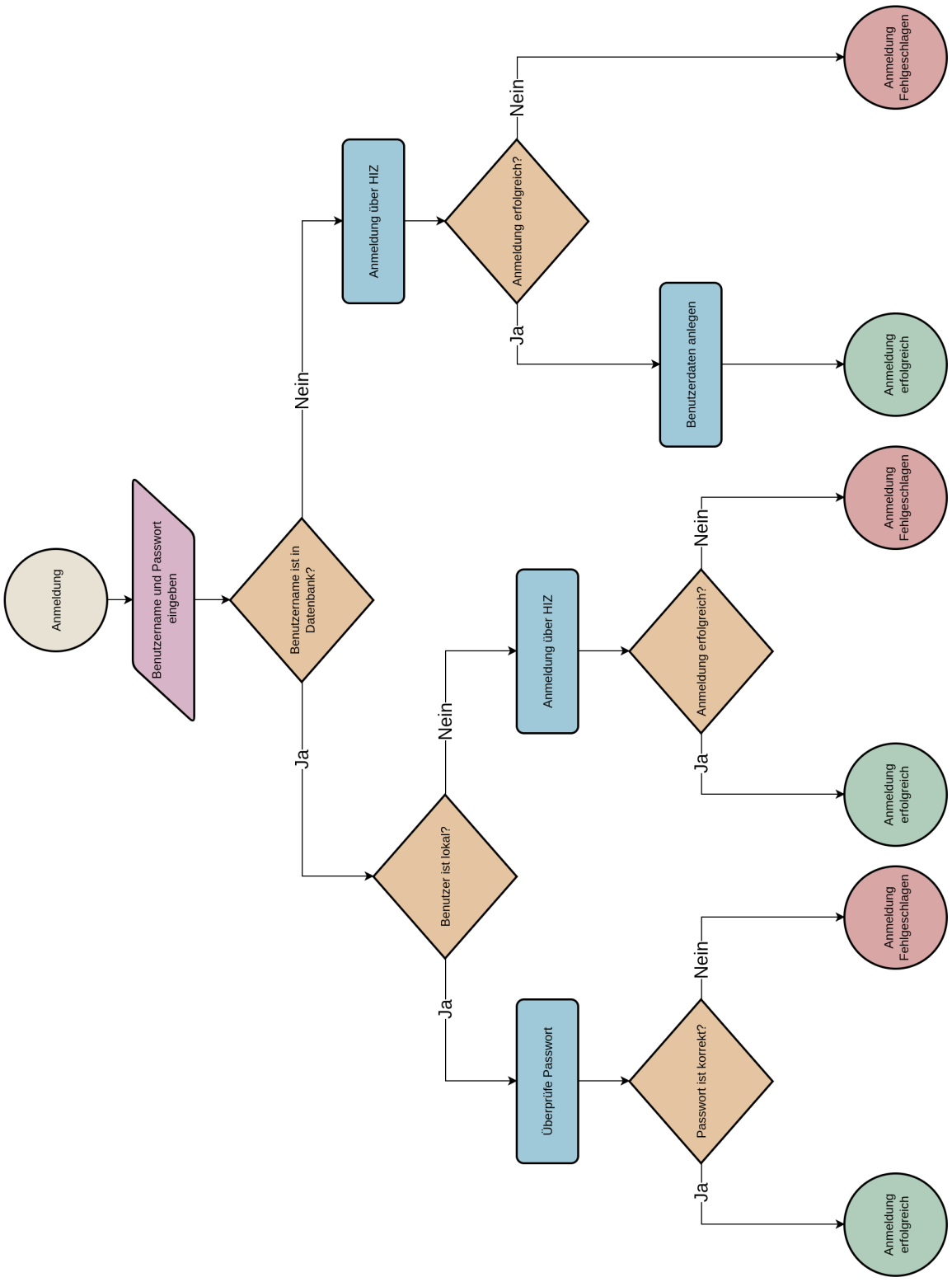


Abbildung 4.5: Authentifizierungsablauf der Anwendung (eigene Darstellung)

5 Implementierung

Nachdem im vorherigen Kapitel die Anwendung konzipiert wurde, wird in diesem Kapitel die tatsächliche Umsetzung der Anwendung beschrieben. Zuerst wird beschrieben, wie mit Hilfe von JSpecify Null-Safety in der Anwendung gewährleistet wird. Danach wird die Umsetzung der Architektur beschrieben, gefolgt von der Umsetzung des Datenmodells. Anschließend wird die Migration von Benutzerdaten beschrieben und abschließend wird die Implementierung von User Sessions und Datenspeicherung erläutert.

5.1 Null-Safety durch JSpecify

Um die im Soll-Konzept geforderte Wartbarkeit zu gewährleisten, wurden JSpecify-Annotationen in der Anwendung verwendet. Zur Sicherstellung der Null-Safety in der gesamten Anwendung befindet sich in jedem Paket eine `package-info.java`-Datei, in der die JSpecify-Annotationen für das gesamte Paket definiert werden. Listing 5.1 ist ein Beispiel für eine solche `package-info.java`-Datei. Hier wird das gesamte Paket als `@NullMarked` markiert, was bedeutet, dass alle Referenzen in diesem Paket standardmäßig nicht null sein dürfen. Durch die statische Code-Analyse von IntelliJ IDEA wird eine Warnung für alle Referenzen, die null sein können, aber nicht als solche markiert sind, angezeigt.

Listing 5.1: package-info.java

```
@NullMarked
package de.htwsaar.stl.studiumsplaner.app.application.port.in;

import org.jspecify.annotations.NullMarked;
```

5.2 Umsetzung der Architektur

5.2.1 Maven-Projektstruktur

Um die in Abschnitt 4.1 beschriebene Architektur umzusetzen, wird ein Parent-Maven-Projekt erstellt, welches die gesamte Anwendung enthält. Dieses Parent-Projekt wird in mehreren Modulen organisiert. Die Hauptanwendung wird in einem eigenen Modul implementiert und die Parser jeweils in eigenen, separaten Modulen. Die Parser-Module werden als Abhängigkeiten in der Hauptanwendung eingebunden, um die benötigten Daten abzurufen und zu parsen.

Listing 5.2: Ausschnitt aus der pom.xml des Parent Projekts

```
<project xmlns="http://maven.apache.org/POM/4.0.0">
    <modelVersion>4.0.0</modelVersion>
```

```
    <groupId>de.htwsaar.stl.studiumsplaner</groupId>
    <artifactId>studiumsplaner-parent</artifactId>
    <version>1.0.0</version>

    <packaging>pom</packaging>

    <modules>
        <module>exam-parser</module>
        <module>semester-parser</module>
        <module>moduldb-parser</module>
        <module>timetable-parser</module>
        <module>excel-parser-core</module>
        <module>studiumsplaner-app</module>
        <module>courses-parser</module>
    </modules>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.jspecify</groupId>
                <artifactId>jspecify</artifactId>
                <version>1.0.0</version>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>${spring-boot.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
            <dependency>
                <groupId>org.joinfaces</groupId>
                <artifactId>joinfaces-platform</artifactId>
                <version>${joinfaces.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <!-- ... -->

</project>
```

Listing 5.2 zeigt einen Ausschnitt aus der `pom.xml` des Parent-Projekts, in dem die enthaltenen Module spezifiziert sind. Hier werden erst die Metadaten des Parent-Projekts wie `groupId`, `artifactId` und `version` definiert. Anschließend werden die enthaltenen Module in der `<modules>`-Sektion aufgelistet. Jedes Modul ist ein eigenständiges Maven-Projekt, das innerhalb des Parent-Projekts organisiert ist. Die Hauptanwendung ist in dem Modul `studiumsplaner-app` implementiert und die Parser-Module sind in den Modulen `exam-parser`, `semester-parser`, `moduldb-parser`, `timetable-parser` und `courses-parser` implementiert. `excel-parser-core` ist eine gemeinsame Bibliothek, die von den Parser-Modulen verwendet wird, um Excel-Dateien zu parsen. In der `dependencyManagement`-Sektion werden die Abhängigkeiten definiert, die von allen Modulen verwendet werden. Hier werden die Versionen der Abhängigkeiten zentral verwaltet, um Konsistenz im ge-

samten Projekt zu gewährleisten.

