

## Konzeption und Entwicklung eines Systems zur Verwaltung und Erstellung von Modulhandbüchern

**Tobias Wylega**

Suggested citation:

Wylega, Tobias. 2024. "Konzeption und Entwicklung eines Systems zur Verwaltung und Erstellung von Modulhandbüchern." Hannover: Hochschule Hannover. <https://doi.org/10.25968/opus-3547>.

### Abstract

Diese Bachelorarbeit befasst sich mit der Konzeption und Entwicklung eines webbasierten Systems zur Verwaltung und Erstellung von Modulhandbüchern. Ziel der Arbeit ist es, den bisherigen – durch manuelle Prozesse und redundante Datenpflege gekennzeichneten – Workflow an der Hochschule Hannover zu optimieren, um eine effiziente und benutzerfreundliche Alternative zu schaffen.

Ausgehend von einer fundierten Anforderungsanalyse – die unter anderem eine Aufstellung der Use Cases sowie eine kritische Betrachtung des bestehenden Systems umfasst – wird ein ganzheitlicher Lösungsansatz verfolgt. Die Implementierung beinhaltet die Weiterentwicklung eines Backends auf Basis von NestJS und Prisma sowie die Neuentwicklung eines Frontends mit Angular. Besondere Aufmerksamkeit gilt dabei der Codequalität, der Benutzerfreundlichkeit und der Mehrsprachigkeit, um ein langfristig wartbares, erweiterbares und benutzbares System zu gewährleisten.

Im Rahmen der Implementierung wird zudem ein automatisierter Prozess zur Generierung von PDF-Dokumenten integriert, der den administrativen Aufwand reduziert und die Konsistenz der Modulhandbücher sicherstellt. Die Evaluation des Prototyps zeigt, dass das entwickelte System die identifizierten Schwachstellen des bisherigen Prozesses adressiert und als solide Grundlage für zukünftige Arbeiten dient.

### Terms of use

CC BY-SA 4.0

**HOCHSCHULE  
HANNOVER**  
UNIVERSITY OF  
APPLIED SCIENCES  
AND ARTS

–  
*Fakultät IV  
Wirtschaft und  
Informatik*

# **Konzeption und Entwicklung eines Systems zur Verwaltung und Erstellung von Modulhandbüchern**

Tobias Wylega

Bachelorarbeit im Studiengang Mediendesigninformatik

13. August 2024





**Autor:** Tobias Wylega  
tobias.github.bachelorarbeit@wylega.de  
Matrikelnummer: 1629483

**Erstprüfer:** Prof. Dr. Dennis Allerkamp  
Abteilung Informatik, Fakultät IV  
Hochschule Hannover  
dennis.allerkamp@hs-hannover.de

**Zweitprüfer:** Prof. Dr. Matthias Hovestadt  
Abteilung Informatik, Fakultät IV  
Hochschule Hannover  
matthias.hovestadt@hs-hannover.de

### **Selbständigkeitserklärung**

Hiermit erkläre ich, dass ich die eingereichte Bachelorarbeit selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Hannover, den 13. August 2024

Unterschrift

# Inhaltsverzeichnis

<b>1. Einleitung</b> .....	<b>1</b>
1.1. Hintergrund und Motivation .....	1
1.2. Definition und Zweck eines Modulhandbuchs .....	1
1.3. Ähnliche Arbeiten .....	3
1.4. Aufbau der Arbeit .....	3
<b>2. Anforderungsanalyse</b> .....	<b>5</b>
2.1. Systemumgebung .....	5
2.1.1. Geplante Anwendungen .....	5
2.1.2. Struktur des bestehenden Backends .....	6
2.1.3. Struktur der bestehenden Datenbank .....	7
2.1.4. Struktur einer beispielhaften Angular Anwendung .....	8
2.2. Interview mit Studiendekan .....	8
2.3. Analyse des aktuellen Arbeitsprozesses und Identifikation von Schwachstellen ....	9
2.4. Zielgruppen .....	10
2.4.1. Studieninteressierte .....	10
2.4.2. Studierende .....	10
2.4.3. Modulverantwortliche und Studiengangsverantwortliche .....	10
2.5. Use Cases .....	11
2.6. Anforderungen .....	14
2.6.1. Funktionale Anforderungen .....	15
2.6.2. Nicht-Funktionale Anforderungen .....	19
2.6.3. Ergebnis .....	24
2.7. Struktur eines Modulhandbuches .....	24
2.7.1. Moduleigenschaften .....	25
2.7.2. Teilmoduleigenschaften .....	27
2.8. Zwischenfazit .....	28
<b>3. Entwurf</b> .....	<b>29</b>
3.1. Aufbau der Datenbank .....	29
3.1.1. Benötigte Tabellen .....	29
3.1.2. Überlegungen zur Übersetzbarkeit .....	30
3.1.3. Resultierendes Schema .....	34
3.2. Benutzeroberflächen .....	35
3.2.1. Grundgerüst .....	35
3.2.2. Komponenten und Ansichten .....	37

3.3. Benötigte Endpunkte im Backend .....	45
3.4. Zwischenfazit .....	48
<b>4. Implementierung .....</b>	<b>49</b>
4.1. Backend .....	49
4.1.1. Datenbank .....	49
4.1.2. HTTP-Endpunkte .....	50
4.1.3. PDFs generieren .....	64
4.2. Frontend .....	69
4.2.1. Routing .....	69
4.2.2. Design .....	71
4.2.3. Übersetzbarkeit .....	75
4.2.4. Erstellen eines neuen PDFs .....	79
4.2.5. Subscriptions, Intervalle und Memory Leaks .....	81
4.2.6. Suchfunktion .....	81
4.2.7. URL-basierte Erkennung von Seitenelementen .....	82
4.2.8. Anlegen und Bearbeiten von Modulen .....	83
4.3. Dokumentation .....	87
4.4. Podman .....	89
4.5. Zwischenfazit .....	96
<b>5. Review .....</b>	<b>97</b>
5.1. Interview mit modilverantwortlicher Person .....	97
5.2. Abweichungen zum Prototypen .....	97
5.3. Vergleich der PDFs .....	98
5.4. Überprüfung, ob Anforderungen erfüllt sind .....	101
5.5. Zwischenfazit .....	109
<b>6. Fazit .....</b>	<b>111</b>
<b>7. Ausblick .....</b>	<b>113</b>
<b>Anhang .....</b>	<b>115</b>
Quellcode .....	115
Interview-Fragen .....	116
<b>Literaturverzeichnis .....</b>	<b>117</b>

# 1. Einleitung

## 1.1. Hintergrund und Motivation

Modulhandbücher begegnen vielen Menschen bereits vor dem Start des Studiums bei der Recherche nach interessanten Studiengängen [1]. In Modulhandbüchern befinden sich die wesentlichen Informationen und Rahmenbedingungen zu den einzelnen Modulen eines Studiengangs. Da sich Lehrinhalte, Zuständigkeiten und auch ganze Studiengänge verändern [2], müssen die Modulhandbücher ebenfalls regelmäßig bearbeitet und bereitgestellt werden. Dieser Aktualisierungsprozess wird vom zuständigen Dekanat der Abteilung Informatik als suboptimal angesehen, weshalb Verbesserungsvorschläge diskutiert werden. Weiterhin befinden sich andere Systeme in der Entwicklung (mehr dazu in Unterabschnitt 2.1.1), welche von einer korrekten und aktuellen Auflistung aller Module eines Studiengangs profitieren. In dieser Arbeit soll eine Webanwendung erstellt werden, welche die Bearbeitung und Veröffentlichung von Modulhandbüchern erleichtert. Der neue Prozess soll eine bessere Usability haben und effizienter sein. Das Ergebnis dieser Arbeit soll eine lauffähige Software sein, mit der Modulhandbücher in einer Oberfläche verwaltet, bearbeitet und angezeigt werden können. Die Weboberfläche, die von den Anwendern genutzt werden wird, wird „StudyModules“ heißen.

## 1.2. Definition und Zweck eines Modulhandbuchs

§ 7 der niedersächsischen Studienakkreditierungsverordnung [3] beschreibt, dass jeder Studiengang in zeitlich und thematisch abgegrenzte Module einzuteilen ist. Des Weiteren ist eine Beschreibung des Moduls und dessen Merkmale erforderlich. Die Aufgabe eines Modulhandbuchs ist es, die Orientierung im Studium zu erleichtern und die Prüfungsordnung abzubilden. Das Handbuch soll über die einzelnen Lehrveranstaltungen informieren und diese detailliert beschreiben. [4]

Es gibt noch weitere Dokumente, die Informationen über die Module eines Studiengangs enthalten. In Abbildung 1 ist das vorgeschlagene Curriculum des Studiengangs Mediendesigninformatik zu sehen. Hier entspricht eine Spalte einem Semester. Die Höhe der Module spiegelt die Anzahl der zu erreichenden Credits wider.

Hochschule Hannover, Studiengang B.Sc. Mediendesigninformatik (MDI)							
Credits	Semester						
	1	2	3	4	5	6	7
0							
5	Programmieren 1	Programmieren 2	Programmierprojekt (MDI)	Software Engineering 1	Praxisphase/ Auslandssemester	Wahlpflichtfach Informatik 1	Wahlpflichtfach Informatik 2
10	Grundlagen der Informatik	Datenbanksysteme 1	Betriebssysteme und Netze 1	Statistik		Web and Mobile Applications (MDI)	Praxisprojekt 2 (F3-F4 interdisziplinär / Film / Animation)
15	Mathematik 1	Mathematik 2 (MDI)	Usability (MDI)	Computergrafik 1 (MDI)		Seminar	
20	Startprojekt	Algorithmen und Datenstrukturen	Concept Design	Webtechnologien	Praxis-/Auslandsseminar	Praxisprojekt 1 (F3-F4 interdisziplinär / Film / Animation)	Bachelor-Arbeit mit Kolloquium
25	Animation 1						
30	Bildbearbeitung 1	Mediendesign	Projekt (Design)	Content Design		Content Design Entwurf	
	Betriebswirtschaft	Englisch				Ergänzendes Fach 2	

Angewandte Informatik	Mediendesign	Wahlmodule (Informatik und/oder Mediendesign)	Fächerübergreifende Module
-----------------------	--------------	---	----------------------------

**Legende**

Abbildung 1: Curriculum Mediendesigninformatik [5]

Im Anhang des besonderen Teils der Prüfungsordnung [6] befindet sich eine Auflistung aller Module des Studiengangs in tabellarischer Form (Abbildung 2).

Erster Studienabschnitt			
Pflichtmodule_1. Studienabschnitt			
M-Kürzel	Modul-Bezeichnung	Art <sup>M</sup>	CR <sup>M</sup>
MDI-100	Mathematik 1	PF	6
MDI-101	Startprojekt	PF	4
MDI-102	Programmieren 1	PF	6

Abbildung 2: Auszug Anhang Prüfungsordnung [6]

Es gibt verschiedene Zielgruppen für Modulhandbücher, die in Abschnitt 2.4 genauer betrachtet werden. Des Weiteren wird in Abschnitt 2.7 die Struktur von verschiedenen Modulhandbüchern untersucht und anschließend in Abschnitt 3.1 ein dazu passendes Datenbankschema erstellt.

### 1.3. Ähnliche Arbeiten

Bevor mit der Planung des neuen Systems zur Verwaltung von Modulhandbüchern begonnen wurde, fand zunächst eine Recherche zu ähnlichen Arbeiten statt. Ein System, welches ein ähnliches Problem löst, ist der „Curriculum Designer“ im HISinOne [7] der HIS Hochschul-Informationen-System eG [8]. Das HISinOne wird beispielsweise von der Universität Hohenheim zur Verwaltung der Modulhandbücher genutzt [9]. Weiterhin ist das HISinOne an der Hochschule Hannover im Einsatz [10], um dort zum Beispiel die Prüfungsanmeldungen zu realisieren.

Da es keine öffentlichen Dokumentationen für die Nutzung der Schnittstellen vom HISinOne gab und zu diesem Zeitpunkt auch unklar war, ob das HISinOne hochschulweit zur Verwaltung von Modulhandbüchern eingesetzt werden sollte, bot es sich an, ein eigenes System zu entwickeln. Das selbst entwickelte System hatte den zusätzlichen Vorteil, dass es an alle besonderen Anforderungen angepasst werden konnte, ohne einen Antrag bei der HIS Hochschul-Informationen-System eG stellen zu müssen. Sollte die Hochschule Hannover entscheiden, das HISinOne in Zukunft auch für die Verwaltung von Modulhandbüchern zu nutzen, könnten die Informationen der Handbücher über Webservices vom HISinOne und über die REST-Schnittstellen des neuen Systems synchronisiert werden.

### 1.4. Aufbau der Arbeit

In dieser Arbeit werden zunächst in Kapitel 2 die Anforderungen an das neue System ermittelt. Dafür wird der aktuelle Prozess analysiert, auf Schwachstellen geprüft und es werden die Zielgruppen ermittelt. Anschließend werden Use Cases sowie ein zukünftiger Arbeitsprozess entwickelt. Abschließend wird eine Liste der Anforderungen erstellt und die Struktur eines Modulhandbuchs analysiert.

In Kapitel 3 werden die Strukturen verschiedener Modulhandbücher untersucht und verglichen. Aus den Ergebnissen der Untersuchung soll dann ein Datenbankschema erstellt werden. Außerdem werden Entwürfe für die Benutzeroberflächen erstellt.

In Kapitel 4 wird zu dem zuvor erstellten Datenbankschema ein Backend erstellt. Anschließend werden aus den zuvor erstellten Entwürfen die Benutzeroberflächen angefertigt.

Zuletzt werden in Kapitel 5 die Ergebnisse überprüft. Dazu wird ein Interview geführt und es werden die Anforderungen auf Erfüllung geprüft. Anschließend gibt es in Kapitel 6 ein Fazit zum Stand des Systems und es wird eingeschätzt, ob es einsetzbar ist. Im Ausblick (Kapitel 7) werden Ideen für zukünftige Erweiterungen diskutiert.

# 2. Anforderungsanalyse

Die Planung des neuen Systems für Modulhandbücher beginnt mit der Anforderungsanalyse. Der Schritt der Anforderungsanalyse hat eine besondere Wichtigkeit, da es mit fortlaufender Projektlaufzeit immer aufwändiger wird, Fehler zu korrigieren oder Anpassungen vorzunehmen. [11, Seite 55] Damit diese wichtige Phase gründlich absolviert wird, folgt der Ablauf der Anforderungsanalyse den Empfehlungen von Chris Rupp und den SOPHISTen [12]. Es wird zunächst in Abschnitt 2.1 die Umgebung des Systems erkundet. Anschließend wird mithilfe eines Interviews (Abschnitt 2.2) der aktuelle Arbeitsprozess analysiert (Abschnitt 2.3) und dessen Probleme erkundet. Dann werden die Zielgruppen des neuen Systems ermittelt (Abschnitt 2.4). Aus den Zielgruppen ergeben sich Use Cases (Abschnitt 2.5) und dazugehörige Anforderungen (Abschnitt 2.6). Zuletzt wird in Abschnitt 2.7 die Struktur eines Modulhandbuches untersucht.

## 2.1. Systemumgebung

In diesem Abschnitt soll zunächst die Umgebung des neuen Systems ermittelt werden. Hierzu werden die bestehenden Anwendungen untersucht. Das neue System soll dieselben Technologien wie die bestehenden Anwendungen verwenden. Dies hat den Vorteil, dass ein Entwickler nach Einarbeitung in die Technologien alle Anwendungen weiterentwickeln kann. Damit die Anforderungen an das neue System klar definiert werden können, muss also zunächst die Umgebung untersucht werden.

In der Abteilung Informatik der Hochschule Hannover gibt es bereits mehrere Anwendungen, die ein gemeinsames Backend nutzen. Dieses Backend soll in dieser Arbeit erweitert werden, um auch den anderen Anwendungen die Auflistung der Modulhandbücher anbieten zu können. Zusätzlich wird eine neue Anwendung erstellt, die mit dem angepassten Backend kommunizieren wird.

### 2.1.1. Geplante Anwendungen

Zusätzlich zur Entwicklung dieses Projektes gibt es noch zwei weitere Anwendungen, die jedoch kein Teil dieser Arbeit sind. Im Folgenden werden die geplanten Anwendungen kurz erklärt.

Für Studierende, die planen möchten, welches Modul sie in welchem Semester erledigen, wird es die Anwendung StudyPlan geben. Hier können Studierende vom vorgeschlagenen Studienverlauf abweichen und dabei sicherstellen, trotzdem alle Module in der gewünschten Zeit zu erledigen. StudyPlan wird von dem angepassten Backend profitieren, weil StudyPlan eine stets aktuelle Auflistung aller Module inklusive deren Zeitaufwände benötigt, um die Planung des Studienverlaufs zu ermöglichen.

Die Entwicklung von StudyPlan läuft parallel zur Entwicklung dieser Bachelorarbeit – es muss also sichergestellt werden, dass es rechtzeitig nutzbare Ergebnisse gibt. Derzeit gibt es bereits einen ersten Prototyp von StudyPlan, welcher allerdings noch nicht produktiv genutzt wird.

Außerdem ist die Anwendung StudyGraph geplant, welche Studieninhalte visualisiert und somit auch die Informationen zu den angebotenen Modulen benötigt. [13]

### 2.1.2. Struktur des bestehenden Backends

Das bestehende Backend, auch StudyBase genannt, ist mithilfe des auf JavaScript basierenden Framework NestJS erstellt. NestJS legt einen Fokus auf „effiziente, zuverlässige und skalierbare serverseitige Anwendungen“ [14].

Das NestJS Backend ist modular aufgebaut. Jede in Unterabschnitt 2.1.1 beschriebene Anwendung stellt im Backend ein Modul dar. Die Module der einzelnen Anwendungen enthalten ebenfalls Module, die die einzelnen Funktionen abbilden. Beispielsweise gibt es im Modul StudyPlan das Modul *degrees*, welches alle Studiengänge verwaltet.

Zusätzlich gibt es Module, die zwischen allen Anwendungen geteilt werden. Diese Shared-Modules bieten beispielsweise Funktionen zur Benutzerverwaltung und zum Versand von E-Mails an.

Um Funktionalitäten anbieten zu können, nutzen Module verschiedene Konzepte. Damit ein Modul beispielsweise eine HTTP-GET-Anfrage bearbeiten kann, muss es eine Controller-Klasse haben. Ein Controller nimmt die Anfrage an und verarbeitet sie. Falls hierbei Daten benötigt werden, ruft der Controller eine Service-Klasse auf. Diese lädt die angefragten Daten aus der Datenbank und gibt sie an den Controller zurück. Für die Datenbankzugriffe wird Prisma genutzt. Ein Auszug der Dateistruktur ist in Abbildung 3 zu sehen.

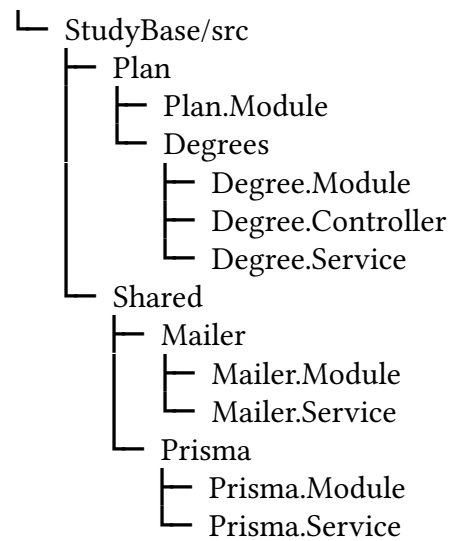


Abbildung 3: Ordnerstruktur der StudyBase

### 2.1.3. Struktur der bestehenden Datenbank

Es gibt eine schema.prisma-Datei (Code 1), in der die Struktur der relationalen Datenbank definiert ist. Somit muss kein SQL geschrieben werden, sondern es können Methoden von Prisma genutzt werden. Es gibt Tabellen für Module und Studiengänge. Die Tabellen werden von Prisma generiert. Änderungen an der Struktur müssen demnach an der schema.prisma-Datei erfolgen. Dadurch ist die Struktur der Datenbank versioniert und kann in einer Quellcodeverwaltung abgelegt werden.

In Code 1 ist beispielsweise zu sehen, wie eine Tabelle mit dem Namen „Module“ definiert ist. Die Tabelle hat unter anderem eine Spalte id, mit dem Datentyp Int und einen name mit dem Datentyp String.

```
1 model Module { prisma
2   id            Int           @id @default(autoincrement
3   name          String
4   niveau        String
5   abbreviation  String
6   description   String
7   ...
8   degreePrograms DegreeProgramToModule[]
9 }
```

Code 1: Auszug aus schema.prisma

### 2.1.4. Struktur einer beispielhaften Angular Anwendung

Eine Vorgabe dieser Arbeit ist es, Angular für die Entwicklung der Weboberfläche zu nutzen. Durch die Nutzung von Angular gibt es einen einheitlichen Techstack, da die geplanten Anwendungen (Unterabschnitt 2.1.1) ebenfalls in Angular entwickelt werden sollen.

Eine Angular Anwendung besteht aus Komponenten und Seiten. Auf einer Seite werden 0 bis n Komponenten in einer HTML-ähnlichen Struktur organisiert.

Eine Komponente ist ein wiederverwendbares Element auf einer Website – beispielsweise ein Dropdown oder eine Textbox. Eine Komponente ist ebenfalls in der HTML-ähnlichen Struktur organisiert. In der Komponente können sich HTML-Elemente und andere Angular-Komponenten befinden.

Komponenten und Seiten haben eine .HTML-Datei für die Beschreibung der Struktur, eine .SCSS-Datei für die Beschreibung des Aussehens sowie eine .TS-Datei für kleinere Logiken. Geschäftslogik wird meist in separate Service-Klassen ausgelagert – ähnlich wie schon in Unterabschnitt 2.1.2 beschrieben. [15]

## 2.2. Interview mit Studiendekan

Da die Anforderungen sowie der aktuelle Arbeitsablauf noch unklar waren, musste eine Methode gefunden werden, um beides gründlich zu durchleuchten. Hierzu soll im Folgenden ein Interview durchgeführt werden. Ein Interview hat den entscheidenden Vorteil, dass der Verlauf des Gesprächs individuell angepasst werden kann. Wenn sich neue Fragen ergeben oder Fragen nicht ausreichend beantwortet wurden, kann im Interview direkt nachgefragt werden. [12, Kapitel 6.3.3]

Das Interview wurde mit dem derzeitigen Studiendekan Prof. Dr. Felix Heine durchgeführt. Das Interview orientierte sich an den Vorschlägen der SOPHISTen. Bereits bei der Einladung zum Interview wurden einige der Vorschläge beachtet. Der Studiendekan konnte sich den Interviewort selbst auswählen und erhielt die geplanten Fragen vorab zur Einsicht. Dadurch soll für den Interviewpartner eine möglichst angenehme Umgebung geschaffen werden, was zu einer erhöhten Kooperationsbereitschaft führen soll. [12, Seite 107-109] Die übersendeten Fragen können bei Interesse im Anhang nachgelesen werden.

Die Ergebnisse des Interviews sind in den folgenden Abschnitten zu lesen. Es ergaben sich zum einen die Konkretisierung der Zielgruppen, welche in Abschnitt 2.4 zu finden sind. Zum anderen wurde der aktuelle Arbeitsprozess klar definiert sowie dessen Schwachstellen aufgezeigt (Abschnitt 2.3).

### **2.3. Analyse des aktuellen Arbeitsprozesses und Identifikation von Schwachstellen**

Der Prozess, um Modulhandbücher zu bearbeiten, hat sich bereits in der Planungsphase dieser Arbeit verändert. Bisher gab es für die Modulhandbücher Word-Dokumente, welche in einem Git-Repository verwaltet wurden. Bei Änderungen mussten jeweils das deutsche und das englische Word-Dokument bearbeitet werden. Anschließend kann es notwendig sein, die Änderungen auch im Curriculum des Studienganges (Abbildung 1) und im Anhang des besonderen Teils der Prüfungsordnung (Abbildung 2) vorzunehmen.

Die kritischste Schwachstelle sind hier Redundanzen. Die Eigenschaften eines Modulhandbuches sind an mehreren verschiedenen Stellen hinterlegt und müssen überall von Hand angepasst werden. Bei dem manuellen Eintragen von Daten können schnell Fehler auftreten, da die verschiedenen Stellen nicht automatisch synchronisiert werden. [16]

Als Vorbereitung für ein neues System wurden vom Studiendekan die zuvor genannten Word-Dokumente maschinell eingelesen, in ein JSON-Format umgewandelt und anschließend in eine PostgreSQL-Datenbank eingespielt. Weiterhin wurde ein Python-Script erstellt, welches aus den Datensätzen in der Datenbank mithilfe von LaTeX ein PDF-Dokument für die Modulhandbücher generieren kann. Die Datenbank enthält jedoch weiterhin Redundanzen und könnte daher optimiert werden.

---

## **2.4. Zielgruppen**

Im folgenden Abschnitt sollen die verschiedenen Zielgruppen eines Modulhandbuches ermittelt und definiert werden. Die Übersicht der Zielgruppen wird für die später folgende Ermittlung der Use Cases benötigt (Abschnitt 2.5). Hierdurch wird ermöglicht zu verstehen, wer das Modulhandbuch verwenden wird und welche Anforderungen die verschiedenen Gruppen haben. Zur Ermittlung wurde zum einen im ECTS User-Guide [1] recherchiert und zum anderen das Interview (Abschnitt 2.2) genutzt.

### **2.4.1. Studieninteressierte**

Der ECTS User-Guide [1] beschreibt, dass Modulhandbücher bereits bei der Wahl eines Studienganges helfen können. So können Personen, die an einem bestimmten Studiengang interessiert sind, mithilfe der Beschreibungen in den Modulhandbüchern herausfinden, welche Inhalte gelehrt werden. Modulhandbücher sind demnach eine gute Anlaufstelle, um einen ersten Eindruck zu den angebotenen Modulen eines Studienganges zu erhalten.

### **2.4.2. Studierende**

Eine weitere Zielgruppe sind Studierende. Das neue System könnte bei der Suche nach einem geeigneten Wahlpflichtfach unterstützen. Auch könnten Studierende mithilfe der Modulbeschreibungen verstehen, welche Inhalte in einem Modul gelernt werden und welche Voraussetzungen es gibt. Dadurch können Studierende einschätzen, ob sie genug Vorwissen für ein bestimmtes Modul haben. Weiterhin können Studierende mithilfe der Modulhandbücher zu jedem Modul einen Überblick über die zu erbringende Arbeitszeit erhalten sowie Informationen zu den Prüfungsleistungen finden.

### **2.4.3. Modulverantwortliche und Studiengangverantwortliche**

Aus dem Interview mit dem Studiendekan (Abschnitt 2.2) geht hervor, dass es neben den Studierenden noch andere zu betrachtende Zielgruppen gibt. Für die Bearbeitung von Modulhandbüchern sind an der Hochschule Hannover verschiedene Personengruppen zuständig. Zum einen gibt es die studiengangverantwortliche Person (SVP). Diese ist für die Veröffentlichung des Dokumentes verantwortlich. Die SVP ist nicht dafür zuständig, die Inhalte der einzelnen Modulbeschreibungen anzupassen. Für diese Anpassungen gibt es die Modulverantwortlichen. Jedes Modul hat eine Person, die die Informationen der Modulbeschreibung aktuell halten soll und gleichzeitig Ansprechpartner für Fragen ist. In der Abteilung Informatik ist aktuell der Studiendekan gleichzeitig auch Studiengangverantwortlicher. Modulverantwortliche sind die Professoren und Dozierenden der Abteilung.

## 2.5. Use Cases

Im folgenden Abschnitt werden die Ergebnisse aus dem Interview und der Recherche im ECTS User Guide [1] verwendet, um aufzuzeigen, welche Funktionen die einzelnen Akteure im neuen System verwenden können.[12, Seite 192] Hierzu werden Use Cases bestimmt, damit im nächsten Abschnitt (Abschnitt 2.6) daraus Anforderungen abgeleitet werden können. Die Use Cases wurden hergeleitet, indem die Bedürfnisse und Aufgaben der Akteure analysiert und mit den im ECTS User Guide beschriebenen Funktionen abgeglichen wurden. So soll sichergestellt werden, dass alle zuvor bestimmten Akteure ihre Aufgaben vollständig mit dem neuen System erledigen können.

<b>Name</b>	<b>UC1 - Informationen zu Studiengang</b>
<b>Kurzbeschreibung</b>	Der Use Case beschreibt, wie eine studieninteressierte Person einen Überblick über alle Module eines Studiengangs erhalten kann. Die Person möchte hierzu ein PDF erhalten, in dem alle Module und deren Eigenschaften aufgelistet sind.
<b>Akteur</b>	Studieninteressierte Person (Unterabschnitt 2.4.1)
<b>Vorbedingungen</b>	Der Studiengang ist im System hinterlegt.
<b>Hauptszenario</b>	<ol style="list-style-type: none"><li>1. Die Person besucht die Website der Hochschule Hannover</li><li>2. Der gesuchte Studiengang wird aufgerufen</li><li>3. Die Seite des Studiengangs enthält eine Verlinkung zum System StudyModules</li><li>4. Die Person klickt im neuen System auf „PDF anzeigen“</li><li>5. Das System zeigt das PDF an</li></ol>

<b>Name</b>	<b>UC2 - Informationen zu Modul</b>
<b>Kurzbeschreibung</b>	Der Use Case beschreibt, wie eine studierende Person Informationen zu einem bestimmten Modul erhalten kann. Dabei können die angebotenen Module mithilfe einer Filterfunktion oder Suchfunktion durchsucht werden. Zum Beispiel könnte die studierende Person auf der Suche nach einem spannenden Wahlpflichtfach sein.
<b>Akteur</b>	Studierende Person (Unterabschnitt 2.4.2)
<b>Vorbedingungen</b>	Der Studiengang ist im System hinterlegt.

---

<b>Hauptzenario</b>	<ol style="list-style-type: none"> <li>1. Die Person besucht die Website der Hochschule Hannover</li> <li>2. Der gesuchte Studiengang wird aufgerufen</li> <li>3. Die Seite des Studiengangs enthält eine Verlinkung zum System StudyModules</li> <li>4. Im neuen System werden alle Module aufgelistet</li> <li>5. Die Person filtert nach allen Wahlpflichtfächern</li> <li>6. Die Person öffnet ein Modul</li> <li>7. Informationen zum ausgewählten Modul werden in einer modernen Oberfläche angezeigt</li> </ol>
---------------------	--

<b>Name</b>	UC3 - Modul bearbeiten
<b>Kurzbeschreibung</b>	Der Use Case beschreibt, wie Informationen eines Moduls verändert werden können. Das System prüft dabei, ob die angegebenen Informationen plausibel sind. Beispielsweise müssen Zeitaufwände und ECTS zusammenpassen.
<b>Akteur</b>	Modulverantwortliche Person (Unterabschnitt 2.4.3)
<b>Vorbedingungen</b>	Erfolgreich mit einem Account eingeloggt, der das ausgewählte Modul bearbeiten darf
<b>Hauptzenario</b>	<ol style="list-style-type: none"> <li>1. User drückt auf „Bearbeiten“</li> <li>2. System wechselt in den Bearbeitungsmodus</li> <li>3. User aktualisiert Texte in den Eingabefeldern</li> <li>4. User drückt auf „Speichern“</li> <li>5. System prüft, ob Änderungen plausibel sind</li> <li>6. System wechselt in den Anzeigemodus</li> </ol>

<b>Name</b>	UC4 - Datensatz anlegen
<b>Kurzbeschreibung</b>	Der Use Case beschreibt, wie ein neuer Datensatz angelegt werden kann. Das System prüft dabei, ob die angegebenen Informationen plausibel sind. Ein Datensatz kann zum Beispiel ein Modul sein, ein Teilmodul oder ein Studiengang. Zur Übersichtlichkeit wurde nicht für jede Art ein einzelner Use Case erstellt.
<b>Akteur</b>	Studiengangsverantwortliche Person
<b>Vorbedingungen</b>	Erfolgreich mit einem Account eingeloggt, der Datensätze anlegen darf
<b>Hauptszenario</b>	Am Beispiel eines Moduls: <ol style="list-style-type: none"><li>1. User drückt auf Module</li><li>2. User drückt auf „Hinzufügen“</li><li>3. System wechselt in den Bearbeitungsmodus</li><li>4. User füllt Eingabefelder aus</li><li>5. User drückt auf „Speichern“</li><li>6. System prüft, ob Angaben plausibel sind</li><li>7. System zeigt „Modul erfolgreich angelegt“</li></ol>

<b>Name</b>	UC5 - Änderungen rückgängig machen
<b>Kurzbeschreibung</b>	Der Use Case beschreibt, wie Änderungen an einem Modul rückgängig gemacht werden können. Damit können sowohl eigene Änderungen als auch die Änderungen anderer User zurückgesetzt werden.
<b>Akteur</b>	Studiengangsverantwortliche Person
<b>Vorbedingungen</b>	Erfolgreich mit einem Account eingeloggt, der Module bearbeiten darf
<b>Hauptszenario</b>	<ol style="list-style-type: none"><li>1. User öffnet Modul</li><li>2. User drückt auf „Änderungs-Historie“</li><li>3. System zeigt Änderungen</li><li>4. User drückt auf „Änderungen rückgängig machen“</li><li>5. System zeigt „Änderungen rückgängig gemacht“</li></ol>

<b>Name</b>	UC6 - Prüfungsordnung verifizieren
<b>Kurzbeschreibung</b>	Der Use Case beschreibt, wie das System bei der Erstellung der Prüfungsordnung unterstützen kann. Die hierzu erforderliche Tabelle kann im System generiert werden, um zu vergleichen, ob alle Daten korrekt hinterlegt sind.
<b>Akteur</b>	Studiengangsverantwortliche Person
<b>Vorbedingungen</b>	Alle Module des Studiengangs angelegt
<b>Hauptzenario</b>	<ol style="list-style-type: none"> <li>1. User erstellt die Tabelle manuell</li> <li>2. User öffnet die generierte Tabelle im System</li> <li>3. User vergleicht beide Tabellen, um sicherzustellen, dass sie korrekt sind</li> </ol>

Die Auflistung der Use Cases bildet eine Grundlage für die folgenden Abschnitte. Aus den Use Cases können nun konkrete Anforderungen ermittelt werden. Sicherlich können verschiedene Menschen noch weitere Ideen zur Verwendung des Systems haben, jedoch sollte diese Auflistung die wichtigsten Punkte umfassen, um ein System abzubilden, welches den aktuellen Workflow verbessert. Es ist wichtig zu betonen, dass sich Anforderungen stetig verändern können und dass es möglich ist in Zukunft auch andere Use Cases durch das System abzudecken. Das System soll deshalb so implementiert werden, dass zukünftige Änderungen einfach möglich sind.

## 2.6. Anforderungen

Als Nächstes sollen konkrete Anforderungen an das zukünftige System aufgestellt werden. Diese Anforderungen sollen zum einen dabei helfen, in der Entwurfs- und Implementierungsphase (Kapitel 3, Kapitel 4) konkrete Vorgaben zu haben, die abgearbeitet werden können, und zum anderen in Kapitel 5 ermitteln zu können, ob das System einsetzbar ist.

Die Anforderungen an das System leiten sich aus den Use Cases, einer ISO-Norm und der Systemumgebung ab. Diese lassen sich in zwei Kategorien unterteilen: funktionale Anforderungen (basierend auf den Use Cases) und nicht-funktionale Anforderungen (abgeleitet aus der ISO-Norm und der Systemumgebung).

Um die Herkunft der Anforderungen klar darzustellen, werden sie in funktionale und nicht-funktionale Anforderungen gegliedert. Obwohl diese Trennung fachlich nicht zwingend erforderlich ist, dient sie der besseren Übersichtlichkeit der folgenden Auflistung.

Jede Anforderung in den folgenden Auflistungen enthält entweder das Wort „muss“, „sollte“ oder „könnte“. [12, Kapitel 1.5.2] Damit wird zwischen Anforderungen unterschieden, die zwingend erforderlich sind (muss), Anforderungen, die sehr sinnvoll sind (sollte), und Anforderungen, die nicht zwingend erforderlich sind, aber die Nutzer begeistern würden (könnte). Eine weitere Priorisierung wird an dieser Stelle nicht benötigt, sondern kann bei Bedarf erfolgen.

### 2.6.1. Funktionale Anforderungen

Die funktionalen Anforderungen ergeben sich aus den zuvor ermittelten Use Cases (Abschnitt 2.5). Hierbei wurde überlegt, welche Anforderungen erfüllt sein müssen, um einen Use Case vollständig abbilden zu können.

In Use Case 1 ist beschrieben, wie eine nicht angemeldete Person (NP) Informationen zu einem Studiengang erhalten möchte. Damit die Übersicht eines Studienganges auf der Website der Hochschule verlinkt werden kann, muss das System einen Link bereitstellen können, der direkt zur Übersicht führt und sich nicht verändert. Außerdem muss es eine Funktion geben, um das PDF mit dem Modulverzeichnis des Studienganges zu öffnen.

#### Aus Use Case 1 ergeben sich folgende Anforderungen:

##### **F1 LINK ZUR ÜBERSICHTSSEITE ANBIETEN**

System muss einen statischen Link zur Übersicht eines Studienganges bereitstellen.

##### **F2 PDF ANZEIGEN**

NP muss ein PDF mit allen Modulbeschreibungen ansehen können.

In Use Case 2 ist beschrieben, wie eine Person Informationen zu einem Modul erhalten möchte. Das System muss hierzu alle Module des ausgewählten Studienganges auflisten. Die Module müssen auswählbar sein, damit weitere Details angezeigt werden können. Im genannten Beispiel ist die Person auf der Suche nach einem Wahlpflichtmodul. Für diesen Fall muss das System die Möglichkeit bieten, nach bestimmten Informationen zu filtern. Denkbar wären Filter für die Informationen „Wahlpflichtfach“, „Semester“, „Erreichbare Credits“.

**☰ Aus Use Case 2 ergeben sich folgende Anforderungen:****F3 MODULE AUFLISTEN**

NP muss eine Liste aller Module sehen können.

**F4 FILTERFUNKTION**

NP sollte Filter nutzen können, um das gesuchte Modul zu finden.

**F5 SUCHFUNKTION**

NP sollte eine Suchfunktion nutzen können, um das gesuchte Modul zu finden.

**F6 MODULE ANZEIGEN**

NP muss die Details eines Moduls ansehen können.

In Use Case 3 ist beschrieben, wie eine Person Informationen zu einem Modul bearbeiten möchte. Damit dies nur vom genannten Akteur erledigt werden kann, wird eine Login-Funktion benötigt. Aus der Login-Funktion ergeben sich weitere Anforderungen. So kann ein Logout-Button sinnvoll sein und es könnte praktisch sein, das eigene Passwort ändern zu können. Weiterhin könnte es hilfreich sein, wenn die eingegebenen Informationen auf Plausibilität überprüft werden. Denkbar wäre beispielsweise eine Prüfung, ob die angegebenen ECTS mit dem angegebenen Zeitaufwand zusammenpassen.

**☰ Aus Use Case 3 ergeben sich folgende Anforderungen:****F7 LOGIN**

Ein User muss sich anmelden können.

**F8 LOGOUT**

Ein User sollte sich ausloggen können.

**F9 PASSWÖRTER ZURÜCKSETZEN**

SVP sollte Passwörter zurücksetzen können.

**F10 EIGENES PASSWORT ZURÜCKSETZEN**

SVP sollte das eigene Passwort zurücksetzen können.

**F11 MODULE BEARBEITEN**

Modulverantwortliche Person muss Module bearbeiten können, für die sie als Ansprechpartner hinterlegt ist.

**F12 PLAUSIBILITÄTSCHECKS BEI MODULEN**

System sollte Änderungen an Modulen auf Plausibilität prüfen.

In Use Case 4 ist beschrieben, dass die studiengangverantwortliche Person Datensätze erstellen können muss. Für die verschiedenen Entitäten im neuen System muss

es dementsprechend jeweils eine Bearbeitungsmaske geben. Im System müssen die Datensätze sowohl erstellbar als auch bearbeitbar sein. Außerdem muss man Datensätze löschen können. Aus dem Interview mit dem Studiendekan hat sich außerdem ergeben, dass Studiengänge ausblendbar sein müssen, damit alte Prüfungsordnungen versteckt werden können. Diese werden von Studierenden nicht benötigt, weil darin enthaltene Veranstaltungen möglicherweise nicht mehr angeboten werden. Die veralteten Studiengänge sollen jedoch nicht direkt gelöscht werden, damit der Studiendekan bei Bedarf auf alte Modulbeschreibungen zugreifen kann. Für die Erstellung einer neuen Prüfungsordnung schlägt der Studiendekan eine Funktion vor, die bestehende Studiengänge oder Module duplizieren kann, da es manchmal zwischen den Prüfungsordnungen nur geringfügige Änderungen gibt.

### ☰ Aus Use Case 4 ergeben sich folgende Anforderungen:

#### **F13 MODULE VERWALTEN**

SVP muss Module verwalten (anlegen, bearbeiten, löschen) können.

#### **F14 MODULE DUPLIZIEREN**

SVP sollte Module duplizieren können.

#### **F15 STUDIENGÄNGE VERWALTEN**

SVP sollte Studiengänge verwalten können.

#### **F16 STUDIENGÄNGE DUPLIZIEREN**

SVP sollte Studiengänge duplizieren können.

#### **F17 STUDIENGÄNGE AUSBLENDEN**

SVP sollte Studiengänge ausblenden können.

#### **F18 AUSGEBLENDETE STUDIENGÄNGE ANSEHEN**

SVP sollte ausgeblendete Studiengänge ansehen können.

#### **F19 BENUTZER VERWALTEN**

SVP sollte User verwalten können.

#### **F20 TEILMODULE VERWALTEN**

SVP muss Teilmodule verwalten können.

#### **F21 VORAUSSETZUNGEN VERWALTEN**

SVP muss Voraussetzungen verwalten können.

In Use Case 5 ist beschrieben, dass die Änderungen an einem Modul rückgängig gemacht werden können. Hierzu müssen diese zunächst angezeigt werden. Diese Funktion ist außerdem für Teilmodule und auch alle anderen Arten von Datensätzen sinnvoll.

☰ **Aus Use Case 5 ergibt sich folgende Anforderung:**

**F22 ÄNDERUNGEN ANZEIGEN (MODUL)**

SVP sollte sich einzelne Änderungen an einem Modul anzeigen lassen können.

**F23 ÄNDERUNGEN WIDERRUFEN (MODUL)**

SVP könnte einzelne Änderungen an einem Modul rückgängig machen.

**F24 ÄNDERUNGEN ANZEIGEN**

SVP könnte sich einzelne Änderungen an beliebigen Datensätzen anzeigen lassen können.

**F25 ÄNDERUNGEN WIDERRUFEN**

SVP könnte einzelne Änderungen an beliebigen Datensätzen rückgängig machen.

In Use Case 6 ist beschrieben, wie die studiengangverantwortliche Person den Anhang der Prüfungsordnung erstellt. Da die Prüfungsordnung ein wichtiges Dokument ist, soll diese auch in Zukunft zunächst manuell erstellt werden. Die im Anhang der Prüfungsordnung enthaltene Tabelle kann dann im System generiert werden und kann mit der manuell erstellten Tabelle verglichen werden. Hiermit ist dann sichergestellt, dass die Daten im neuen System und in der Prüfungsordnung übereinstimmen.

☰ **Aus Use Case 6 ergibt sich folgende Anforderung:**

**F26 ANHANG DER PRÜFUNGSORDNUNG**

SVP könnte sich die Auflistung aller Module als Tabelle anzeigen lassen, um sie mit dem Anhang der Prüfungsordnung vergleichen zu können.

### 2.6.2. Nicht-Funktionale Anforderungen

Die nicht-funktionalen Anforderungen ergeben sich aus einem Brainstorming unter der Berücksichtigung der ISO-Norm ISO/IEC 25000 [12, Kapitel 12] und ergeben sich aus den Bedingungen aus Abschnitt 2.1.

Die ISO-Norm 25000 beschreibt verschiedene Merkmale, die zur Messung von Softwarequalität genutzt werden können. Um eine gute Softwarequalität zu erreichen, sollten aus allen Merkmalen konkrete Anforderungen an das neue System abgeleitet werden. Die Merkmale sind jeweils in Submerkmale unterteilt. Nicht jedes Submerkmal ist für das neue System relevant, jedoch sollten möglichst viele Submerkmale in Anforderungen übersetzt werden, um eine gute Qualität sicherzustellen.

Das Merkmal *Änderbarkeit* besteht aus den Submerkmalen *Analysierbarkeit*, *Modifizierbarkeit*, *Stabilität* und *Testbarkeit* [12, 12.4.1]. Damit Software diese Kriterien erfüllt, muss der Quellcode eine hohe Qualität aufweisen. Daher soll das System einigen Prinzipien folgen, die in von R. C. Martin beschrieben sind. [17] Der Quellcode soll so geschrieben sein, dass mit geringem Aufwand jede Klasse und jede Methode verstanden werden kann. Alle Teile des Codes sollen selbsterklärend und möglichst kurz sein. Komplexe Abläufe sollen in Teilabläufe aufgeteilt werden, sodass Methoden und Klassen eine gewisse Größe und Komplexität nicht überschreiten und somit innerhalb kurzer Zeit von nachfolgenden Entwicklern verstanden werden können. [17, Kapitel 1] Der Quellcode kann dadurch auf eine effiziente Art von Entwicklern analysiert werden (*Analysierbarkeit*). Durch den Einsatz von Dependency Injection können Abhängigkeiten ausgetauscht werden (*Modifizierbarkeit*). Durch das gezielte Einsetzen von Fehlermeldungen und Try-Catch-Blöcken wird für eine gute *Stabilität* des Systems gesorgt. [17, Kapitel 7] Abschließend ergeben sich aus kleinen Komponenten mit genau einer Verantwortlichkeit und wenigen Abhängigkeiten gut testbare Einheiten, die mit automatisierten Tests getestet werden könnten. [17, Kapitel 9]

## ☰ Änderbarkeit

### **N1 MODULARITÄT**

Der Quellcode des Systems sollte aus Komponenten bestehen.

### **N2 WIEDERVERWENDBARKEIT**

Einzelne Komponenten könnten wiederverwendbar sein.

### **N3 HOHE KOHÄRENZ**

Die Komponenten sollten genau eine Verantwortlichkeit haben.

### **N4 LOSE KOPPLUNG**

Die Komponenten dürfen nicht zu viele Abhängigkeiten haben.

### **N5 KOMPLEXITÄT**

Methoden und Klassen sollen eine geringe Komplexität haben.

### **N6 DEPENDENCY INJECTION**

Durch die Nutzung von Dependency Injection sollen Abhängigkeiten austauschbar sein.

### **N7 STABILITÄT**

Mithilfe von Fehlercodes und Try-Catch-Anweisungen sollen Fehler abgefangen werden.

### **N8 UNITTESTS**

Geschäftslogik könnte mithilfe von Unittests automatisiert getestet werden.

### **N9 E2E-TESTS**

System könnte mithilfe von e2e-Tests automatisiert getestet werden.

Das Merkmal *Benutzbarkeit* besteht aus den Submerkmalen *Verständlichkeit*, *Erlernbarkeit*, *Bedienbarkeit*, *Attraktivität* und *Konformität*. [12, 12.4.1] Für die Ermittlung der folgenden Anforderungen wurde in einem Brainstorming überlegt, wie die Submerkmale der Benutzbarkeit in konkrete Anforderungen übersetzt werden können. Beispiel: Damit eine Oberfläche verständlich ist, sollte sie selbsterklärend sein und nicht das Lesen eines Benutzerhandbuches erfordern (*Verständlichkeit*).

Die folgenden Anforderungen sind kein 1:1-Abbild der zukünftigen Anwendung, sondern legen eher die Grundprinzipien fest, die bei der Entwicklung beachtet werden sollen. Eine vollständige Spezifikation wäre an dieser Stelle sehr aufwändig und ist daher nicht notwendig.

### ☰ Benutzbarkeit

#### **N10 AKTUELLER PFAD**

System könnte anzeigen, welcher Pfad aufgerufen wurde (z. B. Fakultät->Studiengang->Modul).

#### **N11 RÜCKFRAGEN**

Vor dem Löschen eines Elements muss eine Rückfrage erscheinen.

#### **N12 WIEDERHERSTELLBARKEIT**

Gelöschte Elemente könnten wiederherstellbar sein.

#### **N13 LADEBALKEN**

Ladezeiten >1s sollten einen Ladebalken zeigen.

#### **N14 VERSTÄNDLICHKEIT**

Fehlermeldungen sollten verständlich sein.

#### **N15 LÖSUNG ANBIETEN**

Fehlermeldungen könnten eine Lösung anbieten.

#### **N16 RESPONSIVE**

Das System könnte auf verschiedenen Displaygrößen nutzbar sein.

#### **N17 EINGABEMETHODEN**

Das System könnte verschiedene Eingabemethoden unterstützen.

#### **N18 SELBSTERKLÄREND**

Das System sollte selbsterklärend sein und kein Handbuch benötigen.

Das Merkmal *Effizienz* besteht aus den Submerkmalen *Zeitverhalten*, *Verbrauchsverhalten* und *Konformität*. [12, 12.4.1] Damit das System eine gute Effizienz hat, müssen alle Anfragen performant gestaltet werden. In den Anforderungen werden Grenzwerte gesetzt, die das System erreichen soll. Falls einzelne Seiten viele Daten laden müssen, sodass die gesetzten Ziele nicht erreicht werden können, müssen diese Daten im Hintergrund geladen werden, sodass die Seite selbst geladen wird und fehlende Daten dann nachträglich eingesetzt werden können. Unter dem Begriff der Effizienz könnte auch die Effizienz des Entwicklungsprozesses oder die Effizienz der Aufgabenerledigung verstanden werden, weshalb zu diesen Gebieten auch Anforderungen erstellt wurden.

### ☰ Effizienz

#### **N19 STARTZEIT FRONTEND**

Jede Seite im Frontend sollte innerhalb einer Sekunde geladen sein.

#### **N20 STARTZEIT BACKEND**

Das Backend sollte im kritischen Fehlerfall innerhalb einer Minute neu starten.

#### **N21 DEPLOYMENT**

Das Deployment könnte automatisiert sein.

#### **N22 EFFIZIENZ DER AUFGABENERLEDIGUNG**

Jeder Schritt im Use Case sollte mit möglichst wenigen Klicks absolvierbar sein.

Das Merkmal *Funktionalität* besteht aus den Submerkmalen *Angemessenheit*, *Richtigkeit*, *Interoperabilität*, *Sicherheit* und *Ordnungsmäßigkeit*. [12, 12.4.1] Da die Modulhandbücher bisher in Englisch und Deutsch bereitstehen, muss diese Funktionalität bestehen bleiben. Für die Zukunft wäre auch die Einführung weiterer Sprachen denkbar, weshalb die Mehrsprachenfähigkeit vorbereitet werden sollte. Um den Arbeitsaufwand gering zu halten, sollte das System sinnvolle Eingaben vorschlagen und das generierte PDF sollte dem ursprünglichen PDF ähneln. Außerdem muss sichergestellt sein, dass bestimmte Endpunkte nur von autorisierten Nutzern verwendet werden können.

### ☰ Funktionalität

#### **N23 ZWEI SPRACHEN**

Das System muss in Englisch und Deutsch verfügbar sein.

#### **N24 MEHRSPRACHENFÄHIGKEIT**

Das System sollte für beliebig viele Sprachen erweiterbar sein.

#### **N25 AUSWAHLMÖGLICHKEITEN**

System sollte möglichst oft vorschlagen, welche Eingaben sinnvoll wären.