Listing 5.3: pom.xml des Semesterzeiten-Parser-Moduls

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
            http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>de.htwsaar.stl.studiumsplaner</groupId>
        <artifactId>studiumsplaner-parent</artifactId>
        <version>1.0.0</version>
    </parent>

    <artifactId>semester-parser</artifactId>

    <properties>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.
           sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.jsoup</groupId>
            <artifactId>jsoup</artifactId>
            <version>1.22.2</version>
        </dependency>
        <dependency>
            <groupId>org.jspecify</groupId>
            <artifactId>jspecify</artifactId>
        </dependency>
    </dependencies>
</project>
```

Listing 5.3 zeigt die `pom.xml` des Semesterzeiten-Parser-Moduls. Hier wird das Parent-Projekt in der `<parent>`-Sektion referenziert, um die gemeinsamen Abhängigkeiten und Konfigurationen zu erben. Das `artifactId` des Moduls wird auf `semester-parser` gesetzt. In der `<dependencies>`-Sektion werden die spezifischen Abhängigkeiten für dieses Modul definiert, wie z.B. die Bibliothek `jsoup`, die zum Parsen von HTML-Dokumenten verwendet wird. JSpecify wird ebenfalls als Abhängigkeit hinzugefügt, um die Verwendung von Null-Sicherheit in der Anwendung zu unterstützen. Es wird allerdings keine Version angegeben, da die Version bereits im Parent-Projekt definiert ist und von diesem geerbt wird.

Die `pom.xml` der Hauptanwendung ist ähnlich aufgebaut. Es wird auch das Parent-Projekt referenziert und die spezifischen Abhängigkeiten für die Hauptanwendung werden in der `<dependencies>`-Sektion definiert. Hier werden auch die Parser-Module als Abhängigkeiten mit der `groupId de.htwsaar.stl.studiumsplaner` und den entsprechenden `artifactIds` referenziert, um diese in der Hauptanwendung zu verwenden. Hier werden außerdem die Spring-Boot-Starter-Abhängigkeiten hinzugefügt, um die Spring-Boot-

Funktionalitäten in der Anwendung zu nutzen. Eine Besonderheit in dieser `pom.xml` ist die Verwendung eines zusätzlichen Repositories in der `<repositories>`-Sektion. Dieses muss konfiguriert werden, da die `prime-lucide`-Abhängigkeit nicht im Maven Central Repository verfügbar ist. Prime-Lucide ist eine Bibliothek, die es ermöglicht, die Lucide-Icons, eine große Bibliothek von Icons, in der Benutzeroberfläche der Anwendung zu verwenden. Eine weitere Besonderheit ist die Verwendung des `spring-boot-maven-plugin` in der `<build>`-Sektion. Dieses Plugin wird verwendet, um die Anwendung als ausführbares JAR-Paket zu erstellen, das alle Abhängigkeiten enthält und direkt ausgeführt werden kann. Die vollständige pom.xml der Hauptanwendung ist in Listing A.1 im Anhang dargestellt.

5.2.2 Struktur der Parser-Module

Die Parser-Module sind wie in der Konzeptionierung beschrieben in eine API und eine Implementierung unterteilt. Dies ist durch zwei separate Pakete innerhalb des Moduls umgesetzt. Das `api`-Paket enthält die Schnittstellen, die von der Hauptanwendung verwendet werden, um die Parser-Funktionalitäten zu nutzen. Dies ist meistens durch ein Interface mit einer `parse`-Methode umgesetzt. Das `api`-Paket enthält außerdem die benötigten Datenklassen, die von der `parse`-Methode zurückgegeben werden.

Listing 5.4: Schnittstelle des Moduldb-Parser-Moduls

```
public interface ModuldbParser {
    List<ParsedModule> parseModules(String bKey) throws
        ModuldbParseException;
}
```

Das `impl`-Paket enthält die konkrete Implementierung der Schnittstellen und die Logik zum Parsen der Daten.

5.2.3 Struktur der Hauptanwendung

Die hexagonale Architektur der Hauptanwendung zeigt sich durch die Paketstruktur der Anwendung. Die Hauptanwendung ist in mehrere Pakete unterteilt, die die verschiedenen Teile der Anwendung repräsentieren. Abbildung 5.1 zeigt die Paketstruktur der Hauptanwendung. Um die Übersicht zu wahren, sind nur die wichtigsten Pakete dargestellt.

Die Pakete `domain.model` und `application` repräsentieren den Anwendungskern. `domain.model` enthält die Domänenmodelle und `application` die Anwendungsdienste. `application.port` enthält die Schnittstellen der Anwendung. `application.port.in` enthält die eingehenden Ports, die von den treibenden Adaptern verwendet werden, um die Anwendungsdienste aufzurufen. `application.port.out` enthält die ausgehenden Ports, die von den Anwendungsdiensten verwendet werden, um auf externe Systeme zuzugreifen. `application.service` enthält die Implementierungen der Anwendungsdienste, die die Geschäftslogik der Anwendung implementieren.

Das `adapter`-Paket enthält die Adapter, die die Schnittstellen zu den externen Systemen implementieren. `adapter.in` enthält die treibenden Adapter, die die Anwendung von außen aufrufen. `adapter.in.view` enthält die Adapter, die die Benutzeroberfläche der Anwendung implementieren. `adapter.out` enthält die getriebenen Adapter, die die

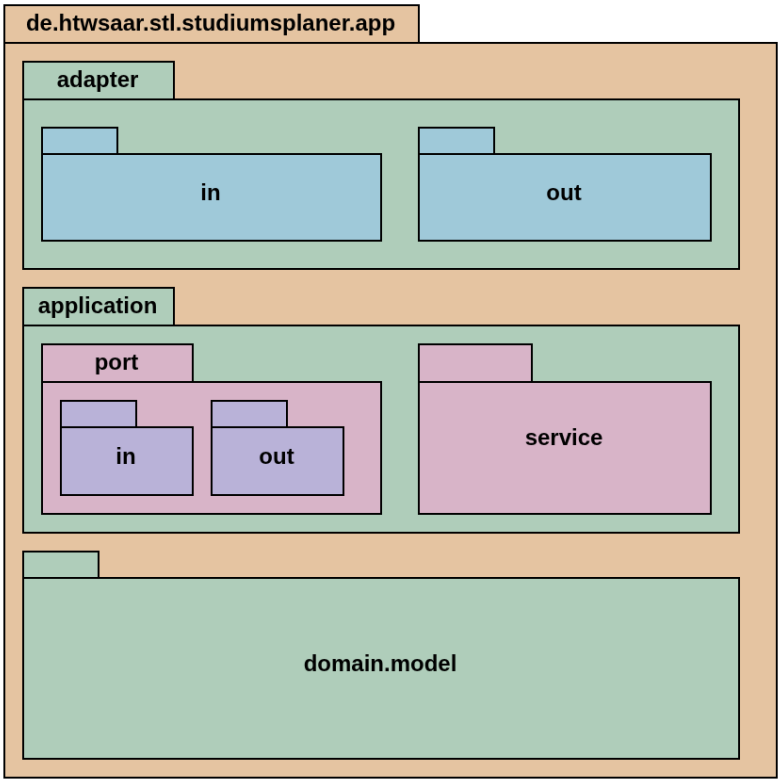


Abbildung 5.1: Paketstruktur der Hauptanwendung (eigene Darstellung)

Schnittstellen zu den externen Systemen implementieren, auf die die Anwendung zugreift. Diese implementieren die ausgehenden Ports der Anwendung. `adap-ter.out.database` enthält die Adapter, die die Schnittstellen zur Datenbank implementieren. Die Adapter für die Parser-Module liegen direkt im `adapter.out`-Paket.

Listing 5.5 und Listing 5.6 zeigen ein Beispiel für einen Anwendungsdienst, der die Logik zum Parsen von Modulen eines Studiengangs implementiert. Listing 5.5 zeigt die Schnittstelle des Anwendungsdienstes, der als eingehender Port dient. Listing 5.6 zeigt die Implementierung des Anwendungsdienstes, der die Logik zum Abrufen und Parsen der Module implementiert. Dieser nutzt die ausgehenden Ports `ModuleSource` und `ModuleRepository`, um die Module von der externen Quelle abzurufen und in der Datenbank zu speichern und enthält die Logik, um zu prüfen, ob ein Modul bereits in der Datenbank existiert, bevor es gespeichert wird.

Listing 5.5: ModulesParseUseCase