#### **N26 ÄHNLICHKEIT ZUM URSPRÜNGLICHEN PDF**

Das generierte PDF sollte dem ursprünglichen PDF ähneln.

#### **N27 SICHERHEIT**

Es muss verhindert werden, dass nicht autorisierte Benutzer datenverändernde Endpunkte nutzen.

Das Merkmal *Übertragbarkeit* besteht aus den Submerkmalen *Anpassbarkeit*, *Installierbarkeit*, *Koexistenz*, *Austauschbarkeit* und *Konformität*. [12, 12.4.1] Eine vereinfachte Installierbarkeit könnte durch den Einsatz von Podman- oder Docker-Containern erreicht werden. Damit zukünftige Anwender das System installieren können, wird zudem eine Dokumentation benötigt. Weiterhin sollte das System aus austauschbaren

Komponenten bestehen. Es ist beispielsweise denkbar, in Zukunft eine andere Datenbank zu nutzen oder andere Kompilierungsserver einzusetzen.

### ☰ Übertragbarkeit

#### **N28 DOKUMENTATION ZUR INSTALLATION**

Es sollte dokumentiert sein, wie das System installiert wird.

#### **N29 CONTAINER**

Es könnten Podman-Container genutzt werden, um die Komponenten des Systems bereitzustellen.

#### **N30 AUSTAUSCHBARKEIT**

Einzelne Komponenten sollten austauschbar sein (Datenbank, Kompilierungsserver, Frontend).

Das Merkmal *Zuverlässigkeit* besteht aus den Submerkmalen *Reife*, *Fehlertoleranz*, *Robustheit*, *Wiederherstellbarkeit* und *Konformität*. Das System sollte mit Fehlern umgehen können. Diese sollten entweder abgefangen oder dem Benutzer mitgeteilt werden. Besser wäre es, wenn während der Erledigung der Use Cases keine Fehler auftreten würden. Gegen den Schutz vor Angriffen könnten Rate-Limits ein sinnvoller Mechanismus sein, um beispielsweise nach mehreren fehlgeschlagenen Loginversuchen weitere Versuche zu unterbinden.

### ☰ Zuverlässigkeit

#### **N31 STABILITÄT**

Das System muss bei auftretenden Fehlern weiterhin funktionieren / sich selbst wiederherstellen.

#### **N32 REIFE**

Das System sollte beim Erledigen der Use Cases keine Fehler erzeugen.

#### **N33 ROBUSTHEIT**

Das System sollte Rate-Limits nutzen, um bei größeren Lasten gegebenenfalls Anfragen zu blockieren.

In Abschnitt 2.1 sind die bereits bestehenden Anwendungen beschrieben. Aus diesen Erkenntnissen ergeben sich einige technische Anforderungen, die bei der Entwicklung beachtet werden müssen.

### ☰ Technische Anforderungen

**N34 NEUE ANWENDUNG**

Das Frontend muss eine neue Anwendung sein.

**N35 TECHNOLOGIEN IM FRONTEND**

Das Frontend muss Angular nutzen.

**N36 BESTEHENDE ANWENDUNG**

Das bestehende Backend muss erweitert werden.

**N37 TECHNOLOGIEN IM BACKEND**

Das Backend muss Prisma und NestJS nutzen.

**N38 BESTEHENDE DATENBANK**

Die bestehende Datenbank muss erweitert werden.

### 2.6.3. Ergebnis

Das Ergebnis dieses Abschnittes ist eine Auflistung der Anforderungen an das neue System. Diese sollte alle zuvor aufgestellten Use Cases abdecken. Außerdem gibt es weitere Anforderungen, die eine gute Softwarequalität gewährleisten sollen und die Rahmenbedingungen der späteren Entwicklung bestimmen. In Kapitel 5 kann anhand der Auflistung überprüft werden, ob das neue System für die produktive Verwendung einsatzbereit ist.

## 2.7. Struktur eines Modulhandbuches

In diesem Unterabschnitt soll die Struktur eines Modulhandbuches analysiert werden, um herauszufinden, wie die spätere Datenbank aufgebaut sein muss. Damit ein Datenbankschema erstellt werden kann, müssen die verschiedenen Entitäten ermittelt werden, die Beziehung zwischen den Entitäten und die Attribute für jede Entität.

Die Modulhandbücher der Abteilung Informatik haben für alle drei Studiengänge (BIN, MDI und MIN) dieselbe Struktur. Es gibt eine Aufteilung in Module und Teilmodule. Ein Modul kann dabei 0 bis n verschiedene Teilmodule haben und jedes Teilmodul kann zu 1 bis n Modulen gehören. Teilmodule können studiengangübergreifend mit Modulen verknüpft werden. *Beispiel:* Es gibt das Modul „MDI-103 Grundlagen der Informatik“ und das Modul „BIN-103 Grundlagen der Informatik“. Beide verweisen auf das Teilmodul „BIN-103-01 Grundlagen der Informatik“. Somit können studiengangübergreifende Module abgebildet werden. In der Regel hat an der Abteilung Informatik jedes Modul genau ein Teilmodul und umgekehrt gehört in der Regel jedes Teilmodul zu genau einem Modul.

### 2.7.1. Moduleigenschaften

Das Modul enthält zunächst grundlegende Informationen. Diese werden aufgelistet, um ein besseres Verständnis der benötigten Datenstruktur zu erhalten. Jedes aufgelistete Feld soll später in Unterabschnitt 4.1.1 implementiert werden, daher ist es notwendig, die benötigten Felder und deren Inhalt herauszufinden.

#### **E1 TITEL**

Zusammengesetzt aus einem Kürzel des Studiengangs, einer eindeutigen Zahl und dem Namen des Moduls.

**Beispiel:** BIN-100 Mathematik 1

#### **E2 UNTERTITEL**

Ein alternativer, ggf. etwas präziserer Name des Moduls sowie ein Kürzel. In den aktuell veröffentlichten Handbüchern ist hier oft nur das Kürzel angegeben.

**Beispiel:** Mathematische Grundlagen der Informatik (BIN-MAT1) oder C/C++ (BIN-PR3)

#### **E3 MODULNIVEAU**

Hier ist angegeben, ob das Modul ein Grundlagenmodul oder Vertiefungsmodul ist.

#### **E4 PFLICHT / WAHLPFLICHT**

Hier ist angegeben, ob das Modul ein Pflichtmodul oder Wahlpflichtmodul ist. Studierende müssen alle Pflichtmodule absolvieren und müssen eine bestimmte Anzahl an selbst ausgewählten Wahlpflichtmodulen absolvieren.

#### **E5 TEILMODULE**

Eine Auflistung aller Teilmodule.

#### **E6 VERANTWORTLICHE(R)**

Name der verantwortlichen Person. Diese Person ist für die Bearbeitung des Modulhandbuches zuständig.

**Beispiel:** Wohlfeil, Stefan, Prof. Dr.

#### **E7 CREDITS**

Anzahl erreichbarer ECTS bei Absolvierung dieses Moduls. Eine Zahl zwischen 2 (Englisch) und 30 (z. B. Masterarbeit).

**E8 PRÄSENZSTUNDEN / SELBSTSTUDIUM**

Aufwand des Studiums, aufgeteilt nach der Zeit, die in der Hochschule verbracht wird, und der Zeit, die im Selbststudium verbracht wird (Arbeit an Übungen, Prüfungsvorbereitung ...).

**Beispiel:** 68 h / 112 h

**E9 STUDIENSEMESTER**

Vorgeschlagenes Semester. Anhand dieser Information wird das Curriculum generiert (Abbildung 1).

**Beispiel:** 4-6

**E10 MODULDAUER**

Hier ist angegeben, wie lange es dauert, das Modul zu absolvieren.

**Beispiel:** 1 Semester

**E11 VORAUSSETZUNGEN NACH PRÜFUNGSORDNUNG**

Für die Absolvierung des Moduls zwingend erforderliche Voraussetzungen.

**Beispiel:** Alle Modulprüfungen des 1. bis 3. Semesters

**E12 EMPFOHLENE VORAUSSETZUNGEN**

Für die Absolvierung des Moduls empfohlene Voraussetzungen.

**Beispiel:** Alle Module der Semester 1 bis 3 sowie BIN-112 und BIN-202

**E13 STUDIEN-/ PRÜFUNGSLEISTUNGEN**

Eine kommasetrennte Auflistung der zu erbringenden Leistungen.

**Beispiel:** Referat (Hausarbeit plus Präsentation/Vortrag), Anwesenheitspflicht

**E14 ANGESTREBTE LERNERGEBNISSE**

Eine stichpunktartige, kommasetrennte Auflistung der Kompetenzen, die in diesem Modul erworben werden.

**Beispiel:** Die Studierenden sind in der Lage, dreidimensionale Objekte zu gestalten, zu bewegen und zueinander in Beziehung zu setzen.

**E15 GRUPPE**

Jedes Modul gehört zu einer Gruppe. Anhand der Gruppe werden die Module im PDF gedruckt.

**Beispiel:** Wahlpflichtmodule 2. Studienabschnitt

### 2.7.2. Teilmoduleigenschaften

Die Teilmodule haben ebenfalls die Informationen: E2, E6, E7, E8, E9, E12, E13 und E14. Zusätzlich gibt es noch weitere benötigte Felder.

#### **E16 TITEL**

Der Titel des Teilmoduls setzt sich zusammen aus dem Kürzel des übergeordneten Moduls, einer eigenen Nummer und dem Namen des Teilmoduls.

**Beispiel:** BIN-100-01 Mathematik 1

#### **E17 SPRACHE**

Hier ist angegeben, in welcher Sprache die Veranstaltung stattfindet.

**Beispiel:** „nach Vereinbarung“ oder „deutsch“

#### **E18 ZUORDNUNG ZU CURRICULA**

Eine kommasetrennte Auflistung aller Studiengänge, in denen dieses Teilmodul verwendet wird.

**Beispiel:** BIN, MDI

#### **E19 VERANSTALTUNGSART, SWS**

Eine Kurzbeschreibung zum Ablauf der Veranstaltung sowie deren Dauer in Semesterwochenstunden.

**Beispiel:** Vorlesung mit Übung, 4 SWS

#### **E20 EMPFEHLUNGEN ZUM SELBSTSTUDIUM**

Hier stehen Vorschläge, wie der Lehrinhalt im Selbststudium vertieft werden kann.

**Beispiel:** Aufbereitung der Lehrveranstaltung anhand von eigenen Projekten

#### **E21 GRUPPENGROÖBE**

Üblicherweise eine Zahl zwischen 1 (Bachelorarbeit) und 100 (Mathematik 1).

#### **E22 INHALT**

Die Inhalte der Veranstaltung kurz zusammengefasst.

**Beispiel:** Neue und aktuelle Trends im Bereich Betriebssysteme und Netze

#### **E23 ANFORDERUNGEN DER PRÄSENZZEIT**

Hier ist kurz beschrieben, was von Studierenden in der Präsenzzeit erwartet wird.

**Beispiel:** Regelmäßige und aktive Teilnahme.

#### **E24 ANFORDERUNGEN DES SELBSTSTUDIUMS**

Hier ist kurz beschrieben, was von Studierenden außerhalb der Präsenzzeit erwartet wird.

**Beispiel:** Vor- und Nachbereitung

#### **E25 LITERATUR**

Eine Auflistung empfohlener Literatur zur Vertiefung des behandelten Themas.

**Beispiel:** Skript zur Vorlesung

Reges, S., Stepp, M.: Building Java Programs, Prentice Hall

## **2.8. Zwischenfazit**

In diesem Kapitel wurden mithilfe verschiedener Methoden die Schwachstellen des alten Prozesses ermittelt und daraus im Anschluss Anforderungen an das neue System ermittelt. Hierbei half die Aufstellung von Zielgruppen und Anwendungsfällen. Die aufgestellten Anforderungen helfen bei der folgenden Planung der Anwendung und werden auch später im Kapitel 5 hilfreich sein, um zu ermitteln, ob das neue System eingesetzt werden kann. Die zum Ende des Kapitels herausgearbeitete Struktur der Modulhandbücher kann im Folgenden zur Entwicklung einer passenden Datenstruktur helfen.

# 3. Entwurf

In diesem Kapitel werden die gesammelten Anforderungen aus Kapitel 2 verwendet, um die Implementierung in Kapitel 4 vorzubereiten. Hierzu wird zunächst ein Datenbankschema erstellt, um die Datenbank fürs Backend anzupassen (Abschnitt 3.1). Anschließend werden die Benutzeroberflächen prototypisch geplant (Abschnitt 3.2). Im letzten Schritt werden die benötigten Endpunkte des Backends ermittelt (Abschnitt 3.3).

Die Entwurfsphase ist für die Entwicklung des Systems wichtig, damit grundlegende Entscheidungen möglichst früh getroffen werden. Wenn die Entwicklung des Systems bereits begonnen hat, benötigen nachträgliche Änderungen oft einen höheren Aufwand. Durch einen guten Entwurf soll die Implementierung beschleunigt werden, da in der Implementierungsphase dann weniger Entscheidungen getroffen werden müssen. [11]

## 3.1. Aufbau der Datenbank

Die Erstellung eines Datenbankschemas ist ein wichtiger Schritt. Mit einem vollständigen Datenbankschema wird sichergestellt, dass alle benötigten Daten in der Datenbank gespeichert werden können, dass auf die Daten effizient zugegriffen werden kann und dass die Datenkonsistenz gewährleistet ist. [18, Kapitel 3]

Die Datenbankschemas werden mithilfe der Krähenfußnotation [18, Seite 26] erstellt. Die Bedeutung der Kardinalitäten ist in Tabelle 7 erklärt. [19]

Zeichen	Bedeutung
o	0 oder eins
	genau eins
o{	0 oder mehrere
{	eins oder mehrere

Tabelle 7: Notation der Kardinalitäten

### 3.1.1. Benötigte Tabellen

Für die Kernfunktionalität werden zunächst Tabellen für Module und Teilmodule benötigt. Um die Organisationsstruktur der Hochschule abbilden zu können, werden weiterhin Tabellen für Fakultäten, Abteilungen und Studiengänge benötigt. Damit ein Login möglich ist, kann die Tabelle User benutzt werden. Für die Anforderung F22 wird eine Tabelle

Changelog und eine Tabelle ChangelogItem benötigt. In der Tabelle Changelog wird für jede Änderung ein Eintrag eingetragen, welcher den ausführenden User und eine kurze Zusammenfassung beinhaltet. In den ChangelogItems stehen dann die konkreten geänderten Felder. Damit ein Modul einer Gruppe zugewiesen werden kann, wird eine Tabelle ModuleGroup benötigt, in der alle verfügbaren Gruppen aufgelistet sind. Des Weiteren werden noch die Tabelle „Job“ und verschiedene „PDFStructure“-Tabellen für die Generierung der PDF-Dateien benötigt (dazu später mehr in Unterabschnitt 4.1.3).

Jede Tabelle erhält eine Spalte „Id“ als Primärschlüssel. [18, Kapitel 3] Über diese Id werden die Relationen zu anderen Tabellen ermöglicht. Zusätzlich wird für jede Eigenschaft aus Abschnitt 2.7 eine dazugehörige Spalte benötigt.

### 3.1.2. Überlegungen zur Übersetzbarkeit

Um die Anforderung N24 umsetzen zu können, wird eine Datenstruktur benötigt, welche die verschiedenen Texte in verschiedenen Sprachen aufbewahren kann. Hierzu wurden zunächst verschiedene Möglichkeiten evaluiert, um das beste Vorgehen zu ermitteln.

#### Idee 1

Für die spätere Entwicklung des Frontends könnte es praktisch sein, alle Texte einer Art in einer eigenen Tabelle aufzubewahren (Diagramm 1). Die Eingabe eines Textes könnte dann mit allen bisherigen Texten in der Tabelle abgeglichen werden, um Fehler zu finden. Für die erste Idee würde also pro Textfeld eine eigene Tabelle angelegt werden. Das Modul hätte beispielsweise für den Titel dann nur eine Id, die auf einen Eintrag in der Tabelle ModuleTitle verweist. ModuleTitle hätte dann einen Verweis auf TranslatedText. In TranslatedText könnten dann die konkreten Texte stehen.

Diese Methodik hat den Nachteil, dass pro Textfeld eine neue Tabelle benötigt wird. Dadurch kann die Datenstruktur schnell unübersichtlich werden. Außerdem könnte der Zugriff kompliziert sein. Für die Entwicklung dieses Systems sollte eine einfachere Lösung gesucht werden.

#### Idee 2

Eine etwas einfachere Methode wäre es, die übersetzten Texte in einer zentralen Tabelle „Translations“ abzulegen (Diagramm 2). Die Tabelle Modul hätte dann z. B. die Spalten TitleTranslationId, welche ein Foreign-Key auf die Tabelle Translation wäre. In Translation würde es dann die Spalten „Id“, „English“ und „German“ geben, um die Texte dort abzuspeichern.

Für diese Lösung muss nur eine einzelne zusätzliche Tabelle erstellt werden, was den initialen Aufwand minimiert. Auch hat die zentrale Speicherung von Texten den Vorteil, dass diese sehr einfach verwaltet werden können. Bei der Einführung einer neuen Sprache müsste nur eine neue Spalte zur Tabelle hinzugefügt werden.

Die Lösung hat allerdings einen entscheidenden Nachteil. Ein Teilmodul hat 12 zu übersetzende Felder. Wenn dieses nun auf der Oberfläche angezeigt werden soll, muss für jedes der Felder ein Datenbank-Join oder eine Unterabfrage gemacht werden. In der Tabelle Modul wird für jedes Textfeld nur eine Id abgelegt, die dann aus der Übersetzungstabelle abgerufen werden muss. Dies hätte einen hohen Aufwand im zukünftigen Quellcode der Anwendungen zur Folge.

#### **Idee 3**

Um Zugriffe auf die Texte einfacher zu gestalten, wurde eine weitere Möglichkeit entwickelt (Diagramm 3). Für jede Entität (z. B. Modul, Teilmodul...) wird neben der normalen Tabelle eine weitere Tabelle mit dem Suffix „\_Translations“ angelegt. Diese Zusatztable enthält die übersetzten Textfelder. Für jede Sprache gibt es eine Zeile in der neuen Tabelle. Wenn es also zwei Sprachen gibt (Englisch & Deutsch), gibt es für jedes Modul zwei Einträge in der dazugehörigen Übersetzungstabelle.

Um nun ein Teilmodul mit 12 Feldern aus der Datenbank zu erhalten, wird mit dieser Lösung nur noch ein Join benötigt. Der Quellcode der zukünftigen Anwendung sollte dadurch deutlich besser wartbar sein.

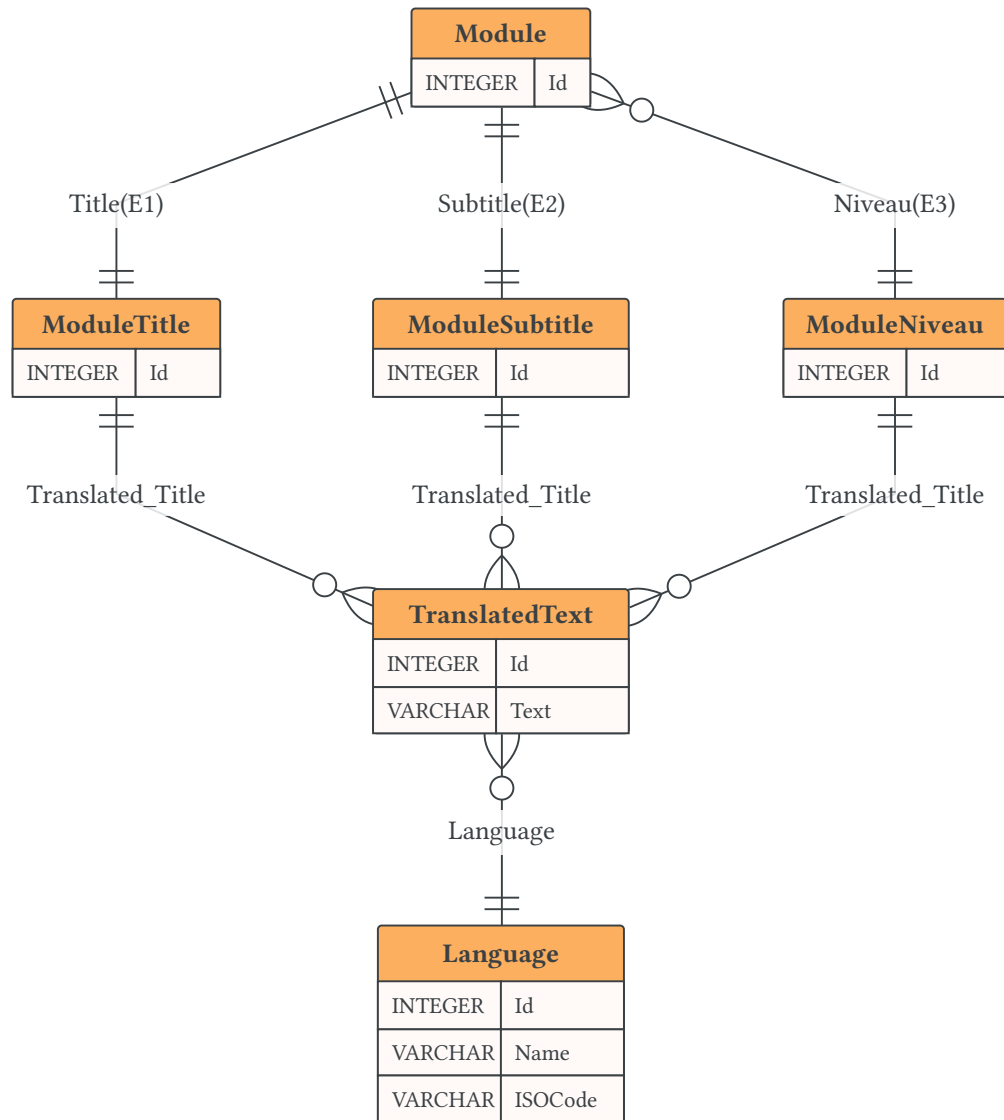


Diagramm 1: ER-Diagramm - Idee 1

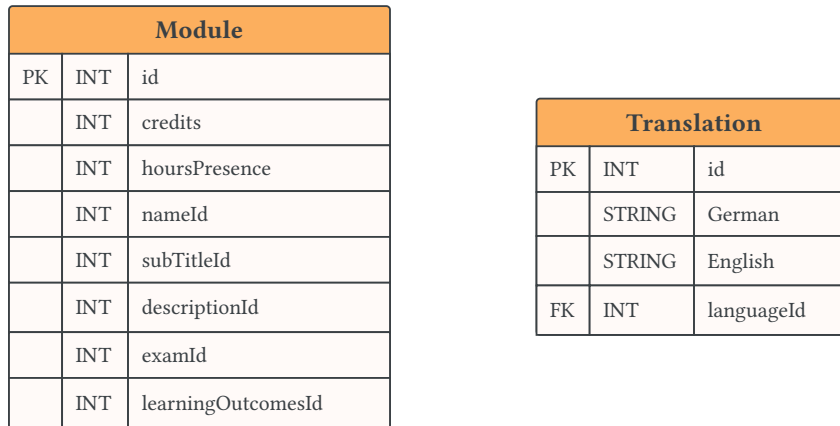


Diagramm 2: ER-Diagramm - Idee 2

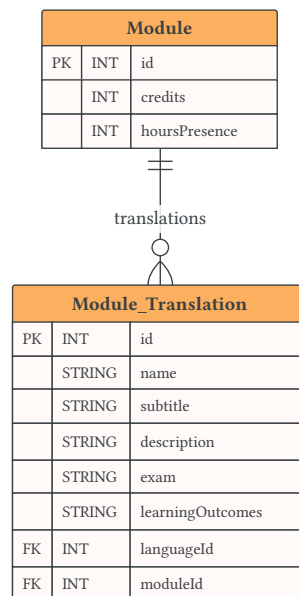


Diagramm 3: ER-Diagramm - Idee 3

### 3.1.3. Resultierendes Schema

In Diagramm 4 ist ein kleiner Teil des entstandenen ER-Diagramms zu sehen. Die vollständige Version des Diagramms ist sehr groß und findet daher hier keinen Platz. Die vollständige Abbildung ist in der Dokumentation des Systems zu finden (Pfad: /docs/backend/Architecture/Database).

In dem Diagramm ist beispielsweise auf der linken Seite zu sehen, dass ein Modul immer einem Studiengang zugewiesen sein muss. Andersherum kann ein Studiengang 0 bis n verschiedene Module anbieten. Das Schema soll Entwickelnde dabei unterstützen, die Datenstruktur des Systems zu verstehen. Anhand des Schemas kann im folgenden Kapitel die Datenbank des Backends erstellt werden, sodass diese dann alle benötigten Daten abspeichern kann.