```
package de.htwsaar.stl.studiumsplaner.app.application.port.in;

public interface ModulesParseUseCase {
    void execute(String courseBKey);
}
```

Listing 5.6: ModulesParseService

```
package de.htwsaar.stl.studiumsplaner.app.application.service;

import ...
```

```
@Service
public class ModulesParseService implements ModulesParseUseCase {
    private final ModuleSource source;
    private final ModuleRepository repository;

    public ModulesParseService(ModuleSource source,
        ModuleRepository repository) {
        this.source = source;
        this.repository = repository;
    }

    @Override
    public void execute(String courseBKey) {
        source.fetchModules(courseBKey)
                .stream()
                .filter(module -> !repository.existsByBKeyAndCKey(
                        module.getbKey(),
                        module.getcKey()
                ))
                .forEach(repository::save);
    }
}
```

Die ausgehenden Ports werden mithilfe von Dependency Injection in den Anwendungsdienst injiziert. Dies ist möglich, da die Adapter, welche die Schnittstellen zu den externen Systemen implementieren, als Spring Beans konfiguriert sind. Spring erkennt die Implementierungen der Ports und injiziert diese automatisch in den Anwendungsdienst. Dies ermöglicht eine lose Kopplung zwischen dem Anwendungskern und den Adaptern, da der Anwendungskern nur von den Schnittstellen der Ports abhängt und nicht von den Implementierungen der Adapter.

Der Anwendungsdienst ist außerdem mit der `@Service`-Annotation von Spring annotiert, um ihn als Spring Bean zu konfigurieren. Dies ermöglicht es Spring, den Anwendungsdienst zu verwalten und ihn bei Bedarf, beispielsweise bei einem Aufruf durch einen treibenden Adapter, zu instanziieren und zu verwalten.

5.3 Umsetzung des Datenmodells

Das in der Konzeptionierung definierte Datenmodell wurde in der Implementierung mithilfe von JPA umgesetzt. Die Entitäten wurden als Java-Klassen implementiert, die mit den entsprechenden JPA-Annotationen versehen sind, um die Beziehung zwischen den Entitäten und der Datenbank zu definieren. Listing 5.7 zeigt beispielhaft einen Ausschnitt der `ExamJpaEntity`-Klasse, die die Entität für Prüfungen in der Datenbank repräsentiert. Hier werden die Attribute der Entität als Felder der Klasse definiert und mit den entsprechenden JPA-Annotationen versehen. Die Annotationen `@ManyToOne` und `@JoinColumn` definieren die Beziehung zwischen der Entität für Prüfungen und der Entität für Semesterzeiten. Hier werden auch die `@Nullable`-Annotationen von JSpecify verwendet, die durch die in Abschnitt 5.1 beschriebene statische Code-Analyse überprüft werden.

Listing 5.7: Ausschnitt der ExamJpaEntity-Klasse

```
@Entity
public class ExamJpaEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private @Nullable Long id;
    private String moduleCode;
    private String name;
    @ElementCollection
    private Set<LocalDate> dates = new HashSet<>();
    @ElementCollection
    private Set<LocalTime> times = new HashSet<>();
    @ElementCollection
    private Set<String> rooms = new HashSet<>();
    private Duration duration;
    @Enumerated(EnumType.STRING)
    private @Nullable ExamRetakeFrequency examRetakeFrequency;
    private @Nullable String place;
    @Column(columnDefinition = "text")
    private @Nullable String additionalInformation;
    private @Nullable String examiner;

    @ManyToOne(optional = false, fetch = FetchType.LAZY)
    @JoinColumn(name = "semester_id", nullable = false)
    private SemesterPeriodJpaEntity semester;

    // Entsprechende Konstruktoren, Getter und Setter
}
```

Eine Besonderheit stellt die Entität für Benutzer dar, da diese die Daten über das Studium des Benutzers als JSON-Objekt speichert. Listing 5.8 zeigt einen Ausschnitt dieser `UserJpaEntity`-Klasse. Hier wird das Attribut `data` als `JsonNode` definiert und mit der Annotation `@JdbcTypeCode(SqlTypes.JSON)` versehen, um anzugeben, dass dieses Attribut als JSON in der Datenbank gespeichert werden soll. Die `@Column`-Annotation mit dem Parameter `columnDefinition = "jsonb"` spezifiziert, dass der Datentyp in der Datenbank `JSONB` sein soll.

Listing 5.8: Ausschnitt der UserJpaEntity-Klasse

```
@Entity
public class UserJpaEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private @Nullable Long id;
    @Column(unique = true)
    private String name;
    // Weitere Attribute wie E-Mail, Passwort, etc.

    @JdbcTypeCode(SqlTypes.JSON)
    @Column(columnDefinition = "jsonb")
    private @Nullable JsonNode data;

    // Entsprechende Konstruktoren, Getter und Setter
}
```

Die Entitäten befinden sich im `adapter.out.persistence.entities`-Paket. Im `adapter.out.persistence.repositories`-Paket befinden sich die JPA-Repositories. Die-

se werden von Spring Data JPA automatisch implementiert und bieten Methoden zum Abrufen, Speichern, Aktualisieren und Löschen von Entitäten in der Datenbank. Die Repositories werden von den Anwendungsdiensten im Anwendungskern über die ausgehenden Ports genutzt, um auf die Datenbank zuzugreifen. Listing 5.9 zeigt beispielhaft das `SemesterPeriodJpaRepository`, welches die Entitäten für Semesterzeiten in der Datenbank verwaltet. Das Interface `JpaRepository`, welches hier implementiert wird, bietet bereits viele Methoden zum Abrufen, Speichern, Aktualisieren und Löschen von Entitäten. In diesem Repository wurde zusätzlich eine Methode `existsByStartAndEnd()` definiert, die überprüft, ob eine Semesterzeit mit einem bestimmten Start- und Enddatum bereits in der Datenbank existiert.

Listing 5.9: SemesterPeriodJpaRepository

```
public interface SemesterPeriodJpaRepository extends JpaRepository<
    SemesterPeriodJpaEntity, Long> {
     boolean existsByStartAndEnd(LocalDate start, LocalDate end);
}
```

Um eine saubere Trennung zwischen dem Anwendungskern und der Datenbank zu gewährleisten, müssen Entitäten in Domänenmodelle umgewandelt werden, bevor sie an die Anwendungsdienste im Anwendungskern zurückgegeben werden. Ebenso müssen Domänenmodelle in Entitäten umgewandelt werden, bevor sie in der Datenbank gespeichert werden. Diese Umwandlung wird durch sogenannte „Mapper“ durchgeführt. Die Mapper befinden sich im `adapter.out.persistence.mappers`-Paket und implementieren das `EntityDomainMapper`-Interface, welches in Listing 5.10 gezeigt wird. Dieses Interface definiert zwei Methoden, `toDomain()` und `toEntity()`, die jeweils eine Entität in ein Domänenmodell und ein Domänenmodell in eine Entität umwandeln. Durch das Interface wird eine einheitliche Struktur für alle Mapper definiert.

Listing 5.10: EntityDomainMapper