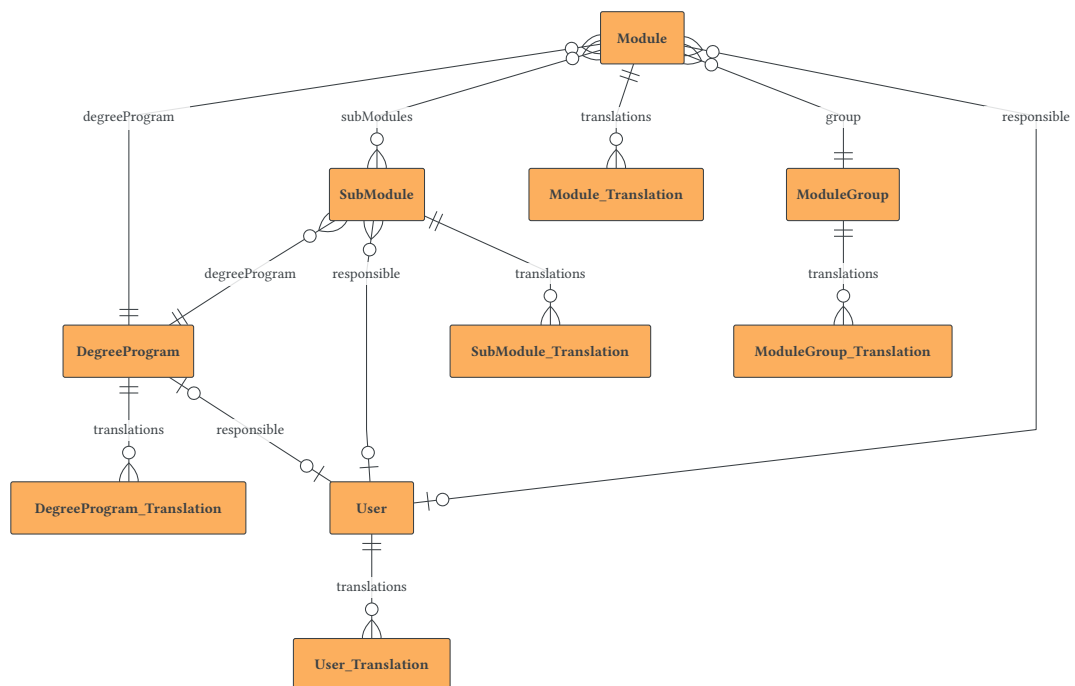


Diagramm 4: ER-Diagramm - Gesamtbild

## 3.2. Benutzeroberflächen

Im Folgenden wird mithilfe von Mockups entworfen, wie die Benutzeroberflächen der neuen Anwendung aussehen sollen. Da an Mockups schnell Änderungen vorgenommen werden können, soll dieser Prozess dabei helfen, zeiteffizient gute Lösungen zu finden. Es werden zunächst in Unterabschnitt 3.2.1 die Elemente zur Navigation durch die Anwendung vorgestellt. Anschließend werden in Unterabschnitt 3.2.2 die verschiedenen Komponenten der Anwendung sowie die daraus zusammengesetzten Ansichten skizziert.

In den Mockups wird darauf geachtet, die von J. Tidwell [20] beschriebenen bekannten UI-Patterns anzuwenden. Dadurch sollen die Ansichten der neuen Anwendung selbsterklärend sein, da sie anderen modernen Websites ähneln. Weiterhin soll jede Ansicht, wenn möglich, genau einen Zweck verfolgen. Entweder soll eine Übersicht mehrere Elemente zeigen, oder ein einzelnes Element soll im Fokus stehen und detailliert gezeigt werden. Alternativ kann ein neues Element erstellt werden, oder eine Aufgabe soll erledigt werden. Das Mischen dieser Zuständigkeiten kann zu einer überladenen Website führen und wird deshalb, wenn möglich, vermieden. [20, Seite 35 ff.]

### 3.2.1. Grundgerüst

Die Oberfläche der Anwendung besteht aus einer oberen Leiste (Toolbar) und einer ausklappbaren Seitenleiste (Drawer). Unter der Toolbar ist die eigentliche Anwendung zu sehen, die aus verschiedenen Ansichten besteht. Die Toolbar ist in jeder Ansicht zu sehen. Diese Art der Navigation ist mittlerweile Standard und sollte für den Großteil der User selbsterklärend sein. [20, Seite 131]

#### **Toolbar**

Um mit möglichst wenig Aufwand (N22) jederzeit die Suchfunktion (F5) nutzen zu können, wird diese in der Toolbar platziert (siehe Abbildung 4). Neben der Suche ist ein Dropdown, mit dem die angezeigte Sprache umgestellt werden kann (N24). Damit jederzeit erkenntlich ist, in welcher Ansicht sich der User befindet (N10), wird diese Information als Breadcrumb auf der Toolbar platziert. [20, Seite 193 f.]

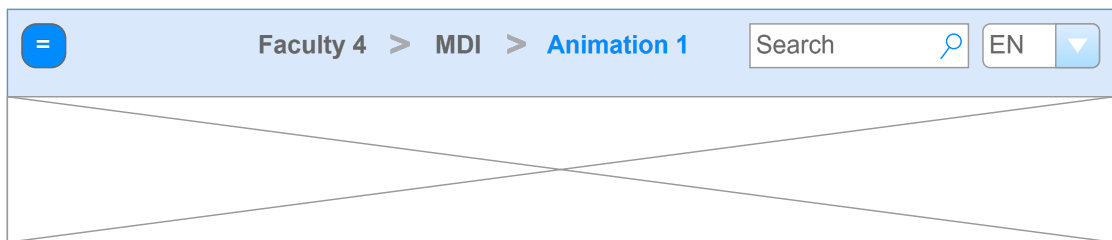


Abbildung 4: Toolbar

### Drawer

Funktionen, die nicht oft benötigt werden, werden in einer ausklappbaren Seitenleiste (Drawer) platziert. Der Drawer kann mithilfe eines Knopfes ausgeklappt werden, welcher sich auf der Toolbar befindet. Somit können auch diese Funktionen mit wenig Aufwand (N22) von jeder Ansicht aus erreicht werden. Im Drawer sind die Masken zur Verwaltung der Benutzer (F19), zur Anzeige gelöschter Module (N12) und zur Ansicht alter Prüfungsordnungen (siehe Abbildung 5). Außerdem wird hier die Versionsnummer angezeigt, damit jederzeit überprüft werden kann, mit welcher Version des Systems gearbeitet wird.



Abbildung 5: Drawer

### 3.2.2. Komponenten und Ansichten

#### Login

Wenn ein User administrative Aufgaben übernehmen möchte, muss er sich zunächst über den Login-Button im Drawer (Abbildung 5) anmelden (F7). Hierzu wird ein Screen benötigt, auf dem der User seine E-Mail und sein Passwort eingeben kann. Da Accounts von der studiengangverantwortlichen Person erstellt werden (F19), wird auf dieser Seite keine Möglichkeit benötigt, sich selbst zu registrieren. Damit verständlich ist, wie ein Account erstellt werden kann, wird diese Information als Infotext unter dem Login-Button platziert (siehe Abbildung 6).

**StudyModules**

E-Mail

\*\*\*\*\*

**Login**

Passwort vergessen, oder noch keinen Account erstellt?  
Melden Sie sich bei der studiengangverantwortlichen Person Ihrer Abteilung.

Abbildung 6: Login

### Suchfunktion

Für die Suchfunktion wird eine Komponente benötigt, mit der ein bestimmtes Modul gefunden werden kann. In die Suchleiste soll der User den Namen des gesuchten Moduls eingeben können. Dabei muss der Modulname nicht vollständig eingegeben werden. Module, die zu dem eingegebenen Text passen, sollen vorgeschlagen werden. Durch die Vorschläge (siehe Abbildung 7) spart der User Zeit, da nicht der vollständige Modulname eingegeben werden muss und auch nicht erst zu einer Ergebnisseite weitergeleitet wird. [20, Seite 502 ff.] Der User kann einen Vorschlag anklicken, um sich dieses Modul anzusehen.

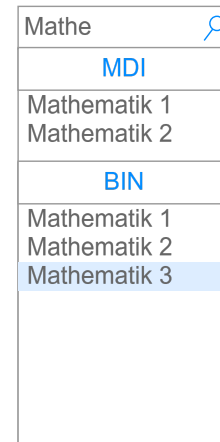


Abbildung 7: Suche mit Dropdown

### Navigation

Auf der Startseite der neuen Anwendung soll nicht direkt die Modulübersicht präsentiert werden, weil diese ohne gesetzte Filter überladen wirken könnte. Stattdessen wird der User zunächst auf eine Übersicht aller Fakultäten geleitet (Abbildung 8). Hier kann entweder eine Fakultät ausgewählt werden, oder es kann per Klick auf „Alle Module“ direkt zur Modulübersicht gewechselt werden. Die Farben der einzelnen Fakultäten sind dieselben wie auf der Website der Hochschule Hannover [21] und sollen den User dabei unterstützen, schnell die richtige Fakultät zu finden. Diese Ansicht wird übersprungen, falls der User einen direkten Link zu einem bestimmten Studiengang aufruft. Dieser Link könnte beispielsweise auf der Website der Hochschule platziert sein. Der User wird dann direkt zur Modulübersicht geleitet.

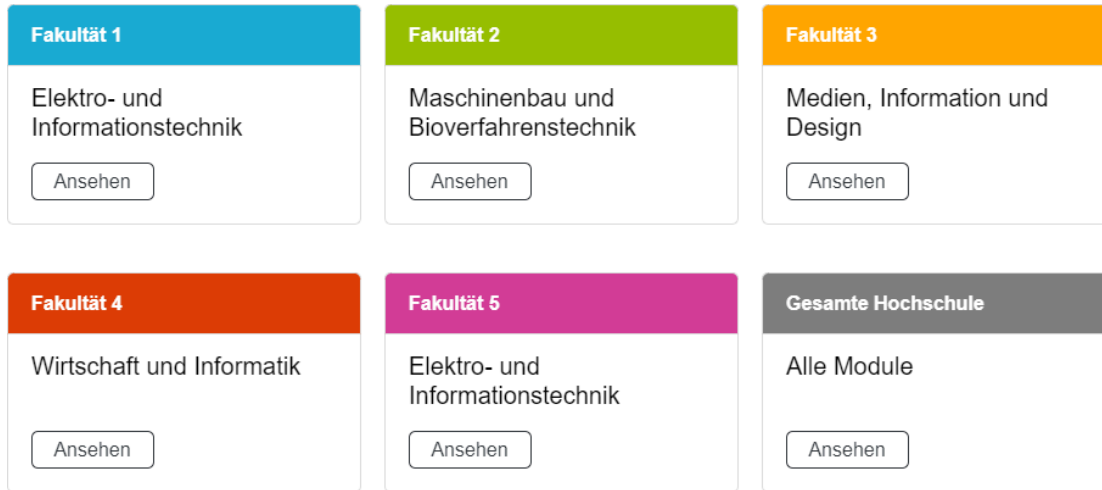


Abbildung 8: Startseite - Auswahl Fakultät

Wenn in der Fakultätsauswahl eine Fakultät ausgewählt wurde, geht es weiter auf die Detailansicht der jeweiligen Fakultät (Abbildung 9). Hier werden alle Studiengänge der Fakultät aufgelistet. Die Studiengänge sind gruppiert nach der Abteilung, zu der sie zugehörig sind. Zur besseren Übersicht sind die Bachelorstudiengänge über den Masterstudiengängen angeordnet und zusätzlich farblich markiert. Des Weiteren gibt es die Möglichkeit, in der oberen Leiste gezielt nach einem Studiengang zu suchen oder nach Bachelor/Master zu filtern. Sobald auf dieser Übersicht ein Studiengang angeklickt wird, öffnet sich die Modulübersicht. In der Modulübersicht sind dann die Filter automatisch an die zuvor ausgewählte Fakultät und den ausgewählten Studiengang angepasst.

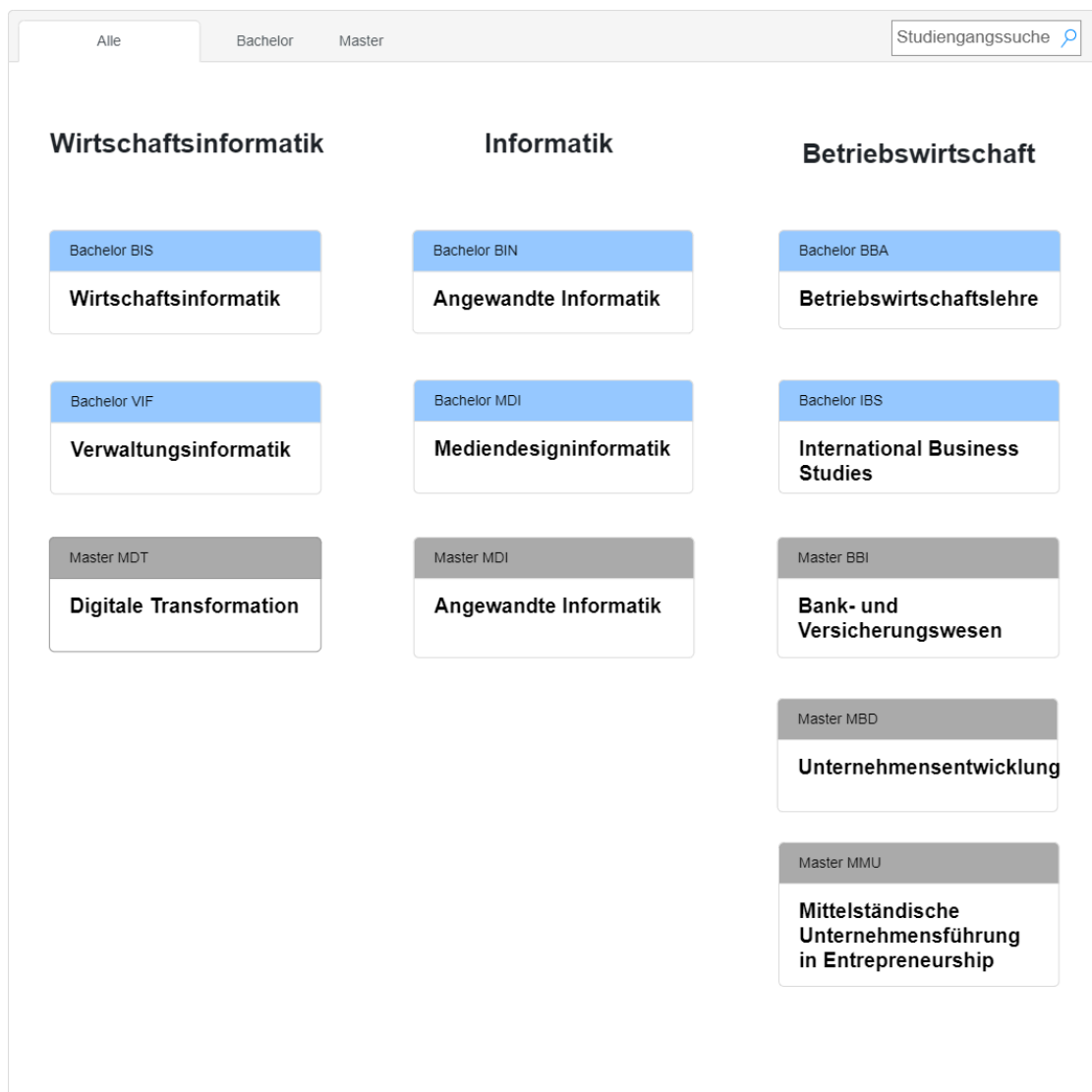
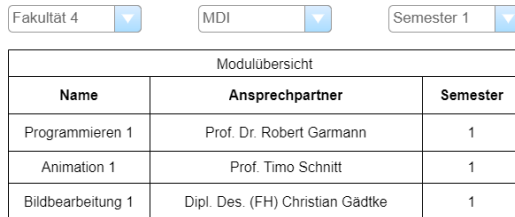


Abbildung 9: Startseite - Auswahl Studiengang

### Modulübersicht

In der Modulübersicht (Abbildung 10) sollen alle Module in einer tabellarischen Übersicht angezeigt werden. In der Tabelle sollen nur die für Use Case 2 benötigten Daten gezeigt werden. Alle weiteren Informationen sind dann in der detaillierten Ansicht zu finden. Die detaillierte Ansicht soll sich durch Anklicken eines Eintrags öffnen. Über der Tabelle gibt es mehrere Filtermöglichkeiten, um die gesuchten Module schneller finden zu können.



The image shows a user interface for a module overview. At the top, there are three filter dropdown menus: 'Fakultät 4', 'MDI', and 'Semester 1'. Below the filters is a table titled 'Modulübersicht' with three columns: 'Name', 'Ansprechpartner', and 'Semester'. The table contains three rows of data.

Modulübersicht		
Name	Ansprechpartner	Semester
Programmieren 1	Prof. Dr. Robert Garmann	1
Animation 1	Prof. Timo Schnitt	1
Bildbearbeitung 1	Dipl. Des. (FH) Christian Gädtke	1

Abbildung 10: Modulübersicht und Filter

### Detailansicht eines Modules

Sobald ein Modul ausgewählt wurde, wird die Detailansicht eines Modules (Abbildung 11) präsentiert. Hier sollen die Details des Modules übersichtlich dargestellt werden. Einzelne Informationen können beispielsweise mithilfe eines Graphen oder anhand von Icons visualisiert werden. Hierdurch können Studierende oder studieninteressierte Personen innerhalb kurzer Zeit einen Eindruck erhalten, was die Rahmenbedingungen des Modules sind.

=
Suchen

## Mathematik 1

**FS** Frauke Sprengel  
Verantwortlich


**6** Credits

**Deutsch**  
Sprache

**1** Semester

**100** Gruppengröße

### Zeitaufwand



Insgesamt: 180 h  
Selbststudium: 112 h  
Präsenz: 68 h  
Präsenz pro Woche: 4 h

● Präsenz ● Selbststudium

### Empfohlene Voraussetzungen

Bestandene Vorprüfung, mind. 134 CP erworben aus anderen Modulen des BIN-Programms

### Veranstaltung

- VÜ** Vorlesung mit Übung
- PE** Prüfung (Klausur oder mündliche Prüfung) und experimentelle Arbeit

### Pflicht-Voraussetzungen

Abgeschlossenes Bachelor-Studium mit Informatikanteil, der mindestens den Programmieren-, Datenbanksysteme- und Betriebssysteme/Netz-Veranstaltungen vergleichbar ist.

### Angestrebte Lernergebnisse

Formale Kompetenz: Kenntnisse der Logik und Vertrautheit mit mathematischen Formalismen zur Beschreibung von Sachverhalten  
Algorithmische und mathematische Kompetenz: Kennenlernen mathematischer Algorithmen, geeignete Lösungsverfahren für elementare Probleme der Mathematik und Informatik auswählen und durchführen  
Übergreifend: kommunikative Kompetenz (Präsentation und Diskussion von Lösungsvorschlägen)

### Literatur

Skript zur Vorlesung Teschl, G., Teschl, S.:  
Mathematik für Informatiker, Springer - Verlag  
Hartmann, P.: Mathematik für Informatiker, Vieweg - Verlag

### Inhalt

Erarbeitung einer Fragestellung und ihrer Lösung in einem vorgegebenen Themengebiet wie Spieleprogrammierung, Robotik, Algorithmen.  
Erprobung von Methoden, die im Projektkontext in teamindividuellen Beratungsgesprächen vermittelt werden.

### Anforderungen

<p><u>Präsenzzeit:</u></p> <p>Aktive Mitarbeit, Selbständiges Bearbeiten von Übungsaufgaben, Diskussion</p>	<p><u>Selbststudium:</u></p> <p>Vor- und Nachbereitung, Literaturarbeit, Selbständiges Bearbeiten von Übungsaufgaben, Diskussion</p>
---	--

Abbildung 11: Detailansicht Modul

### Modul anlegen / bearbeiten

Angemeldete User sehen auf verschiedenen Seiten Buttons, mit denen sie Module anlegen und bearbeiten können. Damit die Dateneingabe für den User möglichst intuitiv ist, orientiert sich die Sortierung der Eingabefelder an der Sortierung der Felder im resultierenden PDF. Um eine einheitliche Optik zu erreichen, wurden anschließend die Felder geringfügig umsortiert, sodass gleiche Datentypen oder Felder, die thematisch zueinander passen, nah beieinander sind. Das resultierende PDF wird in der Vorschau imitiert. Hier werden die Eingaben des Users in Echtzeit angezeigt, sodass der User jederzeit sehen kann, wie die Modulbeschreibung aussehen wird.

Datenerfassung

Studiengang  
MDI

Nummer: 100      Name: Mathe 1

Studiensemester: 1      Moduldauer: 1

Credits: 6      Präsenzstunden: 68      Selbststudium: 112

Untertitel: Mathe 1       Pflichtmodul

Niveau: Grundlage      Verantwortliche Person: Sprengel

Voraussetzungen nach PO: Keine      Empfohlene Voraussetzungen: Keine

Studien-/Prüfungsleistungen: Prüfung      Angestrebte Lernergebnisse: Mathe 1

Teilmodule

Mathe 1

Teilmodulsuche

Vorschau

Modul BIN-100 Mathematik 1

Untertitel	Mathematische Grundlagen der Informatik (BIN-MAT 1)
Modulniveau	Grundlagenmodul
Pflicht / Wahlpflicht	Pflichtmodul
Teilmodule	BIN-100-01 Mathematik 1, Pflicht
Verantwortliche(r)	Sprengel, Frauke, Prof. Dr.
Credits	6
Präsenzstunden / Selbststudium	68 h / 112 h
Studiensemester	1
Moduldauer	1 Semester
Voraussetzungen nach Prüfungsordnung	keine
Empfohlene Voraussetzungen	keine
Studien-/ Prüfungsleistungen	Prüfung (Klausur oder mündliche Prüfung) und experimentelle Arbeit
<b>Angestrebte Lernergebnisse</b>	
	Formale Kompetenz: Kenntnisse der Logik und Vertrautheit mit mathematischen Formalismen zur Beschreibung von Sachverhalten Algorithmische und mathematische Kompetenz: Kennenlernen mathematischer Algorithmen, geeignete Lösungsverfahren für elementare Probleme der Mathematik und Informatik auswählen und durchführen Übergreifend: kommunikative Kompetenz (Präsentation und Diskussion von Lösungsvorschlägen)

Abbildung 12: Modul hinzufügen

Damit in der Auflistung der Änderungen eine hilfreiche Nachricht steht, sollen die vorgenommenen Änderungen beim Speichern eines Moduls zusammengefasst werden. Hierzu fragt ein Pop-up nach der Zusammenfassung und erklärt dem User, wo dieser Text zu sehen sein wird (Abbildung 13).

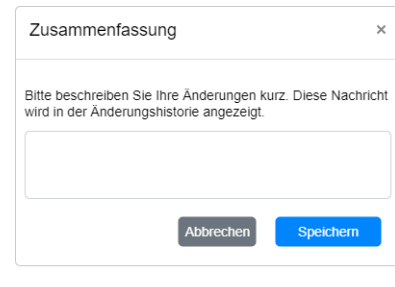


Abbildung 13: Text für Änderungshistorie

Für die Erstellung oder Bearbeitung eines Moduls kann entweder eine vorhandene Vorlage aus einem Dropdown ausgewählt werden (N25, Abbildung 14), oder ein neuer Text durch Klicken auf den „Neu“-Button angelegt werden. Dies ist besonders praktisch, da sich bestimmte Texte oft wiederholen.

Wenn ein neuer Text angelegt wird, muss der User einen Kurztext angeben, der im Dropdown angezeigt wird, sowie die tatsächlichen Texte, die später im Modulhandbuch abgebildet werden (Abbildung 15).



Abbildung 14: Dropdown zur Textauswahl

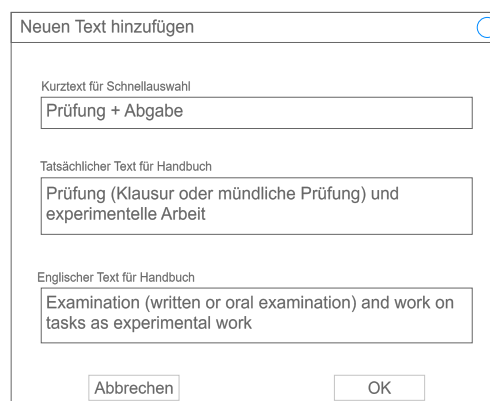


Abbildung 15: Neuen Text hinzufügen

### Userverwaltung

Das Hinzufügen eines neuen Users benötigt ebenfalls eine Eingabemaske (Abbildung 16). Der Inhalt des Feldes „E-Mail-Adresse“ wird automatisch anhand des eingegebenen Vor- und Nachnamens ausgefüllt, kann aber anschließend auch manuell bearbeitet werden, falls die E-Mail-Adresse vom bekannten Schema

The form consists of several input fields: a dropdown menu for a title (currently showing 'Prof. Dr.'), two text boxes for 'Vorname' and 'Nachname', a text box for 'E-Mail Adresse', and a password field represented by a series of asterisks.

Abbildung 16: Neuen User hinzufügen

(vorname.nachname@hs-hannover.de) einer Adresse der Hochschule Hannover abweicht. Dies ist beispielsweise der Fall, wenn mehrere Personen denselben Vor- und Nachnamen haben.

### Änderungshistorie

Alle Änderungen sollen nachverfolgbar sein (F22). Hierzu wird eine Übersicht über alle Änderungen benötigt (Abbildung 17). Wenn eine Reihe der Tabelle, also eine einzelne Änderung, angeklickt wird, öffnet sich eine Ansicht, in der die vorherige Version und die bearbeitete Version nebeneinander dargestellt sind. Mit einem Knopfdruck können die vorgenommenen Änderungen rückgängig gemacht werden. Auch hier wird nach einem Kommentar gefragt, um die Änderungen zusammenzufassen (Abbildung 13)

The interface shows three filter dropdowns: 'Fakultät 4', 'MDI', and 'Modulfilter'. Below them is a table titled 'Änderungshistorie' with the following data:

Modul	Geändert von	Geändert am	Kommentar
Programmieren 1	Prof. Dr. Robert Garmann	14.04.2024	Prüfungsart geändert
Animation 1	Prof. Timo Schnitt	10.04.2024	Tippfehler korrigiert
Bildbearbeitung 1	Dipl. Des. (FH) Christian Gädtke	14.03.2024	Texte angepasst

Abbildung 17: Änderungshistorie

## 3.3. Benötigte Endpunkte im Backend

Damit das zukünftige Frontend mit dem Backend kommunizieren kann, muss das Backend Endpunkte (auch genannt Ressourcen, Endpoints, Routen...) bereitstellen, die das Frontend nutzen kann. Ein Endpunkt ist beispielsweise die Auflistung aller Module und ist mithilfe einer URI aufrufbar (hier z. B. /modules). Ein einzelnes Modul könnte über den Endpunkt /modules/{id} aufgerufen werden. [22, Abschnitt 3.2]

In der vorliegenden Version des „StudyBase-“Backends gibt es bereits mehrere Endpunkte. Im Folgenden soll ermittelt werden, welche Endpunkte für das neue System benötigt werden. Nach dem YAGNI-Prinzip sollen dann in Kapitel 4 nur die Endpunkte ausgearbeitet werden, die für das neue System benötigt werden. [22, Unterabschnitt 4.2.8] Durch die Planung, welche Endpunkte benötigt werden und wie diese aussehen sollen, wird die spätere Implementierung in Unterabschnitt 4.1.2 beschleunigt, da dann dort weniger Entscheidungen getroffen werden müssen.

#### **Endpunkt /modules**

Im zukünftigen Frontend sollen an verschiedenen Stellen die Module aufgelistet werden. Die Suchfunktion soll Vorschläge anhand der Modulliste machen, die Studiengänge sollen ihre Module anzeigen und die studiengangverantwortliche Person soll alle Module verwalten können. Es wird also eine Ressource benötigt, die alle Module auflistet. Für die Suchfunktion muss die Ressource allerdings weniger Informationen anzeigen als für die Detailansicht eines Moduls. Beispielsweise ist für die Suche zunächst uninteressant, was die angestrebten Lernergebnisse eines Moduls sind. Durch die Begrenzung der übersendeten Informationen müssen weniger Daten vom Backend ans Frontend gesendet werden. Dies trägt zu einer besseren Performance bei, weil weniger Daten über das Internet übertragen werden.

Die Suchfunktion benötigt:

1. den Namen des Moduls, um einen Vorschlag anzuzeigen (Abbildung 7)
2. die Id des Moduls, um die Detailansicht zu dem Modul öffnen zu können
3. den Studiengang des Moduls, um die Ergebnisse gruppieren zu können (Abbildung 7)

Die Tabellen im restlichen System profitieren dahingegen von zusätzlichen Informationen. Diese können dabei helfen, die Module zu filtern (F4) oder schnell Informationen zu einem Modul herauszufinden, ohne dieses aufrufen zu müssen.

Es werden also die folgenden Informationen zusätzlich benötigt:

4. Semester, in dem das Modul vorgeschlagen ist (E9), um beispielsweise nur die Module des aktuellen Semesters zu filtern
5. Credits, um nach Modulen filtern zu können, die einen bestimmten Aufwand haben
6. Gruppe, um beispielsweise ein interessantes Wahlpflichtfach zu finden
7. Ansprechpartner (E6), um schnell eine Kontaktinformation zu erhalten

Der Endpunkt /modules soll zusätzlich POST-Anfragen entgegennehmen können, um neue Module anzulegen.

#### **Endpunkt /modules/{id}**

Hier sollten alle Informationen zur Verfügung gestellt werden, die für die Detailansichten eines einzelnen Moduls benötigt werden. Es müssen also alle Informationen aus Unterabschnitt 2.7.1 enthalten sein.

Der Endpunkt `/modules/{id}` soll zusätzlich PUT-Anfragen entgegennehmen können, um bestehende Module bearbeiten zu können.

#### **Endpunkt `/modules/{id}/changes`**

Hier sollten alle Informationen zur Verfügung gestellt werden, die für die Änderungshistorie eines Moduls benötigt werden (Abbildung 17). Dazu gehört die Auflistung aller vorgenommenen Änderungen, der Autor der Änderung sowie der erklärende Text, der bei einer Änderung angegeben werden muss (Abbildung 13).

#### **Endpunkt `/submodules`**

Ähnlich wie schon bei `/modules` wird eine Auflistung aller Teilmodule benötigt. Damit Teilmodule auffindbar sind, sollte die Auflistung den Namen des Teilmodules enthalten. Außerdem könnte die verantwortliche Person enthalten sein, damit modulverantwortliche Personen nach den Teilmodulen filtern können, für die sie verantwortlich sind. Der Endpunkt `/submodules` soll zusätzlich POST-Anfragen entgegennehmen können, um neue Teilmodule anzulegen.

#### **Endpunkt `/submodules/{id}`**

Hier sollten alle Informationen zur Verfügung gestellt werden, die für die Detailansichten eines einzelnen Teilmoduls benötigt werden. Es müssen also alle Informationen aus Unterabschnitt 2.7.2 enthalten sein.

Der Endpunkt `/submodules/{id}` soll zusätzlich PUT-Anfragen entgegennehmen können, um bestehende Teilmodule bearbeiten zu können.

#### **Endpunkt `/group`**

Für das Dropdown, in dem die Gruppe eines Modules (E15) angegeben wird, wird ein eigener Endpunkt benötigt. Der Endpunkt `/group` soll zusätzlich POST-Anfragen entgegennehmen können, um neue Gruppen anzulegen.

#### **Endpunkt `/language`**

Die Anwendung soll mehrere Sprachen unterstützen (N24). Damit ein Wechsel der Sprache möglich ist, liefert dieser Endpunkt eine Liste der unterstützten Sprachen.

#### **Endpunkt `/autocomplete/{languageId}?type={type}`**

Dieser Endpunkt soll alle verfügbaren Übersetzungen des angegebenen Typs zurückgeben. Ein Typ ist hier beispielsweise „EXAM\_TYPE“, für das Dropdown, in dem die Prüfungsart angegeben wird (E13). Mithilfe der Information sollen die verschiedenen Dropdowns in der Modulbearbeitung (Abbildung 12) oder der Usererstellung (Abbildung 16) befüllt werden.

#### **Endpunkt `/faculties`**

Dieser Endpunkt ist für die Übersicht aller Fakultäten (Abbildung 8) zuständig. Hier ist es ausreichend, die Id der Fakultät sowie deren Namen bereitzustellen.

#### **Endpunkt /faculty/{id}**

Für die Auswahl des Studiengangs (Abbildung 9) müssen alle Abteilungen der gewählten Fakultät geladen werden. Außerdem muss für jede Abteilung übermittelt werden, welche Studiengänge angeboten werden. Die Studiengänge müssen ein Feld enthalten, welches zwischen Bachelor- und Masterstudiengang unterscheidet, damit nach dieser Information gefiltert werden kann.

#### **Endpunkt /department**

Dieser Endpunkt bietet eine Auflistung aller Abteilungen an. Zusätzlich sollten POST-Requests angenommen werden, um neue Abteilungen hinzufügen zu können.

#### **Endpunkt /degrees**

Dieser Endpunkt bietet eine Auflistung aller Studiengänge an. Zusätzlich sollten POST-Requests angenommen werden, um neue Studiengänge hinzufügen zu können.

#### **Endpunkt /auth/login**

Für den Login (Abbildung 6) wird ein Endpunkt benötigt, der E-Mail und Passwort entgegennimmt, validiert und einen gültigen Token zurückgibt. Mithilfe des Tokens können die administrativen Funktionen genutzt werden, ohne sich jedes Mal mit E-Mail und Passwort autorisieren zu müssen. Dieser Endpunkt bestand bereits und wurde im Rahmen dieser Arbeit nicht verändert.

#### **Endpunkt /job**

Für die PDF-Generierung wird ein Endpunkt benötigt, der eine Auflistung der Kompilierungsaufträge anbietet. Dieser Endpunkt muss POST-Anfragen annehmen, damit neue Aufträge angelegt werden können. Außerdem müssen PATCH-Anfragen angenommen werden, um den Status des Jobs zu verändern.

### **3.4. Zwischenfazit**

Im vergangenen Kapitel wurde das gesamte System geplant. Für das Backend gibt es ein Datenbankschema sowie einen Plan, wie die Endpunkte der API gestaltet werden sollen. Für das Frontend gibt es Mockups für das Grundgerüst der Anwendung sowie Mockups für einige Komponenten und Ansichten. Dank der detaillierten Planung sind im folgenden Kapitel 4 weniger Entscheidungen zu treffen, sodass die Implementierung risikofreier ist.

# 4. Implementierung

Nachdem in Kapitel 3 das System geplant wurde, soll nun in diesem Kapitel das System erstellt werden. Zunächst wird das Backend vorbereitet. Hierzu wird in Unterabschnitt 4.1.1 als Erstes die Datenbank an das neue Schema angepasst, um im Anschluss die geplanten Endpunkte implementieren zu können. Sobald das Backend vollständig ist, kann das Frontend die benötigten Daten abrufen. Deshalb wird erst im zweiten Schritt dann in Abschnitt 4.2 das Frontend erstellt. In Abschnitt 4.3 ist beschrieben, wie das System dokumentiert wurde. Abschließend wurde das System für ein Deployment mit Podman vorbereitet (Abschnitt 4.4).

## 4.1. Backend

Das Backend wird im Folgenden auch als API bezeichnet und greift auf verschiedene andere Komponenten zu. Es gibt eine Datenbank, in der die Informationen zu den Modulen, Usern und die Änderungshistorie gespeichert werden. Außerdem gibt es die API selbst, die Daten aus der Datenbank lädt und mithilfe von HTTP-Endpunkten an das Frontend weitergibt. Zuletzt gibt es ein Python-Skript und einen Docker-Container, welche für die einzelnen Studiengänge die Modulhandbücher als PDF generieren. In den folgenden Unterabschnitten ist die Implementierung der genannten Komponenten beschrieben.

### 4.1.1. Datenbank

Im ersten Schritt wurde das vorhandene Datenbankschema an das in Abschnitt 3.1 erstellte Schema angeglichen.

Hierzu musste die in Unterabschnitt 2.1.3 gezeigte `schema.prisma`-Datei bearbeitet werden. In der Prisma-Datei entspricht ein wie in Code 1 gezeigter Block einer Tabelle in der Datenbank. Einzelne Zeilen innerhalb des Blocks entsprechen Spalten in der Tabelle. Neben der `id`-Zeile in Code 1 ist zu sehen, wie eine selbstständig hochzählende Zahl möglich ist. Außerdem können in der Prisma-Datei Relationen zwischen Tabellen definiert werden, indem hinter eine entsprechende Zeile `@relation` geschrieben wird. [23] Hierbei ist zu beachten, dass immer in beiden Tabellen eine `@relation` definiert werden muss (siehe Code 2 Zeile 4). Mithilfe von Konsolenbefehlen kann dann die Prisma-Datei auf die Datenbank angewendet werden.[24]

```
1 model Module {
2   id Int @id @default(autoincrement())
3   name String?
4   degreeProgram DegreeProgram @relation(fields: [degreeProgramId], references: [id])
5   degreeProgramId Int
```

prisma

```
6 ...
7
8 model DegreeProgram {
9     id          Int          @id @default(autoincrement())
10    modules     Module[]
11 ...
```

Code 2: Beispiel für Relations-Felder

Bei dem Prozess, die vorhandene Datenstruktur zu ändern, gingen die Test-Daten verloren. Es wurde kein Aufwand investiert, um die Migration verlustfrei zu gestalten. Nach Fertigstellung der Datenstruktur sollen die vom Studiendekan ermittelten Daten in die Datenbank eingesetzt werden, da diese auf einem neueren Stand sind, bereits geprüft wurden und vollständig sein sollten.

Die Datenübernahme wurde mithilfe von Python umgesetzt. Bei der Erstellung des Skripts wurde darauf geachtet, dass es bei jeder Ausführung zunächst die neue Datenbank leert, um anschließend die Daten von der alten Datenbank in die neue Datenbank zu kopieren. Anschließend wurden die Daten der verschiedenen Tabellen eingelesen, auf das neue Format konvertiert und in die neue Datenbank eingesetzt. Die hierzu genutzten Insert-Statements mussten aufgrund der Abhängigkeiten zwischen den verschiedenen Tabellen in einer bestimmten Reihenfolge erfolgen. Weil beispielsweise jedes Modul einem Studiengang zugewiesen wird, müssen erst alle Studiengänge in die Datenbank eingesetzt werden, bevor deren Module eingesetzt werden können.

### 4.1.2. HTTP-Endpunkte

Nachdem die Datenbank vorbereitet war, konnten nun die benötigten Endpunkte im Backend angelegt werden.

Hierzu wurden zunächst Controller-Klassen für die verschiedenen Entitäten angelegt. Beispielsweise wird ein DegreeController benötigt (Code 3), um die Endpunkte für Studiengänge bereitzustellen. Innerhalb der Controller wurde anschließend für jeden benötigten Endpunkt eine Methode erstellt und mithilfe von Dekoratoren näher beschrieben. [25] Der @Controller-Dekorator (Code 3, Zeile 2) sorgt dafür, dass NestJS die Klasse als Controller identifiziert und entsprechend die enthaltenen Endpunkte bereitstellt. Außerdem ist als Parameter angegeben, unter welchem Pfad die Endpunkte erreichbar sind (hier: /degrees). Der @ApiTags-Dekorator (Code 3, Zeile 1) sorgt dafür, dass die Endpunkte in der automatisch generierten Dokumentation der API gruppiert in der Gruppe „Degrees“ dargestellt werden. Weiterhin werden die Methoden für die Endpunkte mit einem Dekorator versehen, der die HTTP-Methode angibt, die genutzt werden muss, um den Endpunkt aufzurufen (Code 3, Zeile 8). [14] Wenn der Endpunkt /degrees per GET-Request aufgerufen wird, wird das Ergebnis der findAll-Methode

zurückgegeben. Wenn der Endpunkt `/degrees/5` aufgerufen wird, wird das Ergebnis der `findOne`-Methode zurückgegeben. Die `findOne`-Methode erhält außerdem weitere Parameter, die auch in vielen anderen Methoden im Backend ähnlich genutzt werden. Der erste Parameter enthält die Id aus dem Pfad der Anfrage (im Beispiel `/degrees/5`, wäre das die 5). Der zweite Parameter enthält ein Request-Objekt. Dieses wird genutzt, um die in dem Request übergebenen Header auszulesen. Im aktuellen Beispiel wird die angefragte Sprache ausgelesen, um vom `DegreeService` Informationen in der Sprache des Benutzers zu erhalten.

Über den Constructor (Code 3, Zeile 4) werden per Dependency Injection [26] die benötigten Services übergeben. Das Framework `nest.js` sorgt dafür, dass die Services einmalig instanziiert werden. Das Instanzieren muss also nicht selbst unternommen werden und durch die einmalige Instanzierung kann die Performance der Anwendung verbessert werden. [27]

```
1  @ApiTags('Degrees')
2  @Controller('degrees')
3  export class DegreeController {
4      constructor(
5          private degreeService: DegreeService
6      ) {}
7
8      @Get('')
9      findAll() {
10         return this.degreeService.findAll();
11     }
12
13     @Get('/:id')
14     findOne(@Param('id') id: string, @Req() request: Request):
    Promise<any> {
15         const language = (request.headers as any)['language'];
16         return this.degreeService.findById(+id, language);
17     }
18 }
```

Code 3: `degree.controller.ts` (Auszug)

### Datenverändernde Endpunkte

Die Endpunkte, die Daten erstellen oder bearbeiten, sind verglichen mit den bisher vorgestellten Endpunkten weitaus komplexer, sodass es mehrere Design-Entscheidungen zu treffen gibt.

Das aufrufende System (hier das Frontend) sendet Daten im JSON-Format an das Backend. Ein erster Ansatz zur Erstellung einer Methode für das Erstellen und Bearbeiten eines Moduls ist es, das erhaltene JSON direkt an den Prisma-Client zu übergeben und zu hoffen, dass der Prisma-Client das JSON korrekt verarbeitet. Allerdings gibt es bei diesem Ansatz gleich mehrere Hindernisse.

Wenn das JSON nur aus einfachen Datentypen wie Zeichenketten, Zahlen und Booleans bestehen würde, wäre der erste Ansatz ausreichend. Da das Modul-JSON jedoch auch Objekte und Arrays enthält, müssen diese zunächst herausgefiltert werden, weil Prisma sonst nicht weiß, wie mit den Objekten und Arrays umgegangen werden soll. Das Herausfiltern kann mithilfe von „destructuring assignment“ [28] erledigt werden (siehe Code 4, Zeile 2). Hierbei werden die Eigenschaften mit komplexeren Datentypen aus dem JSON (`moduleDto`) extrahiert und für die spätere Verwendung in eigene Variablen gespeichert. Die verbleibenden Eigenschaften befinden sich anschließend im Objekt `moduleData` (Code 4, Zeile 13). Dieses Objekt kann dann tatsächlich direkt an den Prisma-Client übergeben werden, um die darin enthaltenen Werte in die Datenbank zu schreiben (Code 4, Zeile 19). Die zuvor extrahierten Daten können nun separat behandelt werden.

In Code 4, Zeile 21 ist zu sehen, wie die Eigenschaft der verantwortlichen Person (`E6`) gesetzt wird. Da beim Erstellen eines Moduls eine verantwortliche Person aus einem Dropdown ausgewählt wird, muss im Backend kein neuer Eintrag für die Person angelegt werden. Stattdessen muss der Eintrag des Moduls nur eine Referenz auf die ausgewählte, bereits in der Datenbank bestehende Person erhalten. Dies ist wie im Beispiel gezeigt mit der Funktion `connect` [29] möglich. Auf die gleiche Art kann auch bei der Bearbeitung eines Moduls eine andere Person gesetzt werden. Die Funktion `connect` ändert im Hintergrund einfach die Eigenschaft `responsibleId` in der Modultabelle.

Auch für n-m-Beziehungen könnte die `connect`-Funktion von Prisma genutzt werden. Ein Modul hat Voraussetzungen (Requirements). Eine Voraussetzung könnte es sein, dass die Module BIN-100 und BIN-101 abgeschlossen sein müssen. Ein Modul hat also eine Voraussetzung und eine Voraussetzung hat 0 bis n viele Module. Übergibt man die gewünschten Modul-Ids mit der Funktion `connect`, erstellt Prisma die benötigten Einträge in einer Zwischentabelle, um die Beziehung abzubilden. Allerdings werden dabei die bereits vorhandenen Einträge nicht automatisch gelöscht. Das Löschen der bereits vorhandenen Einträge muss manuell angefordert werden. Dies ist beispielsweise möglich, indem der `set`-Methode ein leeres Array übergeben wird (siehe Code 4, Zeile 33). Alternativ kann auch direkt die `set`-Methode genutzt werden, um die neuen Module zu setzen (siehe Code 4,

Zeile 46). Das direkte Setzen der neuen Werte mithilfe der set-Methode ist die bevorzugte Variante, da hier mit weniger Anweisungen dasselbe Ergebnis erreicht wird. Der Code wird hierdurch lesbarer, deshalb sollte diese Variante verwendet werden.

Prisma bietet weiterhin eine Methode `upsert` an, die die Update-Methode und die Create-Methode kombiniert. Hierzu werden zunächst die Update- und Create-Objekte in eigene Variablen geschrieben. Anschließend können diese an die Upsert-Methode übergeben werden (siehe Code 4, Zeile 66). Außerdem wird eine Filter-Abfrage benötigt, mit der Prisma ermitteln kann, ob ein Update oder ein Create nötig ist (Code 4, Zeile 61).

```

1  async updateModule(moduleDto: ModuleDto) {
2      const {
3          id,
4          degreeProgramId,
5          groupId,
6          translations,
7          subModules,
8          responsibleId,
9          requirementsHardId,
10         requirementsSoftId,
11         requirementsSoft: requirementsSoftNew,
12         requirementsHard: requirementsHardNew,
13         ...moduleData
14     } = moduleDto;
15
16     const updateArgs = {
17         where: { id },
18         data: {
19             ...moduleData,
20
21             responsible: responsibleId ? {
22                 connect: { id: responsibleId }
23             } : undefined,
24
25             requirementsHard: {
26                 update: {
27                     requiredSemesters: requirementsHardNew.requiredSemesters,
28                     translations: {
29                         upsert: this.upsertTranslations(this.filterValidTranslations
30                     },
31                     modules: {
32                         set: [], // remove all modules, before adding the
new ones
33                         connect: requirementsHardNew.modules.map(module => ({
34                             id: module.id
35                         })))

```

ts

(1)

(2)

```
36     }
37   }
38 },
39
40   requirementsSoft: {
41     update: {
42       requiredSemesters: requirementsSoftNew.requiredSemesters,
43       translations: {
44         upsert: this.upsertTranslations(this.filterValidTranslations
45       },
46       modules: {
47         set: requirementsSoftNew.modules.map(module => ({
48           id: module.id
49         })))
50     }
51   }
52 },
53   ...
54 }
55
56 const createArgs = {
57   ...
58 };
59
60 const upsertArgs = {
61   where: { id },
62   update: updateArgs.data,
63   create: createArgs.data
64 }
65
66 const upsertedModule = await
67   this.prisma.module.upsert(upsertArgs);
```

Code 4: Backend: Erstellen und Bearbeiten von Modulen

Abschließend muss evaluiert werden, ob der entstandene Code verständlich und kompakt genug ist. Das vorgestellte Beispiel ist durch die vielen komplexen Datentypen stark gewachsen. Um den Code also wartbarer, lesbarer und verständlicher zu machen, wurde zuletzt noch die Codequalität optimiert. Hierzu wurden statt einer gemeinsamen Upsert-Methode zwei unterschiedliche Methoden (Create und Update) erstellt. Hierdurch wurde die Zuständigkeit klarer definiert. Außerdem wurden die verschiedenen Zuweisungen der komplexen Datentypen in jeweils eigene Methoden extrahiert. Dies hatte neben dem nun weitaus besser lesbaren Codes den zusätzlichen Vorteil, dass die Methoden an verschiedenen Stellen verwendet werden konnten, sodass redundanter Code verringert wurde. Jedoch ergab sich daraus auch ein Nachteil. Durch das sequenzielle Verändern der Daten innerhalb der Methode bestand die Gefahr, dass die ersten Veränderungen erfolgreich sind, aber beispielsweise beim Zuweisen der zuständigen Person ein Fehler auftritt. In diesem Fall wäre das Update nur teilweise erfolgreich. In der früheren Version, als alle Updates in einer Abfrage stattfanden, wäre in so einem Fall das gesamte Update fehlgeschlagen. Ein mögliches Teilupdate kann zu unerwarteten Fehlern führen und ist für den Benutzer des Systems entweder nur schwer zu erkennen oder unverständlich. Um dieses Verhalten wiederherzustellen, wurden letztlich alle Anweisungen in einer Transaktion [30] zusammengefasst. Hierdurch konnte der Vorteil der besseren Codequalität bestehen bleiben und dennoch das gewünschte Verhalten erzielt werden. Im Falle eines Fehlers macht Prisma nun automatisch die bisher vorgenommenen Änderungen rückgängig, sodass keine inkonsistenten Daten entstehen können. Der neue Code ist deutlich lesbarer (siehe Code 5).

```
1  async update(moduleDto: ModuleDto) {
2    const {
3      id,
4      responsibleId,
5      responsible,
6      requirementsHardId,
7      requirementsSoftId,
8      requirementsHard: requirementsHardNew,
9      requirementsSoft: requirementsSoftNew,
10     degreeProgramId,
11     groupId,
12     group,
13     translations,
14     subModules,
15     ...moduleData
```

ts

```
16   } = moduleDto;
17
18   await this.prisma.$transaction(async (prisma) => {
19     await prisma.module.update({
20       where: { id },
21       data: moduleData
22     });
23
24     await this.updateRequirements(prisma, moduleDto);
25     await this.upsertModuleTranslations(prisma, moduleDto);
26     await this.connectResponsible(prisma, moduleDto);
27     await this.connectSubModules(prisma, moduleDto);
28     await this.connectGroup(prisma, moduleDto);
29   });
30 }
31
32 async connectResponsible(prisma: any, moduleDto: ModuleDto) {
33   const { responsibleId, id } = moduleDto;
34
35   if (!responsibleId) return;
36
37   await prisma.module.update({
38     where: { id },
39     data: {
40       responsible: responsibleId ? {
41         connect: { id: responsibleId }
42       } : undefined
43     }
44   });
45 }
```

Code 5: Backend: Erstellen und Bearbeiten von Modulen (Verbessert)

### Changelog

Für die Anzeige der Änderungen (F22) müssen die Änderungen zunächst im Backend ermittelt werden.

Im gezeigten Codebeispiel werden zunächst alle verfügbaren Felder eines Moduls ermittelt. Hierzu wird die Methode `Object.keys` [31] verwendet, welche ein Array aller Felder des

Moduls zurückgibt. Die erhaltenen Felder werden im Anschluss in die Kategorien „Arrays“, „Übersetzungen“ und „Primitive Felder“ aufgeteilt (Code 6).

```
1  const fields = Object.keys(unchangedObject);
2
3  fields.forEach(field => {
4    if (Array.isArray(unchangedObject[field])) {
5      if (field === "translations") {
6        compareTranslations(unchangedObject, newObject, baseFieldName);
7      } else {
8        compareArrayField(unchangedObject, newObject, baseFieldName,
9          field);
10     }
11   } else {
12     comparePrimitiveField(unchangedObject, newObject, baseFieldName,
13       field);
14   }
15 });
```

Code 6: compareFields

Damit die ermittelten Felder jetzt verglichen werden können, sind verschiedene Vergleiche nötig, die im Folgenden kurz erklärt werden. Die Implementierungen der Compare-Methoden sind im Anhang (Code 7) zu finden.