```
public interface EntityDomainMapper<E, D> {
    D toDomain(E entity);
    E toEntity(D domain);
}
```

Im `adapter.out.persistence`-Paket befinden sich die sogenannten „Repository Adapter“. Diese sind die getriebenen Adapter, die die ausgehenden Ports implementieren. Die Repository Adapter nutzen die JPA-Repositories, um auf die Datenbank zuzugreifen und die Mapper, um Entitäten in Domänenmodelle und umgekehrt umzuwandeln. Listing 5.11 zeigt beispielhaft einen Ausschnitt aus dem `SemesterPeriodRepositoryAdapter`, der den ausgehenden Port `SemesterPeriodRepository` implementiert. Das JPA-Repository und der Mapper werden hier wieder durch Dependency Injection in den Adapter injiziert.

Listing 5.11: SemesterPeriodRepositoryAdapter

```
@Repository
public class SemesterPeriodRepositoryAdapter implements
    SemesterPeriodRepository {
     private final SemesterPeriodJpaRepository repository;
     private final SemesterPeriodMapper mapper;

     public SemesterPeriodRepositoryAdapter(
```

```
        SemesterPeriodJpaRepository repository,
        SemesterPeriodMapper mapper
    ) {
        this.repository = repository;
        this.mapper = mapper;
    }

    @Override
    public Collection<SemesterPeriod> findAll() {
        return repository.findAll().stream()
                .map(mapper::toDomain)
                .collect(Collectors.toList());
    }

    @Override
    public boolean existsByStartAndEnd(
        LocalDate start,
        LocalDate end
    ) {
        return repository.existsByStartAndEnd(start, end);
    }

    // Weitere Methoden zum Speichern, Aktualisieren und Löschen
}
```

5.4 Migration von Benutzerdaten

Die in der Konzeptionierung beschriebene Migration von Benutzerdaten wird mit einer `MigrationRunner`-Klasse und mehreren `Migration`-Klassen umgesetzt. Die `Migration`-Klassen definieren die einzelnen Migrationsschritte und implementieren das `UserData Migration`-Interface mit einer `int fromVersion()` und einer `JsonNode migrate(JsonNode userData)`-Methode. Die `fromVersion()`-Methode gibt die Version der Benutzerdaten an, von der die Migration ausgeführt werden soll. Die `migrate()`-Methode implementiert die eigentliche Migrationslogik und gibt die migrierten Benutzerdaten zurück.

Der Migrationsrunner kennt alle Migrationsklassen und führt die notwendigen Migrationen in der richtigen Reihenfolge aus. Zunächst wird die Version der Benutzerdaten ausgelesen. Solange diese Version nicht der aktuellen Version der Anwendung entspricht, wird die passende Migration für die jeweils aktuelle Version der Daten gesucht und ausgeführt. Anschließend wird die Versionsnummer der Daten um eins erhöht. Dieser Vorgang wiederholt sich, bis die Daten auf dem aktuellen Stand sind. Die migrierten Benutzerdaten werden anschließend in der Datenbank gespeichert. Sollte die Version der Benutzerdaten höher sein als die aktuelle Version der Anwendung, wird eine `UserDataMigrationException` geworfen, da dies im normalen Betrieb nicht vorkommen sollte. Dieselbe Exception wird auch geworfen, wenn keine passende Migration für eine bestimmte Version gefunden werden kann, da dies auf einen Fehler in der Anwendung hindeutet. Listing 5.12 zeigt die `migrate()`-Methode des Migrationsrunners. `migrations` ist eine Map, die die Migrationsklassen nach ihrer `fromVersion()`-Methode speichert. `CURRENT_VERSION` ist die aktuelle Version der Benutzerdaten, die in der Anwendung verwendet wird.

Listing 5.12: Migrate-Methode des Migrationsrunners

```
public JsonNode migrate(JsonNode jsonNode) {
    int version = jsonNode.path("version").asInt(0);
    JsonNode currentJsonNode = jsonNode;

    if (version > CURRENT_VERSION) {
        throw new UserDataMigrationException("Unsupported document
            version " + version);
    }

    while (version != CURRENT_VERSION) {
        UserDataMigration migration = migrations.get(version);
        if (migration == null) throw new UserDataMigrationException
            ("No suitable migration for version " + version);
        currentJsonNode = migration.migrate(currentJsonNode);
        version++;
        setJsonVersion(currentJsonNode, version);
    }

    return currentJsonNode;
}
```

5.5 Benutzersitzungen und Datenspeicherung

In FA-1 des Soll-Konzepts wurde gefordert, dass Benutzer in der Lage sein sollen, ihre Daten als Dateien zu exportieren und importieren oder diese auf dem Server zu speichern.

Um dies zu ermöglichen, werden die Daten, die ein Benutzer in der Anwendung eingibt, als `UserData`-Objekt in einer sogenannten `UserSessionBean` gespeichert. Diese ist mit einer `@SessionScoped`-Annotation versehen und dadurch mit einer Browser-Session des Benutzers verknüpft.

Standardmäßig werden Beans, die mit der `@SessionScoped`-Annotation versehen sind, vor dem Herunterfahren des Servers serialisiert und auf der Festplatte gespeichert, damit sie beim Hochfahren wieder geladen werden können. Damit standardmäßig keine Daten auf dem Server gespeichert werden, ist der Server so konfiguriert, dass die Session-Beans nicht persistiert werden. Dies vermeidet außerdem Probleme mit Dependency Injection beim Deserialisieren der Session-Beans.

Listing 5.13: Has-Changes-Methode der UserSessionBean

```
public boolean hasChanges() {
    return originalDocument != null
            && userData != null
            && !originalDocument.equals(userData);
}
```

Um zu erkennen, ob ein Benutzer Änderungen an seinen Daten vorgenommen hat, werden zwei `UserData`-Objekte in der Session-Bean gespeichert. Das erste Objekt enthält die Daten, die der Benutzer zuletzt gespeichert hat, während das zweite Objekt die aktuellen Daten enthält. Beim Importieren oder Laden der Daten aus der Datenbank werden beide Objekte auf den gleichen Stand gebracht. Wenn der Benutzer Änderungen an seinen Daten vornimmt, wird das zweite Objekt aktualisiert, während das erste unverändert bleibt.

Durch die in Listing 5.13 gezeigte `hasChanges()`-Methode kann überprüft werden, ob die beiden Objekte gleich sind. Wenn dies nicht der Fall ist, hat der Benutzer Änderungen an seinen Daten vorgenommen. Die `UserData`-Klasse implementiert hierfür die `equals()`-Methode. Diese vergleicht alle Felder der beiden Objekte und gibt `true` zurück, wenn alle Felder gleich sind und `false`, wenn mindestens ein Feld unterschiedlich ist.

Wenn ein Benutzer die Daten als Datei exportieren möchte, werden diese in der Session-Bean mithilfe eines `UserDataJsonConverter`-Objekts, welches die Daten mit Jackson in JSON konvertiert, in eine Datei geschrieben und dann heruntergeladen. Beim Importieren werden die Daten aus der Datei mit Jackson in ein `JsonNode`-Objekt deserialisiert und anschließend mit dem `UserDataJsonConverter`-Objekt in ein `UserData`-Objekt konvertiert. Dieses benutzt den in Abschnitt 5.4 beschriebenen Migrationsrunner, um die Daten auf die aktuelle Version zu migrieren. Anschließend werden die Daten in der Session-Bean gespeichert.

Das Laden der Daten aus der Datenbank geschieht implizit, wenn der Benutzer sich anmeldet. Listing 5.14 zeigt, dass die `checkAuthenticationChange()`-Methode, die in Listing 5.15 gezeigt wird, jedes Mal aufgerufen wird, wenn das `preRenderView`-Ereignis ausgelöst wird. Dieses Ereignis wird ausgelöst, bevor die Seite gerendert wird. Diese Methode prüft, ob der aktuell authentifizierte Benutzer ein anderer ist als der Benutzer, dessen Daten in der Session-Bean gespeichert sind oder ob dieser `null` ist. Wenn dies der Fall ist, wird überprüft, ob der Benutzer Änderungen an seinen Daten vorgenommen hat. Wenn dies nicht der Fall ist, werden die Daten des neuen Benutzers aus der Datenbank geladen und in der Session-Bean gespeichert. Wenn der Benutzer Änderungen an seinen Daten vorgenommen hat, also der in Abschnitt 4.3.1 beschriebene Fall eintritt, wird ein Dialog angezeigt, der den Benutzer fragt, ob er die Änderungen verwerfen möchte, um die Daten des neuen Benutzers zu laden, oder ob er die Änderungen behalten möchte. In diesem Fall werden die Daten des neuen Benutzers nicht geladen und die in der Session gespeicherten Daten bleiben erhalten. Das Speichern der Daten in der Datenbank, welches durch den Benutzer ausgelöst wird, geschieht auch wieder über den `UserDataJsonConverter`, welcher die Daten in ein `JsonNode`-Objekt konvertiert, welches anschließend in der Datenbank gespeichert wird.

Listing 5.14: Ausschnitt aus der Layout.xhtml