Wenn das Feld einen primitiven Datentyp hat, kann der Feldinhalt miteinander verglichen werden (Code 7, Zeile 29). Wenn es sich bei dem Feld jedoch um ein Array handelt, wird der Vergleich etwas komplizierter. In dem Fall muss herausgefunden werden, ob es neue Einträge gibt oder ob Einträge entfernt wurden. Hierzu müssen dann die Ids verglichen werden (Code 7, Zeile 13). Für das Array translations, in dem die übersetzten Texte abgelegt werden, gibt es eine weitere Ausnahme. Hierbei müssen die Inhalte der Objekte im Array miteinander verglichen werden, die zu derselben Sprache gehören. Hierzu wird das entsprechende zu vergleichende Objekt mithilfe der filter-Methode [32] herausgesucht (Code 7, Zeile 3) und die darin enthaltenen Eigenschaften verglichen.

```

1  const compareTranslations = (unchangedObject: any, newObject: any,
   baseFieldName: string) => {
2    unchangedObject.translations.forEach((oldTranslationObject: any,
   index: any) => {
3      const newTranslationObject =
   newObject.translations.find((newTranslation: any) =>
   newTranslation.languageId === oldTranslationObject.languageId);
4      const languageAbbreviation = languages.find(l => l.id ===
   oldTranslationObject.languageId)?.abbreviation;
5
6      if (newTranslationObject && languageAbbreviation) {
7          compareTranslationFields(oldTranslationObject,
   newTranslationObject, baseFieldName, languageAbbreviation);
8      }
9    });
10 };
11 ...
12
13 const compareArrayField = (unchangedObject: any, newObject: any,
   baseFieldName: string, field: string) => {
14   const unchangedObjectIds = unchangedObject[field].map((obj: any)
   => obj.id);
15   const newObjectIds = newObject[field].map((obj: any) => obj.id);
16
17   // Check if the arrays contain the same elements, ignore the order
18   if (JSON.stringify(unchangedObjectIds.sort()) !==
   JSON.stringify(newObjectIds.sort())) {
19     changes.push({
20       field: `${baseFieldName}.${String(field)}`,
21       oldValue: unchangedObjectIds,
22       newValue: newObjectIds
23     });
24   }
25 };
26
27
28

```

ts

(1)

```
29 const comparePrimitiveField = (unchangedObject: any, newObject:
any, baseFieldName: string, field: string) => {
30   if (unchangedObject[field] !== newObject[field]) {
31     changes.push({
32       field: `${baseFieldName}.${String(field)}`,
33       oldValue: unchangedObject[field],
34       newValue: newObject[field]
35     });
36   }
37 };
```

Code 7: Vergleichsmethoden

### Öffentliche Endpunkte

Des Weiteren ist es wichtig, zwischen öffentlichen und privaten Endpunkten zu unterscheiden. Damit User im Frontend auch ohne Anmeldung die Module ansehen können, müssen manche Endpunkte ohne Authentifizierung erreichbar sein. Hierzu wurde ein eigener Dekorator (Code 8) erstellt. Dieser kann einfach über einen Endpunkt geschrieben werden, um diesen als öffentlich zu markieren (Code 9). Damit dies funktioniert, musste zusätzlich der AuthGuard durch eine eigene Implementierung (Code 10) ersetzt werden. Diese neue Implementierung überprüft, ob in den Metadaten „isPublic“ steht. Wenn dies der Fall ist, kann die Anfrage mit `return true` genehmigt werden. Falls diese Metadaten nicht gesetzt sind, wird die ursprüngliche Implementierung von `canActivate` (Code 10, Zeile 14) aufgerufen, um zu überprüfen, ob ein gültiger Token mitgesendet wurde.

```
1 import { SetMetadata } from '@nestjs/common';
2 export const Public = () => SetMetadata('isPublic', true);
```

ts

Code 8: public.decorator.ts

```
1 @Public()
2 @Get('/:id')
3 findOne(@Req() request: Request, @Param('id') id: string) {
4     const language = (request.headers as any)['language'];
5     return this.moduleService.findOne(+id, language);
6 }
```

Code 9: module.controller.ts

```
1 export class JwtAuthGuard extends AuthGuard('jwt') {
2
3     canActivate(context: ExecutionContext) {
4
5         const isPublic = this.reflector.get<boolean>(
6             'isPublic',
7             context.getHandler(),
8         );
9
10        if (isPublic) {
11            return true;
12        }
13
14        return super.canActivate(context);
15    }
16 }
```

Code 10: jwt-auth.guard.ts

### Informationen über den aufrufenden User

Manche Endpunkte benötigen genauere Informationen über den aufrufenden User, also den User, der im Frontend angemeldet ist. Das Bearbeiten von Modulen soll beispielsweise nur Usern erlaubt sein, die entweder verantwortlich für den gesamten Studiengang sind oder verantwortlich für das bearbeitete Modul. Da sich die User mithilfe eines Jwt-Tokens authentifizieren, kann dieser hierzu einfach genutzt werden. In der jeweiligen Controller-Methode muss dann als Parameter lediglich `@User() user: UserDto` hinzugefügt werden, um dann mithilfe von `user.role` die Rolle des Users zu erfahren oder mit `user.id` die Id des Users.

Die Information über den User wird vom Framework NestJS beim Aufruf der Methode `canActivate` herausgefunden und wird dann automatisch als Parameter übergeben. In Code 10 ist jedoch zu sehen, dass der `@Public`-Decorator aufgrund des frühen Returns diese Anweisung überspringt. Für Methoden, die also für nicht angemeldete Benutzer zur Verfügung stehen sollen und die für angemeldete User anders funktionieren, funktioniert der `@User`-Parameter also nicht ohne weiteren Aufwand. Ein Beispiel hierfür ist die Auflistung aller Studiengänge. Diese sollen auch unangemeldeten Besuchern angezeigt werden, jedoch sollen angemeldete studiengangsverantwortliche Personen auch versteckte Studiengänge angezeigt bekommen.

Damit der User auch bei öffentlichen Endpunkten verfügbar ist, gibt es zwei Möglichkeiten. Als erste Möglichkeit könnte ein weiterer Guard erstellt werden (siehe Code 11). [14] Dieser Guard kann mithilfe des `Use-Guards-Decorator` aktiviert werden (Code 12).

```
1 @Injectable()
2 export class InjectUser extends AuthGuard('jwt') {
3   handleRequest(err: any, user: any) {
4     return user;
5   }
6 }
```

Code 11: InjectUser-Implementierung

```
1 @UseGuards(InjectUser)
2 @Public()
3 @Get('/:id')
4 findOne(@User() user:UserDto, @Req() request: Request, @Param('id') id:
  string) {
5 [...]
```

Code 12: findOne() in department.controller.ts

Eine zweite Möglichkeit wäre, den Aufruf von `canActivate` vorzuziehen (Code 13). Wenn dieser direkt als Erstes durchgeführt wird, wird in jedem Fall der User verfügbar gemacht und öffentliche Endpunkte sind trotzdem möglich. Diese Möglichkeit hat den Vorteil, dass nicht daran gedacht werden muss, wie in Code 12 den `InjectUser`-Guard an die verschiedenen Methoden zu setzen. Aus diesem Grund wird diese Möglichkeit favorisiert.

Da `canActivate` eine Exception wirft, falls der User nicht eingeloggt ist, muss diese abgefangen werden, damit trotzdem geprüft werden kann, ob es den `@Public`-Decorator gibt. Außerdem kann `canActivate` ein `Observable` zurückgeben. Da der Rückgabewert

jedoch ein Boolean-Wert sein muss, wird die Methode `lastValueFrom()` aus der `rxjs-`Bibliothek genutzt, um einen konkreten Wert zu erhalten. [33]

```
1  async canActivate(context: ExecutionContext): Promise<boolean> { ts
2    // canActivate() will inject the context of the current request
3    // we need to call it first to be able to access the user later
   (@User() user:UserDto)
4    let canActivate = super.canActivate(context);
5    try {
6      if (isObservable(canActivate)) {
7          canActivate = await lastValueFrom(canActivate as
Observable<boolean>);
8      } else {
9          canActivate = await canActivate;
10     }
11 } catch (UnauthorizedException) {
12     canActivate = false;
13 }
14
15 const isPublic = this.reflector.get<boolean>(
16     "isPublic",
17     context.getHandler()
18 );
19
20 if (isPublic) {
21     return true;
22 }
23
24 if (context.getHandler().name === "login") return true;
25 if (context.getHandler().name === "license") return true;
26
27 return canActivate;
28 }
```

Code 13: `canActivate()` - Verbessert

### 4.1.3. PDFs generieren

Das Generieren einer Modulbeschreibung in Form einer PDF-Datei verläuft in mehreren Teilschritten:

1. Die Studiengangverantwortliche Person (User) fragt neue PDF-Version an
2. Das Backend erstellt aus den vorliegenden Informationen eine `.tex`-Datei und veröffentlicht einen Auftrag zur Kompilierung
3. Ein Kompilierungs-Server nimmt den Auftrag an und ruft die `.tex`-Datei aus dem Backend ab
4. Der Kompilierungs-Server kompiliert die `.tex`-Datei zu einer PDF-Datei und gibt diese ans Backend zurück
5. Das Backend meldet das Ergebnis an den User zurück
6. Der User prüft das Ergebnis und gibt es frei
7. Die neue PDF-Version steht bereit

In den Schritten 2 bis 5 werden eine `.tex`-Datei und eine `.pdf`-Datei generiert. Dies wird im Folgenden näher beschrieben.

#### Generierung der `.tex`-Datei

Die Grundlage für die Generierung der `.tex`-Datei bildet ein Python-Skript, welches von Prof. Dr. Felix Heine bereitgestellt wurde. Dieses Script wurde im Rahmen dieser Arbeit an die veränderte Datenstruktur angepasst. Anschließend wurde das Skript in TypeScript umgewandelt und in das neue Backend eingebaut. Hierzu wurde das Datenbankschema um die benötigten Einträge erweitert und es wurde ein neuer Controller mit dazugehörigem Service eingesetzt, welcher die Endpunkte und Methoden zur Generierung der `.tex`-Datei anbietet.

Das ursprüngliche Python-Skript enthielt eine statische Auflistung der Felder in der Tabelle des Modulhandbuchs. Darin enthalten war der gewünschte Text (z. B. Moduldauer) und das dazugehörige Feld (`module.courseLength`). Diese statische Auflistung wurde im Rahmen der Umstellung in die Datenbank verschoben. Hierdurch soll der Wartungsaufwand reduziert werden. Mehrere Auslöser können dazu führen, dass sich die Struktur des zu generierenden PDFs ändert. In Zukunft könnte neben Englisch und Deutsch eine zusätzliche Sprache angeboten werden. Auch ist es denkbar, dass zusätzliche Felder eingeführt werden. Für die Studiengänge an der Fakultät 3 wird beispielsweise die Information „Gewichtung“ auf den Modulhandbüchern abgedruckt. Um nun verschiedene PDF-Strukturen für verschiedene Studiengänge anzubieten, ist nun nur noch eine Veränderung der Datensätze in der Datenbank notwendig. Hierfür könnte eine zusätzliche Oberfläche erstellt werden, mit der die User selbst die PDF-Struktur konfigurieren können.

In Code 14 ist zu sehen, wie der LaTeX-Code für ein Submodul generiert wird. Ein reduziertes Beispiel für ein StructureItem ist in Code 15 zu sehen. Mit der Eigenschaft

takeTwoColumns (Code 15, Zeile 3) können gewünschte Eigenschaften in Code 14, Zeile 7 breiter dargestellt werden. Dies passiert im Modulhandbuch der Abteilung Informatik beispielsweise für das Feld E14, da dort in der Regel viel Text enthalten ist. In Code 15, Zeile 14 ist zu sehen, wie an die gespeicherte Information ein Suffix angehängen werden kann. Dies ist beispielsweise wie im gezeigten Beispiel für Zeitdauern interessant, wird aber auch für die Anzeige der Dauer (z. B. 1 Semester) genutzt. In Code 15, Zeile 21 ist abschließend die Übersetzung für die Zeilenüberschrift angegeben.

```
1 private async appendStructureItems(items: any[], entity: any, ts
2   languageAbbreviation: string): Promise<string> {
3   let latexString = "";
4   for (const line of items) {
5     const col1 = line.translations[activeTranslationIndex].name;
6     const col2 = await this.getValue(entity, languageAbbreviation,
7     line.paths as PdfStructureItemPathIncludingField[]);
8
9     if (!line.takeTwoColumns) {
10      latexString += `\\textbf{${col1}} & ${col2} \\\\ \\n`;
11    } else {
12      latexString += `\\multicolumn{2}{p{16.5cm}}{ \\textbf{${col1}} }
13      \\newline ${col2} } \\\\ \\n`;
14    }
15  }
16  return latexString;
17 }
```

Code 14: appendStructureItems()

```
1 { json
2   "position": 7,
3   "takeTwoColumns": false,
4   "paths": [
5     {
6       "field": {
7         "path": "hoursPresence",
8         "suffix": " h / "
9     }
10  }
```

```
10     },
11     {
12         "field": {
13             "path": "hoursSelf",
14             "suffix": " h"
15         }
16     }
17 ],
18 "translations": [
19     {
20         "id": 39,
21         "name": "Präsenzstunden / Selbststudium",
22         "languageId": 1,
23         "pdfStructureItemId": 20
24     }
25 ]
26 }
27
```

Code 15: Beispiel eines StructureItems

Damit zu den Zeilenüberschriften auch die dazugehörigen Werte gedruckt werden können, wird die Methode `getValue` genutzt (Code 14, Zeile 5). Hierzu wird die Eigenschaft `path` genutzt, die jedes Field besitzt (Code 15, Zeile 7). Im genannten Beispiel ist die Ermittlung noch einfach - `hoursPresence` ist ein Feld im Submodul und kann daher einfach ausgelesen werden. Es gibt jedoch auch komplexere Fälle, die betrachtet werden müssen. Es gibt zum Beispiel den Pfad `responsible.firstName`, bei dem auf das Attribut `firstName` des Objekts `responsible` zugegriffen wird. Im Falle der `translations` wird außerdem ein Zugriff auf ein Array benötigt (z. B. `requirementsSoft.translations[0].name`). Als letzte Art gibt es die Methodenaufrufe. Für das Feld E5 wird beispielsweise eine Auflistung der Submodule benötigt. Diese benötigt zur Bereitstellung der gewünschten Informationen einen Zugriff auf die Datenbank und wird daher in einer eigenen Methode absolviert. Im Feld `path` steht dann der Name der aufzurufenden Methode „`submodules()`“. Manche dieser Methoden könnten asynchron sein, weshalb hier auch noch einmal unterschieden werden muss (siehe Code 16, Zeile 14). Die Methoden haben alle dieselbe Signatur, damit sie auf die gleiche Art aufgerufen werden können.

Der Wert für das komplexeste Beispiel „`requirementsSoft.translations[0].name`“ wird wie folgt ermittelt:

1. Die Informationen des Teilmoduls in eine Variable (submodule) laden
2. Den Pfad bei jedem Punkt trennen
3. Für den ersten Teil des Pfades (requirementsSoft) den Wert aus der Variable submodule auslesen
4. Aus diesem neuen Objekt wird der zweite Teil des Pfades ausgelesen (translations[0])
5. Zum Schluss wird aus dem neuen Objekt dann noch die Eigenschaft „name“ ausgelesen
6. Alle Segmente des Pfades wurden abgelaufen, sodass das Ergebnis des letzten Schrittes zurückgegeben werden kann

Die Methode `getNestedProperty` (Code 16, Zeile 23) sorgt für den soeben beschriebenen Ablauf. Mit der `reduce`-Methode (Code 16, Zeile 27) werden die einzelnen Segmente des Pfades abgelaufen.[34] Das `RegExp`-Muster (Code 16, Zeile 29) erkennt, ob das Segment ein Array ist und gibt das Segment, den Namen des Arrays sowie die angegebene Zahl zurück. Da das Segment nicht nochmal benötigt wird, wird es mithilfe des Unterstriches in Code 16, Zeile 33 verworfen. [28]

```
1 private async getValue(module: any, lang: string, paths: PdfStructureItemPathIncludingField[]): Promise<string> {
2     let result = "";
3
4     for (const path of paths) {
5         result += await this.getValueFromPath(module, lang, path.field.path ??
6         "") + " " + path.field.suffix;
7     }
8     return this.formatForLatex(result, lang);
9 }
10
11 private async getValueFromPath(module: any, lang: string, path: string): Promise<any> {
12     if (path in this.specialPaths) {
13         const result = this.specialPaths[path](module, lang);
14         if (result instanceof Promise) {
15             return await result;
16         }
17         return result;
18     } else {
19         return this.getNestedProperty(module, path);
20     }
```

```
21 }
22
23 private getNestedProperty(obj: any, path: string): any {
24     const pathSegments = path.split(".");
25
26     // Use reduce to traverse the object structure
27     return pathSegments.reduce((currentValue, segment) => {
28         // Check if the part matches the pattern "arrayName[index]"
29         const match = segment.match(/^(\\w+)\\[(\\d+)\\]$/);
30
31         if (match) {
32             // match consists of the full match, the array name and the index
33             const [, arrayName, index] = match;
34             return currentValue && currentValue[arrayName] &&
currentValue[arrayName][parseInt(index)];
35         }
36
37         return currentValue && currentValue[segment];
38     }, obj);
39 }
40
41 private getSubmodules(module: any, lang: string): string {
42     if (module.subModules && module.subModules.length > 0) {
43         return module.subModules.map((subModule: any) => {
44             return ` ${subModule.abbreviation}$\\quad$
45             ${subModule.translations[activeTranslationIndex].name},
46             ${this.translations[lang]?.submoduleRequired.true}`;
47         }).join(" \\\\ ");
48     } else {
49         return "-";
50     }
51 }
```

Code 16: getValue-Methoden

### Generierung der .pdf-Datei

Für die erfolgreiche Kompilierung der .tex-Datei muss die Server-Umgebung konfiguriert werden. Zusätzlich zur Möglichkeit, LaTeX kompilieren zu können, müssen die von der .tex-Datei benötigten Pakete bereitstehen. Um diesen Aufwand zu verringern, wird ein Docker-Image eingesetzt, welches es ermöglicht, ohne weiteren Konfigurationsaufwand per HTTP-Request eine .tex-Datei zu kompilieren [35].

Außerdem wurde ein neues Python-Skript erstellt, welches einen Kompilierungs-Server implementiert. Das Skript prüft in regelmäßigen Abständen, ob in der API ein Auftrag zur Erstellung einer neuen PDF-Datei vorliegt. Wird bei der Prüfung ein Auftrag gefunden, markiert der Server diesen als „laufend“, damit dies auch im Frontend ersichtlich ist und damit kein anderer Server den Auftrag übernimmt. Anschließend ruft der Server die .tex-Datei aus dem Backend ab und übergibt sie an den Docker-Container. Der Docker-Container erstellt nun die PDF-Datei und gibt diese an den Server zurück. Der Server übermittelt das Ergebnis zurück an das Backend. [36] Die Erstellung des PDFs ist entkoppelt vom Backend. Somit können beispielsweise mehrere Server betrieben werden, um die PDF-Erstellung zu parallelisieren. Auch können die Server unterschiedlich implementiert werden, sodass auch andere Dateiformate umsetzbar wären. Zukünftige Implementierungen könnten beispielsweise Word-Dokumente in ein PDF-Dokument umwandeln oder LaTeX-Alternativen wie zum Beispiel Typst [37] in ein PDF-Dokument kompilieren. Beim Hinzufügen weiterer Kompilierungs-Server muss das Backend nicht angepasst werden, da dieses lediglich die Aufträge anbietet und die Aufträge nicht selbst an konkrete Server zuweist.

## 4.2. Frontend

Im Gegensatz zum Backend ist das Frontend eine neue Anwendung, zu der es keinen Bestandscode gab. Für das Frontend wurde ein neues Angularprojekt erstellt und mithilfe des Paketmanagers npm [38] die benötigten Pakete hinzugefügt, von denen in den folgenden Unterabschnitten noch einige vorgestellt werden. Weiterhin werden in den folgenden Abschnitten anhand einiger Beispiele vorgestellt, wie das Frontend implementiert ist.

### 4.2.1. Routing

Der Code der Anwendung besteht aus Komponenten und Services. Die Komponenten werden in der Anwendung dargestellt und werden modular genutzt. Die Services können von allen Komponenten genutzt werden und bieten Methoden zum Laden von Daten aus dem Backend an. Damit die Services wissen, unter welcher URL das Backend erreichbar ist, ist diese URL in einer Datei `environment.ts` gespeichert und kann von da abgerufen werden. Dies ermöglicht den schnellen Austausch dieser URL. Für den produktiven Einsatz gibt es zudem eine `environment.prod.ts`, die die URL des produktiven Backends enthält.

Sobald das Frontend mit dem Befehl `ng build --configuration=production` in der Produktiv-Version kompiliert wird, sorgt ein Eintrag in der `angular.json` (Code 17) dafür, dass die Environment-Datei entsprechend ausgetauscht wird.

Die Entscheidung, welche Komponente in der Anwendung gezeigt wird, wird vom RouterModule übernommen. In der `main.component.ts` der Anwendung (Code 18) wird lediglich die Topbar (welche auf jeder Seite gezeigt werden soll) und das Router-Outlet platziert.

Das Router-Outlet wird dann in Abhängigkeit der besuchten URL anhand eines Eintrages aus der `app.routes.ts` (Code 19) mit der dort gesetzten Komponente ausgetauscht.

```
1 "configurations": {
2   "production": {
3     "fileReplacements": [{
4       "replace": "src/environments/environment.ts",
5       "with": "src/environments/environment.prod.ts"
6     }]
7   }
8 }
```

Code 17: angular.json (Auszug)

```
1 <div class="main">
2   <app-topbar></app-topbar>
3   <router-outlet />
4 </div>
```

Code 18: main.component.html

```
1 export const routes: Routes = [
2   { path: 'overview', component: ModuleGridComponent },
3   { path: 'faculties', component: FacultiesComponent },
4   { path: 'faculty/:id', component: FacultyDetailComponent }
5   [...]
```

Code 19: app.routes.ts (Auszug)

Das Wechseln auf eine andere Seite ist nun sehr einfach mithilfe eines Methodenaufrufes möglich. Um beispielsweise auf die Seite `/faculty/4/department/2/course/1/module/1` zu wechseln, wird dieser Aufruf benötigt:

```
await this.router.navigate(['faculty', module.facultyId, 'department',  
module.departmentId, 'course', module.courseId, 'module', module.id]);
```

Verkürzen lässt sich dies dann noch, falls ein Teil der URL bereits korrekt ist. Befindet man sich beispielsweise bereits auf der Seite `/faculty/4/department/2/course/1` und möchte nur noch `/module/1` anhängen, kann das Argument „relativeTo“ genutzt werden, sodass nicht mehr alle Segmente angehängt werden müssen: `await this.router.navigate(['module', module.id], {relativeTo: this.route});`

### 4.2.2. Design

In diesem Unterabschnitt wird erklärt, wie das Design des Frontends implementiert wurde. Hierzu wird zunächst die Entscheidung für ein UI-Framework begründet. Anschließend wird die Arbeit mit SASS-Dateien und die Umsetzung des responsiven Designs erklärt.

#### UI-Framework

Ein UI-Framework kann bei der Implementierung des Frontends unterstützen. Vom Framework angebotene vorgefertigte Komponenten müssen nicht selbst implementiert werden. Nach dem Don't-Repeat-Yourself-Prinzip wird dadurch der Entwicklungsaufwand reduziert und die Codequalität verbessert. [17] Allerdings muss die Nutzung eines Frameworks sorgfältig abgewogen werden. Jede Einführung zusätzlicher Abhängigkeiten birgt gewisse Risiken. Das Framework könnte eine Sicherheitslücke beinhalten [39] oder könnte zu anderen (zukünftigen) Abhängigkeiten inkompatibel sein. Diese Risiken können durch eine vorherige Untersuchung der Frameworks verringert werden. Wenn ein Framework open source ist und eine große Anzahl an Unterstützern hat, ist es theoretisch möglich, dass Sicherheitslücken zügig gefunden und behoben werden [40]. Da dieses Projekt keine geheimen oder personenbezogenen Daten verarbeitet, wird das geringe Risiko, das mit der Verwendung eines Frameworks einhergeht, eingegangen - zugunsten einer angenehmeren Entwicklung und eines schöneren Ergebnisses.

Um das zu benutzende Framework zu identifizieren, wurden zunächst zwei bekannte Frameworks `@ng-bootstrap/ng-bootstrap` [41] und `primeng` [42] miteinander verglichen. Beide Frameworks haben hohe Downloadzahlen und eine gute Dokumentation, was ein Indikator dafür sein könnte, dass sie in diesem Projekt hilfreich sein könnten. Da PrimeNG weitaus mehr Komponenten anbietet, wird in diesem Projekt PrimeNG verwendet. Durch die Nutzung von PrimeNG sollte die Implementierung der verschiedenen Tabellen und Formulare weitaus effizienter sein. Außerdem sieht das System insgesamt einheitlich und modern aus, weil alle PrimeNG-Komponenten dem gleichen Theme folgen. [42]

#### Styling

Um die Darstellung der Inhalte von der Struktur der Inhalte zu trennen, werden in der Webentwicklung die Regeln zur Darstellung üblicherweise in CSS-Dateien definiert und dann in die HTML-Dateien eingebunden [43]. Für dieses System wurde SASS [44], eine

Erweiterung von CSS, verwendet, die zusätzlich Features einführt. Durch die Nutzung von Einrückungen kann der Code so etwas lesbarer gestaltet werden, da lange Selektoren übersichtlicher dargestellt werden können und nicht wiederholt werden müssen. [45]

Es wurde eine globale SASS-Datei erstellt, um Klassen, die an vielen Stellen verwendet werden sollen, zu definieren (siehe Code 20). Es gibt in der Anwendung viele klickbare Elemente. Um darzustellen, dass diese klickbar sind, soll sich der Cursor verändern, wenn er sich über dem Element befindet. Hierzu muss nun einfach die Klasse `.clickable` auf das Element gelegt werden. Neben `.clickable` gibt es noch zahlreiche weitere Klassen wie zum Beispiel (`fullWidth`, `smallPadding`, `mediumPadding` und `largePadding`), die für eine konsistente Darstellung in der Anwendung sorgen. Außerdem wurde `overwritePrimeNg.sass` erstellt, welche ausgewählte Eigenschaften von PrimeNG überschreibt. SASS erlaubt die Nutzung von Import-Statements, sodass die zusätzliche Datei einfach in der globalen SASS-Datei mit der Anweisung `@import "overwritePrimeNg"` importiert werden kann.

```
1  .centerText
2    text-align: center
3
4  .clickable
5    cursor: pointer
6
7  .flex
8    display: flex
9    justify-content: center
10   align-items: center
```

sass

Code 20: Auszug styles.sass

Des Weiteren hat jede Komponente nochmal eine eigene `.sass`-Datei, in der komponentenbezogene Darstellungseigenschaften gesetzt werden können. Diese `.sass`-Dateien werden nur auf die Elemente in der Komponente angewendet, sodass alle anderen Komponenten unverändert bleiben. [46]

### Responsive Design

In verschiedenen Gesprächen mit zukünftigen Nutzern stellte sich heraus, dass die administrativen Oberflächen auf Desktop-PCs bedient werden. Für die administrativen Oberflächen muss die Ansicht auf mobilen Geräten also nicht optimiert werden. Die Übersicht der Studiengänge, die Übersicht der angebotenen Module sowie die moderne Ansicht der Moduldetails könnte jedoch von Studierenden auf mobilen Geräten geöffnet werden und sollte daher für die mobile Ansicht optimiert werden (N16).