```
<f:view>
    <f:event
        type="preRenderView"
        listener="#{userSessionBean.checkAuthenticationChange}"
    />
</f:view>
```

Listing 5.15: Check-Authentication-Change-Methode der UserSessionBean

```
public void checkAuthenticationChange() {
    Long current = SpringSecurityUtils.getPrincipal()
            .map(UserPrincipal::getId)
            .orElse(null);

    if (!Objects.equals(current, loadedUserId)) {
        loadedUserId = current;

        if (!hasChanges()) loadFromUser();
        else showLoginConflictDialog = true;
```

```
    }
}
```

6 Deployment

Nachdem die Anwendung im vorherigen Kapitel implementiert wurde, wird in diesem Kapitel die Bereitstellung der Anwendung beschrieben.

Damit die Anwendung unabhängig von der Umgebung ist, auf der diese bereitgestellt wird, wird die Anwendung in einem Docker-Container bereitgestellt und durch Docker Compose orchestriert. Listing 6.1 zeigt die `docker-compose.yml`-Datei, die die Anwendung orchestriert. Es werden zwei Container gestartet, einer für die Anwendung und einer für die PostgreSQL-Datenbank.

Die Anwendung wird auf Port 8080 bereitgestellt und dieser wird auch auf dem Host-System freigegeben. Über Umgebungsvariablen werden die Datenbankverbindungsdetails an die Anwendung übergeben. Über eine `.env`-Datei, die automatisch von Docker Compose eingelesen wird, können diese Umgebungsvariablen überschrieben werden und die Umgebungsvariablen für das Lightweight Directory Access Protocol (LDAP)-Login definiert werden.

Die Datenbank wird in einem eigenen Container bereitgestellt und die Daten werden in einem Docker-Volume gespeichert. Der Datenbank-Container erhält dieselben Umgebungsvariablen wie die Anwendung, um die Datenbankverbindung zu konfigurieren. Sie ist außerdem so konfiguriert, dass sie automatisch neu gestartet wird, falls der Container abstürzt oder das System neu gestartet wird.

Beide Container werden in einem eigenen Docker-Netzwerk bereitgestellt, sodass die Anwendung und die Datenbank miteinander kommunizieren können, ohne dass der Datenbank-Container von außen erreichbar ist.

Listing 6.1: Docker-Compose-Datei

```yaml
services:
  app:
    build: .
    container_name: studiumsplaner-app
    ports:
      - "8080:8080"
    depends_on:
      - postgres
    environment:
      SPRING_PROFILES_ACTIVE: ${SPRING_PROFILES_ACTIVE:-postgres}
      SERVER_PORT: ${SERVER_PORT:-8080}
      DB_HOST: ${DB_HOST:-postgres}
      # Weitere Umgebungsvariablen zur Datenbankverbindung
    networks:
      - internal

  postgres:
    image: postgres:16
    container_name: studiumsplaner-postgres
    restart: unless-stopped
```

```
    environment:
      POSTGRES_DB: ${DB_NAME:-studiumsplaner}
      POSTGRES_USER: ${DB_USERNAME:-studiumsplaner}
      POSTGRES_PASSWORD: ${DB_PASSWORD:-changeme}
    expose:
      - "5432"
    volumes:
      - postgres-data:/var/lib/postgresql/data
    networks:
      - internal

networks:
  internal:

volumes:
  postgres-data:
```

Das Image der Anwendung selbst wird über eine `Dockerfile`-Datei erstellt, die in Listing 6.2 gezeigt wird. Das Image wird in zwei Schritten erstellt. Im ersten Schritt wird ein Maven-Image verwendet, um die Anwendung zu bauen. Hier werden erst die Abhängigkeiten heruntergeladen und anschließend die Anwendung gebaut. Hier werden die Abhängigkeiten mit `mvn dependency:go-offline` heruntergeladen, bevor der Quellcode kopiert wird, um das Docker-Layer-Caching zu ermöglichen. Wenn sich nur der Quellcode ändert, werden die Abhängigkeiten beim Bauen des Images nicht erneut heruntergeladen. Im zweiten Schritt wird ein Java Runtime Environment (JRE)-Image verwendet, um die Anwendung auszuführen. Die fertige JAR-Datei wird aus dem ersten Schritt kopiert und im Container bereitgestellt. Diese wird dann über den `ENTRYPOINT`-Befehl gestartet.

Durch diesen mehrstufigen Bau wird das fertige Image kleiner, da nur die JRE und die fertige JAR-Datei im Image enthalten sind und nicht die gesamte Maven-Umgebung und der Quellcode.

Listing 6.2: Dockerfile der Anwendung

```
FROM maven:3.9.9-eclipse-temurin-21 AS builder
WORKDIR /app
# Copy Maven files
COPY pom.xml .
COPY studiumsplaner-app/pom.xml studiumsplaner-app/
# Weitere Module der Übersichtlichkeit halber nicht aufgelistet
RUN mvn dependency:go-offline
# Copy the source code
COPY . .
RUN mvn -pl studiumsplaner-app -am package -DskipTests

FROM eclipse-temurin:21-jre
WORKDIR /app
COPY --from=builder \
    /app/studiumsplaner-app/target/studiumsplaner-app-1.0.0.jar \
    app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Auf dem Zielsystem müssen Docker und Docker Compose installiert sein, um die Anwendung bereitstellen zu können. Die Anwendung kann dann mit dem Befehl `docker`

`compose up -d -build` gestartet werden. Die `--build`-Option sorgt dafür, dass das Image der Anwendung neu gebaut wird, falls sich der Quellcode geändert hat. Die `-d`-Option sorgt dafür, dass die Container im Hintergrund gestartet werden.

In Zukunft könnte das Image in einem CI/CD-System gebaut und in einer Container-Registry veröffentlicht werden. Dann müsste das Image nicht mehr auf dem Zielsystem gebaut werden, sondern könnte direkt aus der Registry heruntergeladen werden. Allerdings gibt es momentan keine interne Container-Registry und das Image sollte nicht in einer öffentlichen Container-Registry veröffentlicht werden, da die Anwendung nicht für die Öffentlichkeit bestimmt ist.

7 Evaluation

7.1 Testvorgang

Das Testkonzept der Anwendung umfasst zwei Hauptarten von Tests: Unit-Tests und Architekturtests. Die Unit-Tests überprüfen die einzelnen Komponenten der Anwendung isoliert auf korrekte Funktionalität, während die Architekturtests die Einhaltung der in Abschnitt 4.1 definierten Architekturprinzipien und -richtlinien sicherstellen. Beide Testarten sind entscheidend, um die Qualität und Wartbarkeit der Anwendung zu gewährleisten. Diese Tests sind vor allem wichtig, damit zukünftige Änderungen an der Anwendung die bestehende Funktionalität nicht unbeabsichtigt beeinträchtigen und die Architekturprinzipien weiterhin eingehalten werden.