Damit die Anzeige der Studiengänge (Abbildung 18) auf mobilen Geräten gut aussieht, musste nicht viel angepasst werden. Statt die Karten der Studiengänge nebeneinander aufzureihen, werden diese in der mobilen Ansicht nun untereinander platziert. Für die Desktopansicht wird hier ein Grid [47] mit drei Spalten genutzt. Damit es in der mobilen Ansicht nur eine Spalte gibt, kann der Wert `grid-template-columns` mithilfe von `unset` [48] zurück auf den ursprünglichen Wert gesetzt werden (siehe Code 21). Für die Erkennung, ob die mobile Ansicht benötigt wird, wird eine Media Query genutzt. Diese sorgt dafür, dass die genannte Spalteneigenschaft nur für Geräte mit einer Bildschirmbreite unter 850 Pixeln zurückgesetzt wird. [49]

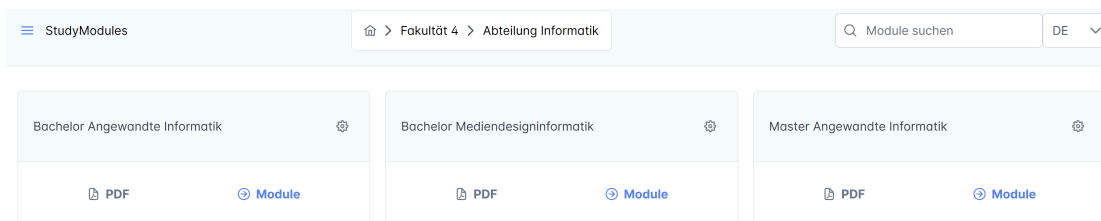
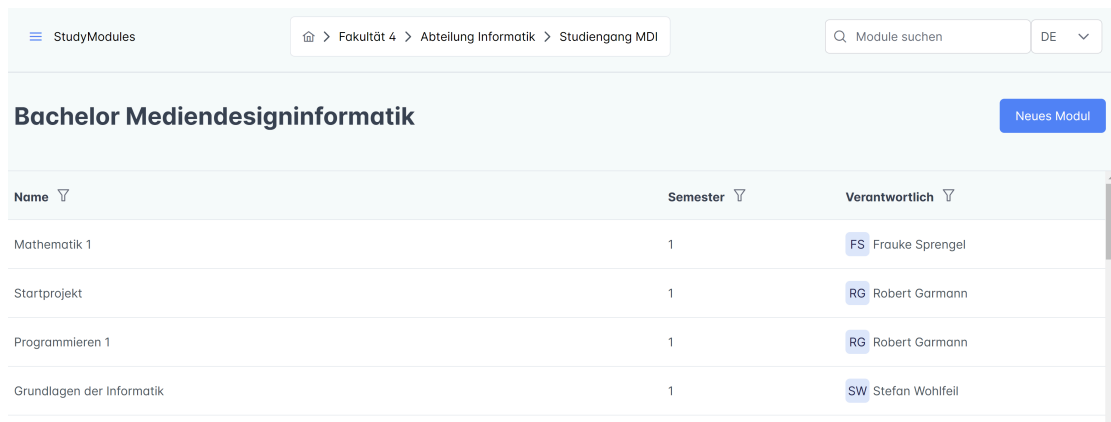


Abbildung 18: Übersicht der Studiengänge

```
1 @media (max-width: 850px) sass  
2   .courses  
3     grid-template-columns: unset
```

Code 21: courses.component.sass

Die Anzeige aller Module ist in der Desktopansicht eine Tabelle (Abbildung 19). Diese wird auf mobilen Endgeräten zwar dargestellt, jedoch muss von links nach rechts gescrollt werden, um alle Felder sehen zu können. Um diese Ansicht zu optimieren, kann eine Funktion von PrimeNG genutzt werden, um für mobile Geräte aus der Tabelle eine Auflistung von Karten zu machen. Das Setzen von `responsiveLayout="stack"` reicht hier aus. Um die Felder auf den mobilen Karten noch zu beschriften, muss für jedes Feld eine Beschriftung hinzugefügt werden. Die Klasse `p-column-title` muss dabei auf die entsprechende Beschriftung gesetzt werden, damit PrimeNG sie passend ein- und ausblendet. [50]



Name	Semester	Verantwortlich
Mathematik 1	1	FS Frauke Sprengel
Startprojekt	1	RG Robert Garmann
Programmieren 1	1	RG Robert Garmann
Grundlagen der Informatik	1	SW Stefan Wohlfel

Abbildung 19: Übersicht aller Module

Zuletzt muss noch die Anzeige der Moduldetails optimiert werden. Diese ist etwas komplexer aufgebaut, da sie aus vielen verschiedenen Komponenten besteht. Die Komponenten in der ersten Reihe in Abbildung 20 sind beispielsweise Info-Card-Components und bestehen aus einem Icon, einem Label und einem Anzeigewert. Neben den Info-Cards gibt es noch Stat-Cards für die Anzeige des Kuchendiagramms (siehe Abbildung 20), Text-Cards für die Anzeige von einfachen Texten (z. B. Voraussetzungen) und Split-Text-Cards für die Anzeige von zwei verschiedenen Texten innerhalb einer Karte. Für die Optimierung bei kleinen Bildschirmgrößen können Info-Card und Text-Card ignoriert werden, da diese bereits vollständig anzeigbar sind. Die Split-Text-Card muss auf kleinen Bildschirmgrößen die Texte untereinander statt nebeneinander zeigen. Hierzu musste die Flex-Direction mithilfe einer Media Query auf column gesetzt werden. [51] Dies war auch bei der Stats-Card erforderlich. Hier wurden zusätzlich die Texte anders ausgerichtet, damit sie auf der Karte zentriert angezeigt werden.

Die einzelnen Karten sind in der Ansicht der Moduldetails in einem Grid angeordnet. Dieses hat in der Desktopansicht 5 Spalten. Die Info-Karten passen in eine Spalte, während sich Karten mit mehr Inhalt auf mehrere Spalten verteilen dürfen. Damit auf der mobilen Ansicht die Karten vollständig angezeigt werden können, ohne dass ein horizontales Scrollen notwendig ist, wird die Eigenschaft `grid-column` von jeder Karte auf 5 gesetzt, sodass es pro Zeile jeweils eine Karte gibt. [47]

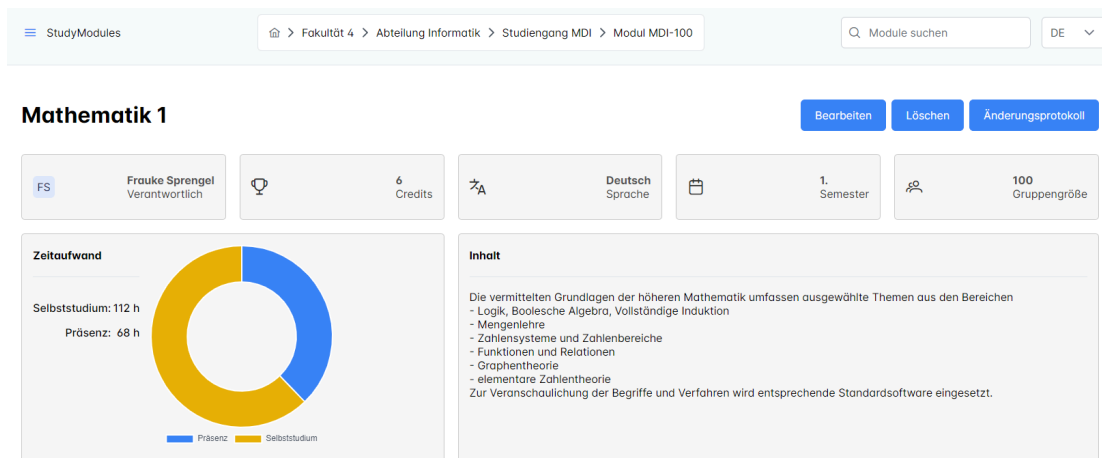


Abbildung 20: Desktopansicht Moduldetails

### 4.2.3. Übersetzbarkeit

Damit jede Komponente weiß, welche Sprache gerade dargestellt werden soll, wird ein Service genutzt (Code 22). Dieser kann per Dependency Injection [26] im Konstruktor einer beliebigen Komponente genutzt werden. Wenn in der Topbar (Abbildung 4) eine andere Sprache ausgewählt wird, wird die Eigenschaft `languageCode` im `LanguageService` verändert (`this.languageService.languageCode = selectedLanguageCode;`).

Der `LanguageService` hat eine Methode `getLanguages`, mit der an verschiedenen Stellen in der Anwendung angezeigt werden kann, welche Sprachen zur Auswahl stehen. Außerdem sind Getter und Setter für die Eigenschaft `languageCode` implementiert. Im Setter wird die ausgewählte Sprache in den `LocalStorage` geschrieben, damit die Benutzer der Website nicht bei jedem Besuch erneut ihre Sprache auswählen müssen. Außerdem wird die neu ausgewählte Sprache an das `languageSubject` weitergegeben. Das `languageSubject` ist ein `BehaviourSubject` aus der Erweiterungsbibliothek `RxJS` [52]. Nach dem Observer-Pattern-Prinzip können andere Komponenten und Services das `languageSubject` beobachten und erhalten im Falle einer Veränderung eine Benachrichtigung. Somit können beim Wechsel der Sprache die neuen Texte geladen werden.

```
1 export class LanguageService {
2     constructor(private http: HttpClient) {}
3
```

ts

```
4   private languageURL: string = environment.backendURL + "language";
5
6   languageSubject = new BehaviorSubject(this.languageCode);
7
8   getLanguages() {
9       return this.http.get<LanguageDto[]>(this.languageURL);
10  }
11
12  set languageCode(value: string) {
13      this.languageSubject.next(value);
14      localStorage.setItem('language', value);
15  }
16
17  get languageCode() {
18      return localStorage.getItem('language') ?? "de";
19  }
20 }
```

Code 22: language.service.ts

Für die Übersetzung der Website müssen zwei verschiedene Arten von Texten übersetzt werden. Die statischen Elemente der Website, wie zum Beispiel Button-Beschriftungen oder Tabellenüberschriften, ändern sich in der Regel nicht und stehen fest im Quellcode der Angular-Anwendung. Die dynamischen Texte, wie zum Beispiel Modulnamen, ändern sich abhängig vom angezeigten Modul und werden aus dem Backend empfangen. Diese beiden Arten an Texten werden auf verschiedene Weise übersetzt.

### Statische Elemente

Für den Wechsel zwischen verschiedenen Sprachen gibt es in Angular bereits zahlreiche Möglichkeiten. Bereits in Angular eingebaut ist das Paket `@angular/localize` [53]. Dieses bietet die Möglichkeit, die Angular-Website für verschiedene Sprachen zu kompilieren. Mithilfe unterschiedlicher (Sub-) Domains oder Unterordner können die verschiedenen Sprachversionen dann bereitgestellt werden. Für den Wechsel der Sprache ist dann allerdings der Wechsel auf eine andere URL und ein damit verbundenes Neuladen der Website nötig. Ein weiterer Nachteil ist die hohe Komplexität der Einrichtung sowie die mangelnde Flexibilität. Die entstehenden Übersetzungsdateien spiegeln die Struktur der HTML-Seiten wider, was bedeutet, dass bei Änderungen der HTML-Struktur auch die Übersetzungsdatei angepasst werden muss. Auch das Anlegen der Übersetzungsdatei ist dadurch nicht trivial. [54]

Ein simplerer Ansatz könnte die Nutzung von Key-Value-Paaren sein. Hierbei wird ein Key in die HTML-Seite geschrieben. Es gibt pro Sprache eine Übersetzungsdatei, die zu jedem Key einen Value zur Verfügung stellt, der die korrekte Übersetzung enthält. Ein Framework tauscht dann zur Laufzeit die Keys durch die gewünschten Values aus. Zur Auswahl eines geeigneten Frameworks wurden verschiedene Metriken betrachtet. Das genutzte Framework soll einfach zu nutzen sein, mit Key-Value-Dateien arbeiten können und eine gute Dokumentation haben. Auch sollten die Downloadzahlen auf NPM hoch genug sein, da ansonsten fraglich ist, ob das Paket die beste Wahl ist. Geringe Downloadzahlen können eine geringe Verbreitung bedeuten, was unter anderem dazu führen kann, dass das Paket keinen langfristigen Support erhält. Diese Anforderungen wurden von @ngx-translate/core[55] und von @jsverse/transloco [56] (früher unter @ngneat/transloco bekannt [57]) erfüllt. Da @ngx-translate/core jedoch nicht mehr weiterentwickelt wird, wurde für dieses System @jsverse/transloco genutzt.

Um nun die statischen Texte in verschiedenen Sprachen anzubieten, wurden die Dateien en.json und de.json in den Assets-Ordner des Frontends gelegt. Die Angular-Konfiguration wurde erweitert, sodass beim Kompilieren die Übersetzungsdateien in den Build-Ordner kopiert werden, sodass diese später auf dem Webserver bereitliegen. Beim Laden der Anwendung stellt Transloco automatisch sicher, dass die benötigte Sprachdatei per HTTP-Request vom Webserver geladen wird. Anschließend mussten nur noch die .html-Dateien angepasst werden, damit dort auch die Texte aus den Übersetzungsdateien geladen werden. Hierzu wurde ein neues Element zu jeder HTML-Seite hinzugefügt: `<ng-container *transloco="let t; prefix:'faculties'">`. Durch das optionale Präfix ist es möglich, Übersetzungen zu gruppieren. Um jetzt beispielsweise das Wort „Fakultätsübersicht“ auszugeben, muss `faculties: {overview: Fakultätsübersicht}` zu de.json hinzugefügt werden. Anschließend kann folgendes Element genutzt werden: `<span>{{t('overview')}}</span>`. [56]

### Dynamische Elemente

Damit die dynamischen Elemente übersetzt werden, benötigt das Backend die gewünschte Sprache. Diese wird im Header des GET-Requests übermittelt. Steht im Header `Language=DE`, werden deutsche Texte zurückgegeben. Damit nicht jeder HTTP-Request im Frontend manuell mit dem Header gefüllt werden muss, wird ein HTTPInterceptor [58] genutzt (Code 23). Dieser erhält die aktuell gesetzte Sprache aus dem LanguageService (Code 23, Zeile 5). Falls sich die dort eingestellte Sprache ändert, wird dies über ein BehaviourSubject kommuniziert und im LanguageInterceptor aktualisiert (Code 23, Zeile 7). In der Methode `intercept()` wird dann der Header auf die aktuell eingestellte Sprache gesetzt. Dies passiert nicht, falls der ursprüngliche Request bereits einen Language-Header hat, damit das Verhalten übersteuert werden kann. Dies ist beispielsweise für die Bearbeitungsmaske eines Moduls hilfreich, in der die englische Modulbeschreibung bearbeitet werden soll, ohne dafür die gesamte UI auf Englisch stellen zu müssen.

In der ApplicationConfig der Angular-Anwendung kann nun mithilfe einer Option konfiguriert werden, dass als HTTPInterceptor die neue Klasse LanguageInterceptor genutzt wird. Somit werden alle Requests mit der Sprache im Header erweitert.

Damit sich beim Wechsel der Sprache auch alle dynamischen Texte ändern, ist ein erneuter Aufruf der API notwendig. Hierzu wird das BehaviourSubject aus dem LanguageService (Code 22) genutzt. Somit können beim Sprachwechsel die dynamischen Informationen erneut aus dem Backend angefragt und in der Oberfläche dargestellt werden.

```
1  export class LanguageInterceptor implements HttpInterceptor {
2      private language: string;
3
4      constructor(private languageService: LanguageService) {
5          this.language = this.languageService.languageCode;
6          this.languageService.languageSubject.subscribe((language) => {
7              this.language = language;
8          });
9      }
10
11     intercept(req: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
12         if (req.headers.has('language')) {
13             return next.handle(req);
14         }
15
16         if (this.language) {
17             const newRequest = req.clone({
18                 setHeaders: {
19                     'language': this.language.toUpperCase()
20                 }
21             });
22             return next.handle(newRequest);
23         }
24         return next.handle(req);
25     }
26 }
```

Code 23: language.interceptor.ts

### 4.2.4. Erstellen eines neuen PDFs

Das Erstellen eines neuen PDFs kann in der Studiengangsübersicht gestartet werden (Abbildung 21). Hierbei öffnet sich ein Popup, welches anzeigt, wann für den Studiengang zuletzt ein PDF veröffentlicht wurde. In Zukunft könnten hier auch die Details der Veränderungen seit der letzten Veröffentlichung aus dem Changelog angezeigt werden. Im Popup kann ausgewählt werden, für welche Sprachen das PDF generiert werden soll (Abbildung 22). Während der Generierung wird dem User der Status angezeigt und sekundlich aktualisiert (Abbildung 23). Der User kann nun entweder auf das Ergebnis warten oder die Maske schließen und in einer separaten Maske die vergangenen Kompilierungsaufträge ansehen.

Sobald das PDF vorliegt, kann der User dieses ansehen und dann entweder verwerfen oder freigeben. Mit der Freigabe steht es dann auch für nicht angemeldete User bereit.

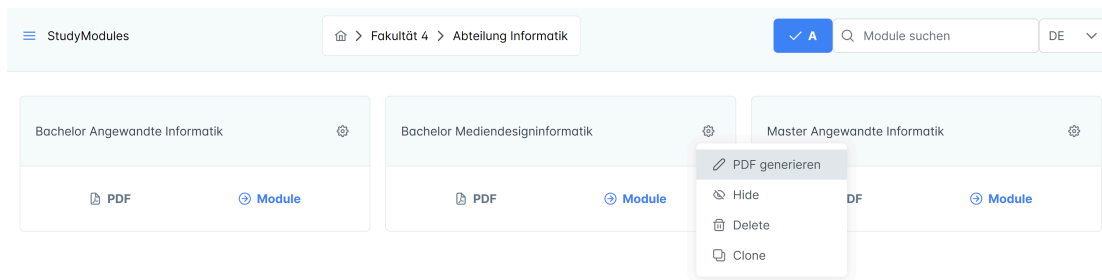


Abbildung 21: Menü - Studiengänge

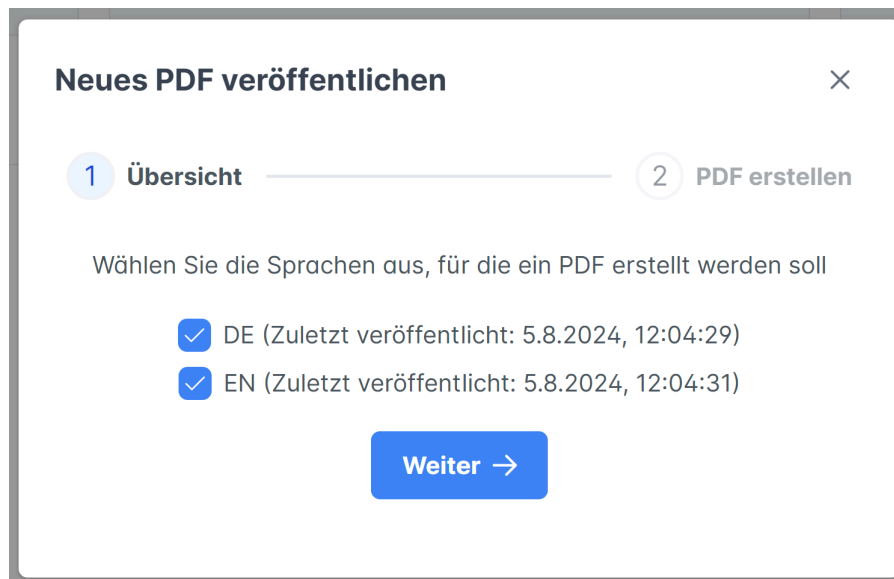


Abbildung 22: PDF veröffentlichen - Schritt 1

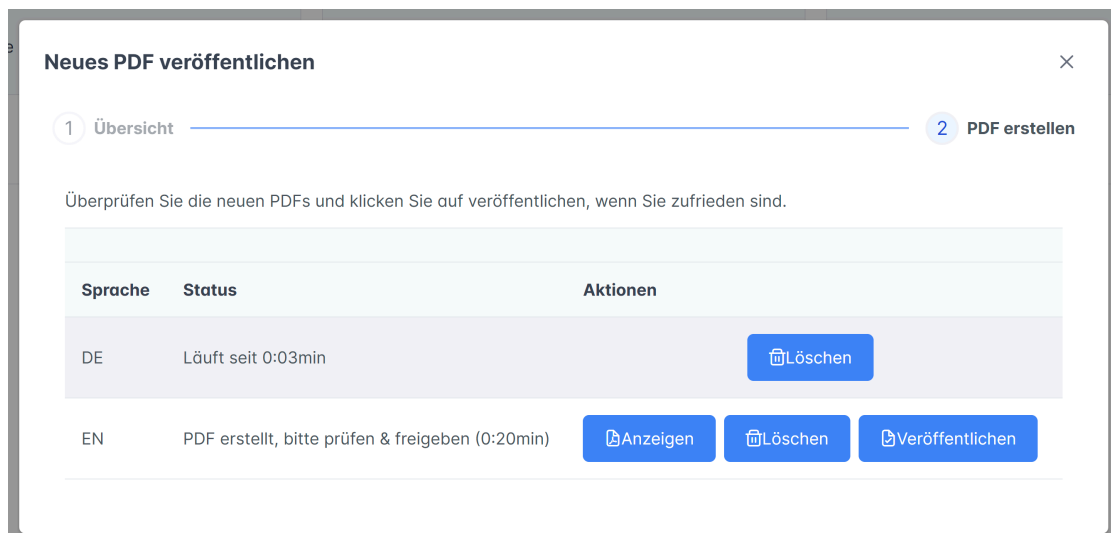


Abbildung 23: PDF veröffentlichen - Schritt 2

### 4.2.5. Subscriptions, Intervalle und Memory Leaks

In Unterabschnitt 4.2.4 wurde beschrieben, wie der Status des Auftrages sekundlich aktualisiert wird. Hierzu wird ein Intervall genutzt, welches sekundlich eine Anfrage an das Backend sendet. Damit dieses Polling stoppt, sobald die Maske geschlossen wird, muss es per Befehl gestoppt werden.

Außerdem werden beim Wechseln der Anzeigesprache die angezeigten Daten in jeder Maske neu geladen. Hierzu wird beim Öffnen der Maske wie in Unterabschnitt 4.2.3 beschrieben ein BehaviourObject genutzt. Auch dieses sollte beim Schließen der Maske deabonniert werden. Andernfalls gibt es die Gefahr von Memory Leaks - solange Events abonniert sind, kann die Maske nicht aus dem Arbeitsspeicher entladen werden. [59]

Aus diesen beiden Gründen muss in einigen Masken das Interface OnDestroy [59] implementiert werden. Dieses wird beim Schließen des Components aufgerufen. Der darin enthaltene Code (Code 24) ist dann recht simpel - das Intervall wird gestoppt und die Subscription auf das languageSubject wird deabonniert.

```
1 ngOnDestroy(): void {
2   if (this.intervalId) {
3     clearInterval(this.intervalId);
4   }
5
6   if (this.languageSubscription) {
7     this.languageSubscription.unsubscribe();
8   }
9 }
```

ts

Code 24: onDestroy Implementierung

### 4.2.6. Suchfunktion

Die Suchfunktion soll während der Eingabe geeignete Module vorschlagen (F5). Zur besseren Übersicht sollen diese nach Studiengang gruppiert werden. Hierzu müssen die aus dem Backend erhaltenen Daten auf das benötigte Format umgewandelt werden. Mit der `.map`-Funktion lässt sich in JavaScript der Aufbau eines Arrays verändern. [60] In Code 25 ist zu sehen, wie das neue Array zusammengebaut wird. Als Erstes werden die Studiengänge hinzugefügt. Für jeden Studiengang werden die darin enthaltenen Module hinzugefügt. Hierbei ist zu beachten, dass als `value` ein Link gesetzt werden muss, welcher beim Anklicken des Eintrages geöffnet werden soll.

```
1  this.courseService.getAll().subscribe(courses => {
2    this.groupedModules = courses.map(course => {
3      const courseLabel = course?.translations?.
[activeTranslationIndex]?.name;
4      const coursePath = `/faculty/${course.department.facultyId}/
department/${course.department.id}/course/${course.id}`;
5
6      const items = course.modules.map(module => {
7        return {
8          label: module.translations?.[activeTranslationIndex]?.name,
9          value: `${coursePath}/module/${module.id}`
10       };
11     });
12
13     return {
14       label: courseLabel,
15       items: items
16     };
17   });
18 });
```

Code 25: Daten für die Suche ermitteln

### 4.2.7. URL-basierte Erkennung von Seitenelementen

Wenn ein User ein Modul öffnet, ändert sich die URL. Für das Modul mit der Id 1 könnte der Pfad dann beispielsweise so aussehen: `/faculty/4/department/2/course/1/module/1`. Dies hat den Vorteil, dass der User den Link mit anderen Usern teilen kann. Außerdem funktionieren dadurch die Navigationstasten im Browser und der Browserverlauf zeigt hilfreiche Informationen.