Integrationstests und End-to-End-Tests sind zwar auch wichtige Testarten, wurden jedoch aufgrund des Umfangs der Arbeit nicht implementiert. Integrationstests würden die Interaktion zwischen mehreren Komponenten der Anwendung testen, während End-to-End-Tests die gesamte Anwendung aus Sicht des Benutzers überprüfen würden.

7.1.1 Unit-Tests

Die Unit-Tests wurden mit dem JUnit-Framework erstellt, das eine weit verbreitete Testbibliothek für Java-Anwendungen ist. JUnit bietet eine einfache Möglichkeit, Testfälle zu definieren und auszuführen, sowie umfangreiche Assertions, um die erwarteten Ergebnisse zu überprüfen. Mockito wurde verwendet, um Abhängigkeiten zu simulieren und die Tests auf die zu testenden Komponenten zu konzentrieren. Beides wurde über den Spring Boot Test Starter in das Projekt integriert.

In der Hauptanwendung wurden nur Klassen aus dem `service`-Paket getestet, da diese die Geschäftslogik der Anwendung enthalten. Adapter sind eng an externe Systeme gebunden, weshalb ein isolierter Test wenig über die tatsächliche Korrektheit der Adapter aussagen kann. Hier wären Integrationstests sinnvoll, um die Interaktion zwischen den Adaptern und den externen Systemen zu überprüfen.

Durch die Strukturierung der Anwendung nach der hexagonalen Architektur und der Dependency Injection von Spring Boot sind die Service-Klassen von den Datenzugriffs- und Präsentationsschichten entkoppelt, was das Testen erleichtert. Abhängigkeiten wie Repositories und externe Services wurden in den Tests mit Mockito simuliert, sodass die Service-Klassen isoliert getestet werden konnten.

Listing 7.1 zeigt beispielhaft einen Unit-Test für eine Service-Klasse, die die Geschäftslogik für das Abrufen von Modulen für einen bestimmten Studiengang implementiert. In diesem Test wird überprüft, ob der Service die erwarteten Module für einen gegebenen Studiengang zurückgibt. Die Abhängigkeiten `CourseRepository` und `ModuleRepository` werden mit Mockito simuliert, um die Testumgebung zu kontrollieren und sicherzustellen, dass der Test nur die Logik der Service-Klasse überprüft.

Listing 7.1: GetModulesForCourseServiceTest

```
@ExtendWith(MockitoExtension.class)
class GetModulesForCourseServiceTest {
    @Mock
    private CourseRepository courseRepository;
    @Mock
    private ModuleRepository moduleRepository;
    @InjectMocks
    private GetModulesForCourseService service;

    @Test
    void returnsModulesForExistingCourse() {
        Course course = mock(Course.class);
        when(course.getbKey()).thenReturn("BKEY-1");
        when(courseRepository.findById(1L)).thenReturn(Optional.of(
            course));

        CourseModule module1 = mock(CourseModule.class);
        CourseModule module2 = mock(CourseModule.class);
        when(moduleRepository.findByBKey("BKEY-1")).thenReturn(List
            .of(module1, module2));

        List<CourseModule> result = service.execute(1L);

        assertThat(result).containsExactly(module1, module2);
    }

    // Weitere Testfälle der Übersicht halber ausgelassen
}
```

7.1.2 Architekturtests

Die Architekturtests wurden mit dem ArchUnit-Framework implementiert, welches es ermöglicht, die Einhaltung von Architekturprinzipien und -richtlinien in Java-Anwendungen zu überprüfen. ArchUnit bietet eine Fluent-API, mit der definiert werden kann, welche Abhängigkeiten zwischen Paketen und Klassen erlaubt oder verboten sind oder welche Klassen wie benannt sein müssen.

Die meisten Regeln wurden in der Klasse `ArchitectureTest` definiert, die im Testpaket der Anwendung liegt und in Listing 7.2 gezeigt wird. In dieser Klasse werden die Regeln für die Architekturtests definiert, wie z.B. die `applicationKnowsNoAdapters`-Regel. Diese stellt sicher, dass Klassen im `application`-Paket nicht direkt von Klassen im `adapter`-Paket abhängen, was ein zentrales Prinzip der hexagonalen Architektur ist.

Es werden auch noch weitere Regeln definiert, wie z.B. die Trennung von eingehenden und ausgehenden Adaptern, die Einhaltung von Namenskonventionen für die Use-Case-Interfaces und die Service-Klassen sowie die Vermeidung von zyklischen Abhängigkeiten zwischen den Paketen.

Listing 7.2: ArchitectureTest

```
@AnalyzeClasses(
        packages = "de.htwsaar.stl.studiumsplaner.app",
```

```
        importOptions = ImportOption.DoNotIncludeTests.class
)
class ArchitectureTest {

    private static final String BASE = "de.htwsaar.stl.
       studiumsplaner.app";

    @ArchTest
    static final ArchRule applicationKnowsNoAdapters =
            noClasses().that().resideInAPackage(BASE + ".
               application..")
                    .should().dependOnClassesThat()
                    .resideInAPackage(BASE + ".adapter..");

    // Weitere Regeln der Übersicht halber ausgelassen
}
```

7.2 Beantwortung der Forschungsfragen

In diesem Kapitel werden die in Abschnitt 1.3 definierten Forschungsfragen beantwortet. Die Beantwortung erfolgt auf Basis der in den vorherigen Kapiteln beschriebenen Konzeptionierung und Implementierung der Anwendung sowie der durchgeführten Tests.

7.2.1 Flexibilität des Datenmodells (FF1)

Um die Informationen aus unterschiedlichen Quellen und in unterschiedlichen Formaten zusammenzustellen, wurde, wie in Abschnitt 4.1 beschrieben, pro Datenquelle ein von der Hauptanwendung unabhängiges Parser-Modul implementiert. Diese Parser-Module sind für die Extraktion und Transformation der Daten aus den jeweiligen Quellen verantwortlich und liefern die Daten in einem einheitlichen Format an die Hauptanwendung. Die Hauptanwendung ist somit nicht direkt von den spezifischen Formaten der Datenquellen abhängig, sondern kann die Daten in einem standardisierten Format verarbeiten.

Wie in Abschnitt 4.2 beschrieben, wurden die Beziehungen zwischen den Entitäten der Anwendung sowie andere Integritätsbedingungen in der Geschäftslogik der Anwendung, anstatt wie üblich in der Datenbank, implementiert. Dies ermöglicht eine flexible Anpassung des Datenmodells an die spezifischen Anforderungen der verschiedenen Studiengänge, ohne dass Änderungen an der Datenbankstruktur erforderlich sind. Die Geschäftslogik kann so angepasst werden, dass sie die spezifischen Anforderungen eines Studiengangs erfüllt, während das zugrunde liegende Datenmodell unverändert bleibt.

7.2.2 Unabhängigkeit und Erweiterbarkeit der Architektur (FF2)

Die in Abschnitt 4.1 beschriebene hexagonale Architektur der Anwendung ermöglicht eine hohe Unabhängigkeit von den bestehenden Systemen und erleichtert die Erweiterung und Anpassung der Anwendung in der Zukunft. Durch die strikte Trennung zwischen Geschäftslogik und externen Systemen über Ports und Adapter kann die Geschäftslogik der Anwendung unabhängig von den spezifischen Implementierungen der externen Systeme entwickelt und getestet werden. Diese Architektur hat sich vor allem bei den

Unit-Tests als vorteilhaft erwiesen, da die Geschäftslogik der Anwendung sehr einfach isoliert getestet werden konnte, ohne dass die externen Systeme verfügbar sein mussten. Die Architektur ermöglicht es auch, neue externe Systeme oder Datenquellen in die Anwendung zu integrieren, ohne dass Änderungen an der bestehenden Geschäftslogik erforderlich sind. Wenn sich die Datenquellen ändern, können die entsprechenden Adapter auch angepasst werden, ohne dass die Geschäftslogik der Anwendung betroffen ist. Dies erleichtert die Wartung und Weiterentwicklung der Anwendung erheblich.

7.2.3 Selbstbestimmte Datenhaltung (FF3)

Um den Studierenden die Möglichkeit zu geben, selbst zu entscheiden, ob und in welcher Form ihre Daten gespeichert werden, wurde ein System entwickelt, bei dem Nutzerdaten erst nur in einer Session gespeichert werden, bevor der Benutzer auswählen kann, ob die Daten als Datei exportiert oder mit seinem HIZ-Login in der Datenbank gespeichert werden sollen. Exportierte Dateien können von den Studierenden auf ihrem eigenen Gerät gespeichert und bei Bedarf wieder in die Anwendung importiert werden.

Um die Datenhaltung konsistent zu gestalten, werden die Daten in eine standardisierte JSON-Struktur serialisiert, welche sowohl für den Export als Datei als auch für die Speicherung in der Datenbank verwendet wird. Die Anwendung stellt sicher, dass die Daten in beiden Fällen in der gleichen Struktur vorliegen, sodass die Konsistenz der Datenhaltung gewährleistet ist. Die JSON-Daten werden außerdem mit einer Versionsnummer versehen, sodass die Anwendung bei Änderungen an der Datenstruktur in der Lage ist, ältere Versionen der Daten zu erkennen und entsprechend zu verarbeiten. Zusätzlich stellt der in Abschnitt 4.3 beschriebene Konflikterkennungsmechanismus sicher, dass beim Laden von Serverdaten keine lokal vorgenommenen Änderungen unbemerkt überschrieben werden.

Es konnte gezeigt werden, dass die in dieser Arbeit entwickelte Anwendung die in Abschnitt 1.3 definierten Forschungsfragen erfolgreich beantwortet. Das flexible Datenmodell, die unabhängige und erweiterbare Architektur sowie die selbstbestimmte Datenhaltung tragen dazu bei, dass die Anwendung auch in Zukunft an die sich ändernden Anforderungen der Studierenden und der Hochschule angepasst werden kann.

8 Fazit und Ausblick

8.1 Fazit

In dieser Arbeit wurde ausgehend von den in Abschnitt 1.2 beschriebenen Herausforderungen eine Webanwendung konzipiert und entwickelt, die Studierenden bei der Planung ihres Studiums und ihres Semesters unterstützt. Die Anwendung aggregiert die relevanten Informationen aus den verschiedenen Quellen und stellt diese in einer einheitlichen Oberfläche dar. Sie ermöglicht es den Studierenden, ihr Gesamtstudium auf einen Blick zu erfassen und ihre Semesterplanung effizient und intuitiv durchzuführen. Die Anwendung wurde abschließend mit Unit-Tests und Architekturtests getestet, um die Qualität und Wartbarkeit der Anwendung sicherzustellen und die Einhaltung der in Abschnitt 4.1 definierten Architekturprinzipien und -richtlinien zu überprüfen.

Wie in Abschnitt 7.2 dargestellt, konnten die in Abschnitt 1.3 definierten Forschungsfragen erfolgreich beantwortet werden. Durch das flexible Datenmodell bleibt die Anwendung auch in Zukunft anpassbar, etwa wenn weitere Studiengänge unterstützt werden sollen oder sich die zugrunde liegenden Daten ändern. Die unabhängige und erweiterbare Architektur ermöglicht es, die Anwendung an die sich ändernden Anforderungen der Studierenden und der Hochschule anzupassen. Die selbstbestimmte Datenhaltung wurde durch eine Kombination aus Session-Speicherung, Export- und Import-Funktionalität sowie einer zentralen Speicherung in der Datenbank umgesetzt. Studierende können dadurch selbst entscheiden, ob und in welcher Form ihre Daten gespeichert werden. Die konsistente Datenhaltung wurde durch ein einheitliches, versioniertes JSON-Format realisiert.

8.2 Ausblick

Wie in Kapitel 6 beschrieben, wäre die Einführung einer CI/CD-Pipeline ein sinnvoller nächster Schritt. Diese könnte sowohl das automatische Bauen und Bereitstellen des Docker-Images in einer Container-Registry als auch die automatische Ausführung der in Abschnitt 7.1 beschriebenen Unit- und Architekturtests bei Codeänderungen übernehmen. Dadurch ließe sich die Anwendung ohne manuellen Build-Schritt in verschiedenen Umgebungen bereitstellen, während gleichzeitig frühzeitig auf Qualitätsprobleme reagiert werden könnte.

Zusätzlich zu den Unit- und Architekturtests könnten auch, wie in Abschnitt 7.1 beschrieben, Integrationstests implementiert werden, die die Interaktion der Anwendung mit den externen Systemen testen, und Ende-zu-Ende-Tests, die die Anwendung aus Sicht des Benutzers testen. Diese Tests könnten ebenfalls in die CI/CD-Pipeline integriert werden.

Neben den Tests und der CI/CD-Pipeline könnten auch neue Features implementiert werden, etwa der Export des Stundenplans zum Drucken oder als iCal-Datei. Letzteres würde es Studierenden ermöglichen, ihre Veranstaltungen direkt in ihren persönlichen

Kalenderanwendungen zu verwalten. Auch Benachrichtigungen für Anmeldefristen oder Änderungen im Stundenplan könnten implementiert werden, um die Studierenden auch über Änderungen auf dem Laufenden zu halten.

Neben den bisher unterstützten Studiengängen könnten, sobald sich die Anwendung in der Praxis bewährt hat, weitere Studiengänge der Fakultät für Ingenieurwissenschaften oder sogar anderer Fakultäten der Hochschule in die Anwendung integriert werden. Dies würde es einer größeren Anzahl von Studierenden ermöglichen, die Anwendung zu nutzen und von den Vorteilen der zentralen Informationsquelle und der effizienten Semesterplanung zu profitieren.

Literatur

[1] Agile Business Consortium. *What is MoSCoW Prioritization?* Agile Business Consortium. 28. Mai 2026. URL: https://www.agilebusiness.org/resource/what-is-moscow-prioritization/ (besucht am 04. 09. 2026).

[2] Apache Software Foundation. *Apache Ant*. 2026. URL: https://ant.apache.org/ (besucht am 18. 08. 2026).

[3] Apache Software Foundation. *Apache Maven*. 2026. URL: https://maven.apache.org/ (besucht am 18. 08. 2026).

[4] Apache Software Foundation. *Standard Directory Layout*. 2026. URL: https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html (besucht am 28. 08. 2026).

[5] Alistair Cockburn. *The Hexagonal (Ports and Adapters) Architecture*. HaT Technical Report 2005.02, v0.9, vom 2005-09-04. 2005. URL: https://alistair.cockburn.us/hexagonal-architecture (besucht am 24. 08. 2026).

[6] Sébastien Deleuze. *Null-safe applications with Spring Boot 4*. Nov. 2025. URL: https://spring.io/blog/2025/11/12/null-safe-applications-with-spring-boot-4 (besucht am 08. 09. 2026).

[7] Docker. *Docker Compose Documentation*. 2026. URL: https://docs.docker.com/compose/ (besucht am 28. 08. 2026).

[8] Docker. *What is Docker?* 2026. URL: https://docs.docker.com/get-started/docker-overview/ (besucht am 24. 08. 2026).

[9] Docker. *What is a Container?* 2026. URL: https://www.docker.com/resources/what-container/ (besucht am 24. 08. 2026).

[10] FasterXML. *Jackson Project Home*. 2026. URL: https://github.com/FasterXML/jackson (besucht am 22. 09. 2026).

[11] C. A. R. Hoare. „An Axiomatic Basis for Computer Programming“. In: *Communications of the ACM* 12.10 (Okt. 1969), S. 576–580. DOI: 10.1145/363235.363259.

[12] JSpecify Team. *JSpecify Documentation*. URL: https://jspecify.dev/ (besucht am 21. 08. 2026).

[13] JetBrains. *Annotations | IntelliJ IDEA Documentation*. URL: https://www.jetbrains.com/help/idea/annotating-source-code.html (besucht am 21. 08. 2026).

[14] JoinFaces Team. *JoinFaces Documentation*. URL: http://joinfaces.org/ (besucht am 20. 08. 2026).

[15] Bernd Müller. *JavaServer Faces und Jakarta Server Faces 2.3 ein Arbeitsbuch für die Praxis*. Hanser, 2021. ISBN: 9783446456709.

[16] PostgreSQL Global Development Group. *PostgreSQL: About*. URL: https://www.postgresql.org/about/ (besucht am 21. 08. 2026).

[17] PostgreSQL Global Development Group. *PostgreSQL Documentation: 16: 8.14. JSON Types*. URL: https://www.postgresql.org/docs/16/datatype-json.html (besucht am 21. 08. 2026).

[18] PrimeFaces Team. *Primefaces 15 Documentation*. 2025. URL: https://primefaces.github.io/primefaces/15_0_0 (besucht am 20. 08. 2026).

[19] React Team. *React Homepage*. 2026. URL: https://react.dev/ (besucht am 08. 09. 2026).

[20] Alexander Salvanos. *Professionell entwickeln mit Java EE 8: Das umfassende Handbuch*. Rheinwerk Verlag, 2018. ISBN: 978-3-8362-4353-7.

[21] Spring Team. *Spring Boot Documentation*. 2026. URL: https://docs.spring.io/spring-boot/index.html (besucht am 27. 08. 2026).

[22] Christian Ullenboom. *Spring Boot 3 und Spring Framework 6 das umfassende Handbuch*. Rheinwerk Verlag GmbH, 2023. ISBN: 9783836290494.

[23] VMware. *Gradle Plugin :: Spring Boot*. Spring Boot 4.1.1. 2026. URL: https://docs.spring.io/spring-boot/gradle-plugin/index.html (besucht am 08. 09. 2026).

[24] VMware. *JSON :: Spring Boot*. Spring Boot 4.1.1. Spring, Broadcom. 2026. URL: https://docs.spring.io/spring-boot/reference/features/json.html (besucht am 22. 09. 2026).

[25] VMware. *Maven Plugin :: Spring Boot*. Spring Boot 4.1.1. 2026. URL: https://docs.spring.io/spring-boot/maven-plugin/index.html (besucht am 08. 09. 2026).

[26] VMware. *SQL Databases: JPA and Spring Data JPA*. Spring Boot 4.1.1. 2026. URL: https://docs.spring.io/spring-boot/reference/data/sql.html#data.sql.jpa-and-spring-data (besucht am 22. 09. 2026).

[27] Vaadin Team. *Vaadin Homepage*. 2026. URL: https://vaadin.com/ (besucht am 08. 09. 2026).

Abbildungsverzeichnis

2.1 Docker vs. Virtual Machine (eigene Darstellung, nach [9]) 12
2.2 Hexagonale Architektur (eigene Darstellung) 14

3.1 Swimlane-Diagramm der Semesterplanung (eigene Darstellung) 18

4.1 Übersicht der Module der Anwendung (eigene Darstellung) 27
4.2 Systemarchitektur der Anwendung (eigene Darstellung) 29
4.3 Datenmodell der Anwendung (eigene Darstellung) 31
4.4 Migrationsablauf von Nutzerdaten (eigene Darstellung) 32
4.5 Authentifizierungsablauf der Anwendung (eigene Darstellung) 35

5.1 Paketstruktur der Hauptanwendung (eigene Darstellung) 41

Tabellenverzeichnis

3.1 Übersicht der beteiligten Systeme 16
3.2 Priorisierung der Anforderungen 23

Listings

2.1 Beispiel einer Parent pom.xml 6
2.2 Beispiel einer Modul-pom.xml 6
2.3 Beispiel einer Docker-Compose-Datei 12

5.1 package-info.java 37
5.2 Ausschnitt aus der pom.xml des Parent Projekts 37
5.3 pom.xml des Semesterzeiten-Parser-Moduls 39
5.4 Schnittstelle des Moduldb-Parser-Moduls 40
5.5 ModulesParseUseCase 41
5.6 ModulesParseService 41
5.7 Ausschnitt der ExamJpaEntity-Klasse 42
5.8 Ausschnitt der UserJpaEntity-Klasse 43
5.9 SemesterPeriodJpaRepository 44

5.10 EntityDomainMapper 44
5.11 SemesterPeriodRepositoryAdapter 44
5.12 Migrate-Methode des Migrationsrunners 45
5.13 Has-Changes-Methode der UserSessionBean 46
5.14 Ausschnitt aus der Layout.xhtml 47
5.15 Check-Authentication-Change-Methode der UserSessionBean 47

6.1 Docker-Compose-Datei 49
6.2 Dockerfile der Anwendung 50

7.1 GetModulesForCourseServiceTest 54
7.2 ArchitectureTest 54

A.1 pom.xml der Hauptanwendung 67

Abkürzungsverzeichnis

HTW	Hochschule für Technik und Wirtschaft des Saarlandes
DI	Dependency Injection
J2EE	Java 2 Platform, Enterprise Edition
JSF	JavaServer Faces
Jakarta EE	Jakarta Enterprise Edition
Java EE	Java Enterprise Edition
MVC	Model-View-Controller
SQL	Structured Query Language
JSON	JavaScript Object Notation
ECTS	European Credit Transfer and Accumulation System
HIZ	Hochschul-IT-Zentrum
API	Application Programming Interface
JPA	Java Persistence API
JRE	Java Runtime Environment
LDAP	Lightweight Directory Access Protocol

Anhang

A Listings

A.1 pom.xml der Hauptanwendung

Listing A.1: pom.xml der Hauptanwendung

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
            http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>de.htwsaar.stl.studiumsplaner</groupId>
        <artifactId>studiumsplaner-parent</artifactId>
        <version>1.0.0</version>
    </parent>

    <artifactId>studiumsplaner-app</artifactId>
    <name>StudiumsPlaner</name>
    <description>StudiumsPlaner</description>

    <properties>
        <java.version>21</java.version>
        <maven.compiler.release>21</maven.compiler.release>
        <project.build.sourceEncoding>UTF-8</project.build.
           sourceEncoding>
        <start-class>
            de.htwsaar.stl.studiumsplaner.app.
               StudiumsPlanerApplication
        </start-class>
    </properties>

    <repositories>
        <repository>
            <id>strassburger-repo</id>
            <url>https://maven.strassburger.org</url>
        </repository>
    </repositories>

    <dependencies>
        <dependency>
            <groupId>org.strassburger</groupId>
            <artifactId>prime-lucide</artifactId>
            <version>1.24.0</version>
        </dependency>

        <dependency>
```

```
            <groupId>de.htwsaar.stl.studiumsplaner</groupId>
            <artifactId>courses-parser</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>de.htwsaar.stl.studiumsplaner</groupId>
            <artifactId>exam-parser</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>de.htwsaar.stl.studiumsplaner</groupId>
            <artifactId>semester-parser</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>de.htwsaar.stl.studiumsplaner</groupId>
            <artifactId>moduldb-parser</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>de.htwsaar.stl.studiumsplaner</groupId>
            <artifactId>timetable-parser</artifactId>
            <version>${project.version}</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-ldap</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.joinfaces</groupId>
            <artifactId>primefaces-spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.joinfaces</groupId>
            <artifactId>omnifaces-spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
```

```
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <version>${spring-boot.version}</version>

                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

Kolophon

Dieses Dokument wurde mit der LaTeX-Vorlage für Abschlussarbeiten an der htw saar im Bereich Informatik/Mechatronik-Sensortechnik erstellt (Version 2.25, August 2024). Die Vorlage wurde von Yves Hary und André Miede entwickelt (mit freundlicher Unterstützung von Thomas Kretschmer, Helmut G. Folz und Martina Lehser). Daten: (F)10.95 – (B)426.79135pt – (H)688.5567pt