Für die Implementierung von verschiedenen Funktionen ergibt sich außerdem ein weiterer Vorteil. Durch die Angabe verschiedener Informationen in der URL können diese auch im Code verwendet werden. Mithilfe der Informationen aus der URL wird zum Beispiel der Breadcrumb in der oberen Leiste (Abbildung 4) befüllt (N10). Damit diese Funktionalität gekapselt ist und sich nicht wiederholt, wurde hierfür eine eigene Methode entworfen (Code 26). Mithilfe des Aufrufs von `getIdFromSegment(„faculty“)` kann dann beispielsweise herausgefunden werden, durch welche Fakultät der User gerade navigiert. Hierzu teilt die Methode `getIdFromSegment()` die aktuelle URL in Segmente auf und gibt die Zahl nach

dem gewünschten Segment zurück. Wenn der Pfad `/faculty/4/department/2` wäre, würde `getIdFromSegment(„faculty“)` den Wert 4 zurückgeben.

```
1 getIdFromSegment(segment: string): string | undefined {
2   const segments = this.router.url.split("/");
3   const segmentIndex = segments.indexOf(segment);
4   return segmentIndex !== -1 ? segments[segmentIndex + 1] : undefined;
5 }
```

Code 26: url-segment.service.ts

### 4.2.8. Anlegen und Bearbeiten von Modulen

Die Masken zur Bearbeitung der Module und Teilmodule sind eine zentrale Stelle der Anwendung. Um hier eine gute Benutzbarkeit zu gewährleisten, sind mehrere Konzepte genutzt worden.

#### Übersetzbarkeit

Für die Anforderung der Übersetzbarkeit wurde in Abbildung 12, Abbildung 14 und Abbildung 15 eine Möglichkeit konzipiert, Texte in verschiedenen Sprachen zu hinterlegen. Nach erneuter Betrachtung des Problems ergab sich eine einfachere Lösung. Ein Nachteil der ursprünglichen Lösung war, dass es für jedes Eingabefeld ein Popup gab. Dies erhöhte den Aufwand der Eingabe drastisch, da für jede Eingabe ein neues Fenster geöffnet wurde und auch wieder geschlossen werden musste. Um die Anzahl der Popups zu verringern, wurden stattdessen gewöhnliche Textfelder genutzt. Um dennoch verschiedene Sprachen zu unterstützen, wird die Eingabemaske nun für jede Sprache einmal wiederholt. Eine Anzeige im oberen Bereich zeigt die einzelnen Schritte des Bearbeitungsprozesses (Abbildung 24).

1 Modulbeschreibung ausfüllen      2 Englische Texte ergänzen      3 Änderungen speichern

Nummer: 100      Name: Mathematik 1      **Modul MDI-100 Mathematik 1**

Untertitel: Mathematische Grundlagen der Informatik (MDI-MAT1)      **Untertitel:** Mathematische Grundlagen der Informatik (MDI-MAT1)

Credits: 6      Specialization: Grundlagenmodul      **Modulniveau:** Grundlagenmodul

Moduldauer: 1      Elective: Pflichtmodul      **Pflicht / Wahlpflicht:** Pflichtmodul

Responsible: FS Frauke Sprengel      **Teilmodule:** BIN-100-01 Mathematik 1, Pflicht

Submodules: 100-01      **Verantwortliche(r):** Prof. Dr. Frauke Sprengel

**Credits:** 6

**Moduldauer:** 1 Semester

**Voraussetzungen nach Prüfungsordnung:** keine

**Empfohlene Voraussetzungen:** keine

**Studien-/ Prüfungsleistungen:** Prüfung (Klausur oder mündliche Prüfung) und experimentelle Arbeit

**Angestrebte Lernergebnisse:** Formale Kompetenz: Kenntnisse der Logik und Vertrautheit mit mathematischen Formalismen zur Beschreibung von Sachverhalten Algorithmische und mathematische Kompetenz: Kennenlernen mathematischer Algorithmen, geeignete Lösungsverfahren für elementare Probleme der Mathematik und Informatik

**Weiter**

Abbildung 24: Modul bearbeiten

In Abbildung 25 sind die einzelnen Komponenten einer Bearbeitungsmaske (z.B. die Modulbearbeitung) zu sehen. Wenn auf „Bearbeiten“ gedrückt wird, lädt der Router das Edit-Component (z.B. module-edit.component.ts). Das Edit-Component erhält das zu bearbeitende Modul mit allen Eigenschaften über einen @Input-Parameter (siehe Code 27). Dieses Objekt wird an die folgenden Komponenten weitergegeben. Hierzu wird das Two-Way-Binding von Angular verwendet, damit das Objekt in beide Richtungen synchronisiert wird. Dadurch kann das Preview-Component bei Änderungen im Editor automatisch aktualisiert werden, was eine Live-Vorschau der Änderungen ermöglicht. Damit das Two-Way-Binding funktioniert, muss zusätzlich ein @Output-Parameter als EventEmitter definiert werden, der bei einer Änderung aufgerufen wird.

```

1 onModuleChange() {
2   this.moduleChange.emit(this.module);
3 }
4 @Input() module!: ModuleDetail;
5 @Output() moduleChange = new EventEmitter<any>();

```

Code 27: Two-Way-Binding

Außerdem lädt das Edit-Component für jede Sprache ein Translator-Component. Das Translator-Component hat die Aufgabe, die benötigte Sprache zu laden und im Translations-Array an die erste Stelle zu schieben. Dies ist notwendig, weil in den folgenden Editor- und Preview-Komponenten immer auf den ersten Eintrag im Array geschaut wird. Wenn im Preview-Component beispielsweise der Name des Moduls gezeigt werden soll, wird `module.translations[0].name` aufgerufen.

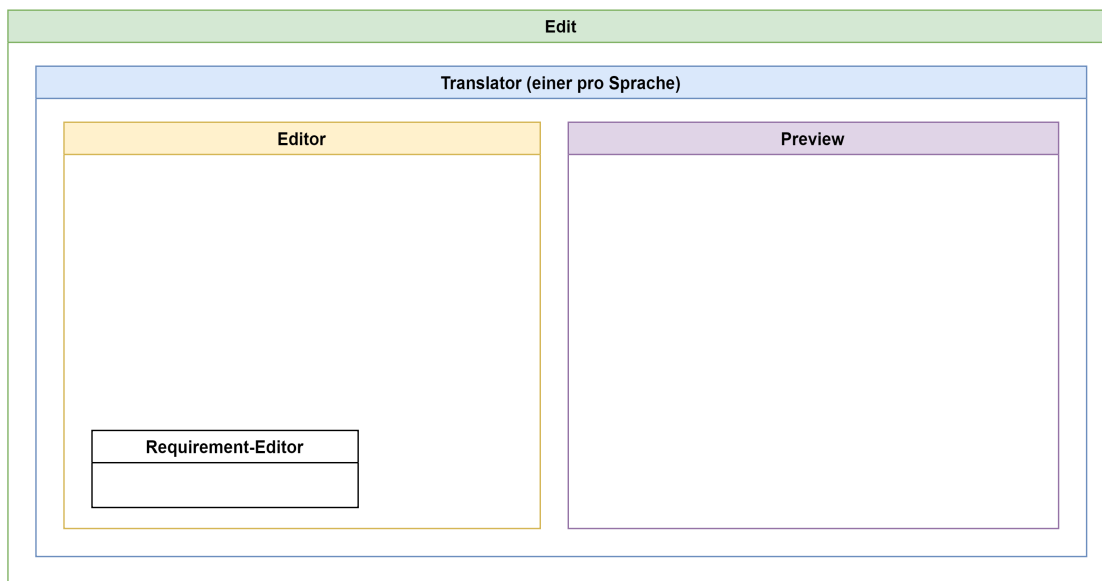


Abbildung 25: Komponenten der Modulbearbeitung

#### **Automatische Vervollständigung von Eingaben**

Um die Eingabe in Textfeldern zu erleichtern, wurde eine automatische Vervollständigung implementiert. In Textfeldern mit kurzen Texten (beispielsweise E13) erscheint ein Vorschlag, sobald der User mit der Eingabe beginnt. Es werden Texte vorgeschlagen, die in anderen Modulen für dasselbe Eingabefeld verwendet werden. Somit kann der User zum einen selbstständig einen neuen Wert eintragen oder alternativ einen vorgeschlagenen Wert auswählen, um nicht den gesamten Text immer wieder eingeben zu müssen. In Textfeldern mit längeren Texten wurde dies nicht implementiert, da sich deren Inhalte nicht oder nur sehr selten wiederholen (beispielsweise E25).

#### **Umwandlung von Textfeldern zu interaktiven Auswahl-Elementen**

Des Weiteren wurden, wie bereits in Abbildung 12 konzipiert, einige Textfelder umgewandelt. In den ursprünglichen Daten, die für diese Arbeit vorlagen, bestanden alle

Informationen aus Texten. Diese Informationen wurden teilweise umgewandelt, um deren Eingaben zu vereinfachen. Ein Beispiel hierfür ist die verantwortliche Person (E6), welche zuvor in einem Textfeld abgelegt wurde. In der implementierten Oberfläche gibt es nun ein Dropdown, in dem aus einer Liste an Personen ausgewählt werden kann. Im Backend wird dann nur die Id der Person gespeichert, statt des gesamten Textes. Auf eine ähnliche Art wurden diverse weitere Felder umgewandelt, sodass bei der Eingabe jetzt weniger Fehler gemacht werden können.

### **Plausibilitätschecks**

Eingabefehler können unentdeckt bleiben und sich somit mit der Zeit häufen. Um diesem Problem entgegenzuwirken, wurden (wie in F12 gefordert) Plausibilitätschecks implementiert. [20, Kapitel 10] Sobald alle Eingaben getätigt wurden, kann ein User per Klick auf „weiter“ auf die nächste Seite wechseln. Bevor jedoch gewechselt wird, werden die eingegebenen Daten überprüft. Fallen hierbei Unstimmigkeiten auf, wird der User darauf hingewiesen. Die entsprechenden Felder färben sich dann rot und zeigen in einem Tooltip den Grund dafür (Abbildung 26). Ein User kann entweder die Daten korrigieren oder die Warnung ignorieren und das Modul dennoch abspeichern. Das System soll vermeintlich falsche Eingaben nicht vollständig verhindern, sondern nur darauf hinweisen, da davon ausgegangen wird, dass das System von Experten bedient wird.

Neben der Prüfung, ob in jedes Feld ein Wert eingegeben wurde, gibt es folgende Überprüfungen:

1. Felder, die wie oben beschrieben eine Autovervollständigung haben, sollten einen Wert beinhalten, den es bereits gibt.
2. Die Abkürzung eines (Teil-) Moduls entspricht einem bestimmten Muster. Dieses wird mithilfe eines regulären Ausdrucks überprüft. Im Falle der Teilmodule wird dieser Ausdruck genutzt:  $/^[A-Z]{3}-[0-9]{3}-[0-9]{2}\$/$ . Die eingegebene Abkürzung muss mit drei Großbuchstaben (A-Z) beginnen, gefolgt von einem Bindestrich, drei Ziffern (0-9), einem weiteren Bindestrich und muss schließlich mit zwei Ziffern enden. Zeichen davor oder danach sind auch nicht zulässig. [61]
3. Die Abkürzung sollte eindeutig, also noch nicht vergeben sein.
4. Der angegebene Zeitaufwand (E8) muss zu den angegebenen ECTS (E7) passen. Ein ECTS entspricht laut StudAkkVO einem Zeitaufwand von 25 bis 30 Stunden. [3]
5. Das Feld Semester (E9) kann einen Bindestrich enthalten (z. B. 4-6). In dem Fall muss die zweite Zahl größer als die erste Zahl sein.

Untertitel

Mathematische Grundlagen der Informatik (MDI-MAT1)

Credits

0

Bitte geben Sie eine gültige Anzahl an Credits ein (>0).

Abbildung 26: Plausibilitätschecks

### 4.3. Dokumentation

Das neue System hat mehrere Komponenten, die zum einen bei der Einführung des Systems installiert werden müssen und zum anderen in Zukunft weiterentwickelt werden müssen. Um diese beiden Aufgaben zu unterstützen, wurden einige Informationen im Rahmen einer Dokumentation zusammengetragen. Die Dokumentation besteht aus vier Abschnitten: Frontend, Backend, API und Deployment. Die Abschnitte Frontend und Backend sind ähnlich aufgebaut. Beide enthalten Informationen zur erstmaligen Installation des Systems. Weiterhin ist beschrieben, wie neue Komponenten hinzugefügt werden können und was dabei zu beachten ist. Im Abschnitt API ist dahingegen beschrieben, wie die vom Backend bereitgestellte API genutzt werden kann, um verschiedene Datenabfragen durchzuführen. Kompliziertere Endpunkte sind hier außerdem aufgeführt. Es ist unter anderem beschrieben, wie der Login funktioniert und wie die Sprache der Antworten geändert werden kann. Im Abschnitt Deployment ist erklärt, wie das System mithilfe von Podman aufgesetzt werden kann. Die Erstellung der Dokumentation lief parallel zur Entwicklung des Systems (Abschnitt 4.1 und Abschnitt 4.2). Um sicherzustellen, dass die wichtigsten Informationen bereitstehen, wurde anschließend das System erneut auf einem Testserver mithilfe der Anleitungen aus der Dokumentation installiert.

Die Texte sind im Markdown-Format verfasst, was bedeutet, dass sie von zahlreichen Editoren unterstützt werden. Zudem können die Plattformen GitHub und GitLab die Markdown-Dateien darstellen, wenn ein Entwickler dort den Quellcode durchsucht.

Zusätzlich zu der Möglichkeit, die Markdown-Dateien mit einem beliebigen Texteditor oder einer der genannten Plattformen zu lesen, wurde in dieser Arbeit zusätzlich das

Framework Docusaurus [62] genutzt, um aus den Markdown-Dateien eine Website zu generieren (Abbildung 27). Diese Website kann genutzt werden, um durch die vollständige Dokumentation in einer Benutzeroberfläche zu navigieren. Der Aufbau der Website könnte auf Entwickler bekannt wirken, da manche Open-Source-Projekte ihre Dokumentation mithilfe von Docusaurus realisieren. [63]

Die Docusaurus-Website wurde überdies mit einer Suchfunktion erweitert, die von Algolia bereitgestellt wird. [64] Auch diese Suchfunktion sollte Entwickelnden bekannt sein, da sie in vielen Dokumentationen genutzt wird. Die Suchfunktion kann mit der Tastenkombination Strg/Ctrl+K aufgerufen werden. In die Suchfunktion können beliebige Texte eingegeben werden, um schnell relevante Dokumentationsinhalte zu finden. Damit zu den Texten dann geeignete Vorschläge gemacht werden können (Abbildung 28), wird ein Crawler von Algolia genutzt, welcher die Website in regelmäßigen Abständen indiziert. [65]

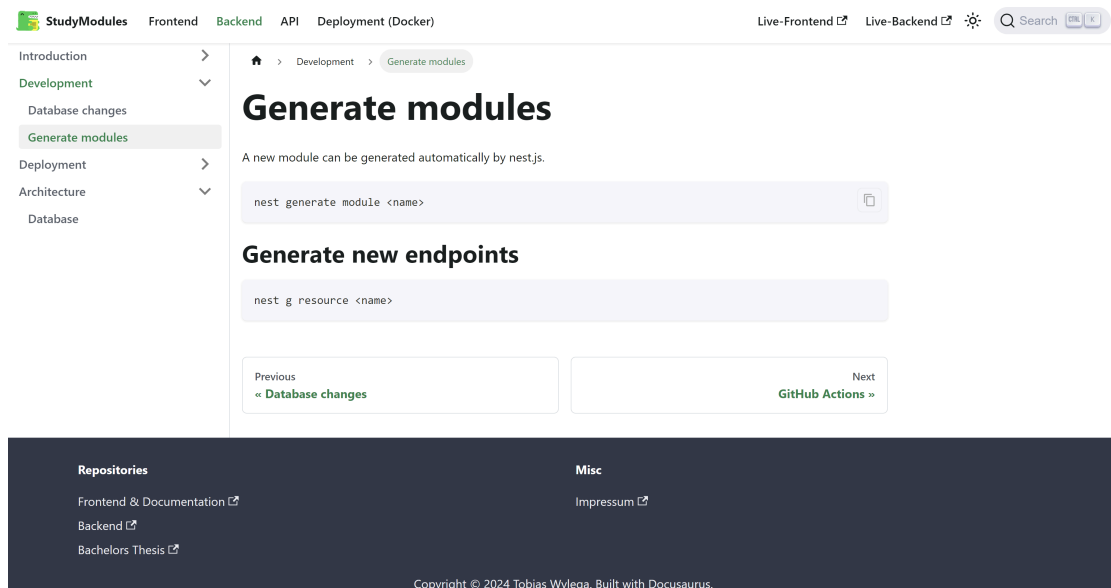


Abbildung 27: Docusaurus-Website

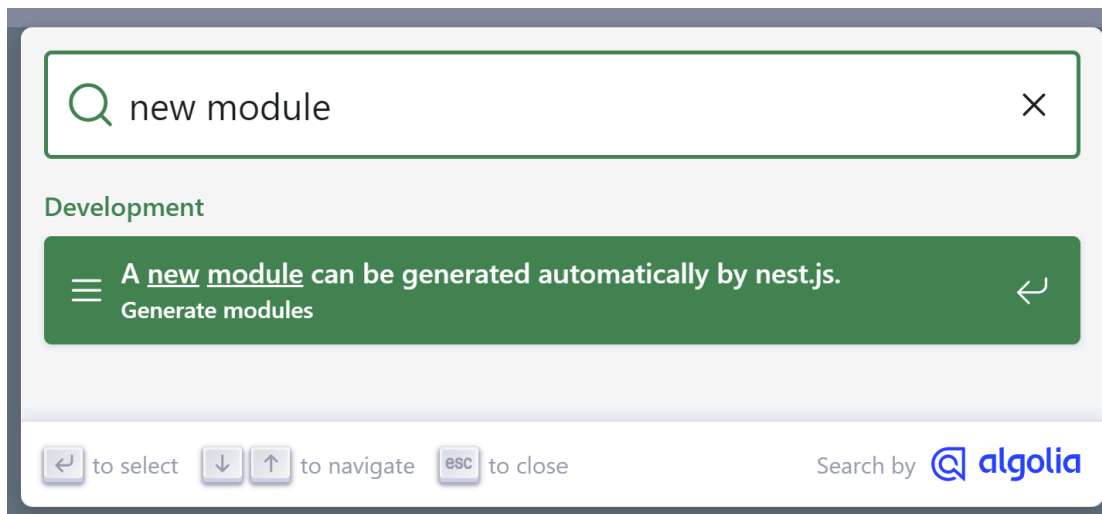


Abbildung 28: Suchfunktion der Dokumentation

### 4.4. Podman

Im letzten Schritt wurden alle Anwendungen für die Verwendung Podman [66] vorbereitet. Podman erlaubt es, Anwendungen in Containern zu kapseln. Podman erinnert sehr stark an Docker [67] und nutzt ähnliche Befehle. Dieses System wurde mit Podman getestet, weil es in Zukunft auch mit Podman gehostet werden soll.

Um das neue System mit Podman bereitzustellen, sind einige Vorbereitungen nötig. Zunächst muss überlegt werden, welche Container benötigt werden. Für jede Anwendung soll ein eigener Container genutzt werden, um die Zuständigkeiten klar zu trennen. Dies bringt den Vorteil, dass einzelne Komponenten einfacher austauschbar sind, da sie über eine definierte Schnittstelle miteinander kommunizieren. Es wird für das Backend, das Frontend, die LaTeX-API, das LaTeX-Poll-Skript und die Dokumentation jeweils ein Container benötigt.

Im ersten Schritt musste nun in jedem Verzeichnis der soeben genannten Anwendungen ein Containerfile erstellt werden, welches beschreibt, wie sich der Container verhalten soll. Aus dem Containerfile wird später ein Image erstellt, mit dem der Container generiert werden kann. Das Image ist also die Vorlage für den Container. Für das Backend und die LaTeX-API gab es bereits ein Containerfile. Für die Dokumentation wurde das Containerfile aus der Docusaurus-Dokumentation [68] übernommen. Für das LaTeX-Poll-Skript und das neue Frontend wurden jeweils ein neues Containerfile erstellt.

Im Containerfile für das LaTeX-Poll-Skript (Code 28) ist ein einfaches Beispiel für ein Containerfile zu sehen. In Code 28, Zeile 1 wird die Basis des Images festgelegt. [69] In diesem Fall wird für das Skript Python benötigt, daher wird ein Python-Image als Basis verwendet. Es wurde eine Alpine-Version des Python-Images ausgewählt. Die Alpine-Version zeichnet sich dadurch aus, dass deren Größe minimal gehalten wurde, indem nur die benötigten Werkzeuge (in diesem Fall Python) installiert sind. [70] Des Weiteren wird in Code 28, Zeile 6 ein benötigtes Paket installiert, damit HTTP-Requests möglich sind. Abschließend wird dann noch ein Crontab installiert, welcher das Python-Skript in einem bestimmten Intervall regelmäßig ausführt.

```
1 FROM python:3.11.9-alpine3.20 Dockerfile
2
3 COPY . /app
4 WORKDIR /app
5
6 RUN pip install requests
7
8 RUN crontab /app/crontab
9
10 # log to stdout for easier debugging
11 CMD ["crond", "-f", "-l", "2"]
```

Code 28: Containerfile für das Latex-Poll-Skript

Das Containerfile des Frontends ist komplexer. Es besteht aus zwei Stages. Die erste Stage basiert auf einem node-alpine-Image (Code 29, Zeile 1) und ist für das Kompilieren des Quellcodes zuständig. Die zweite Stage (Code 29, Zeile 15) ist als Webserver für die Bereitstellung der Website zuständig und benutzt das Image nginx:alpine als Basis, beinhaltet also einen Webserver.

Für das Builden in der ersten Stage wurde darauf geachtet, dass nicht direkt alle Dateien kopiert werden (Code 29, Zeile 11). Stattdessen werden zunächst nur die package.json und die package-lock.json kopiert (Code 29, Zeile 7) und ein npm ci ausgeführt, um die Pakete zu installieren. Dies hat den Vorteil, dass das Ergebnis gecacht werden kann. Podman arbeitet mit Layern, was in diesem Fall bedeutet, dass das Ergebnis nach dem Installieren der Packages in einem Layer abgespeichert wird. Dieses Layer kann nun beim erneuten Builden des Podman-Images aus dem Cache geladen werden (sofern sich die package.json nicht verändert hat). Wenn in Code 29, Zeile 7 alle Dateien direkt in das Image geladen würden, müssten bei jedem Build des Images erneut die Abhängigkeiten installiert werden, was zu einer verlängerten Laufzeit des Buildvorgangs führen würde. [71]



Für die anderen Container ist mehr Konfigurationsaufwand nötig. Hier muss zum Beispiel noch das Port-Mapping gesetzt werden. Hierbei wird ein Port, der innerhalb des Containers existiert, auf einen Port, der außerhalb des Containers existiert, gemappt. Im Fall vom Backend wird beispielsweise der Port 3000 aus dem Container nach außen als Port 3000 freigegeben. Somit ist das Backend auf Port 3000 erreichbar. Bei der LaTeX-API wurde der Port 8080 aus dem Container nach außen als Port 2345 freigegeben, da Port 8080 bereits für die Dokumentation belegt war. Auf dem Server, auf dem die Podman-Container laufen, ist die LaTeX-API nun also unter dem Port 2345 zu erreichen. Allerdings gibt es einen Fallstrick: Das `latex-poll-script` soll auf die LaTeX-API zugreifen. Dies funktioniert allerdings innerhalb des Containers `latex-poll-script` nicht über den Port 2345. Stattdessen wird hier als URL der Name des Containers verwendet, auf den zugegriffen wurde, und als Port der Port innerhalb des Containers. Um also vom Poll-Skript auf die API zuzugreifen, muss diese URL aufgerufen werden: `latex-api:8080`.

Das Backend benötigt außerdem eine Möglichkeit, Daten langfristig aufzubewahren. Bei einem Update des Images muss der Container des Backends neu erstellt werden, sodass darin enthaltene Daten verloren gehen. Um dieses Problem zu umgehen, können Volumes genutzt werden. [72] In der Compose-Datei ist in Code 30, Zeile 19 zu sehen, wie eine Volume definiert wird. Hierzu wird einfach ein Ordner vom Hostsystem auf einen Ordner innerhalb des Containers gemappt. Beide Ordner werden automatisch synchronisiert. Wenn ein Container zerstört wird, bleibt die Volume bestehen und kann für den nächsten Container weiterverwendet werden. Als zusätzlichen Vorteil kann nun auch vom Hostsystem auf die Dateien in der Volume zugegriffen werden. In der Volume wird beispielsweise der Inhalt der Datenbank aufbewahrt.

Nachdem die Images nun erstellt sind und die Compose-Datei die Struktur der Container definiert, kann `podman compose up -d` ausgeführt werden, um die Container zu erstellen und zu starten. In der Desktopanwendung von Podman ist dies nachzuvollziehen (Abbildung 29). In Abbildung 30 ist eine Übersicht der verschiedenen Container zu sehen. Dort sind die verwendeten Ports aufgelistet und es kann erkannt werden, in welche Richtung kommuniziert wird.

```
1  name: studymodules_project yml
2
3  services:
4    frontend:
5      image: localhost/studymodules-frontend
6      ...
7      ...
12   backend:
13     image: localhost/studybase-backend
14     ports:
15       - "3000:3000"
16     environment:
17       - NODE_ENV=production
18     volumes:
19       - C:\Users\tobi\studybase\docker-volume:/app/docker-volume
20
21   documentation:
22     image: localhost/studymodules-documentation
23     ports:
24       - "8080:80"
25       - "443:443"
26     ...
27     ...
33   latex-api:
34     image: localhost/latex-api:latest
35     ports:
36       - "2345:8080"
37     build: .
38     command: make start
39     environment:
40       # SENTRY_DSN:
41       CACHE_HOST: cache
42     depends_on:
43       - backend
44
45   latex-poll-script:
46     image: localhost/studymodules-latex:latest
47
```

```
48 volumes:
49   caddy_data:
50   caddy_config:
```

Code 30: compose-Datei

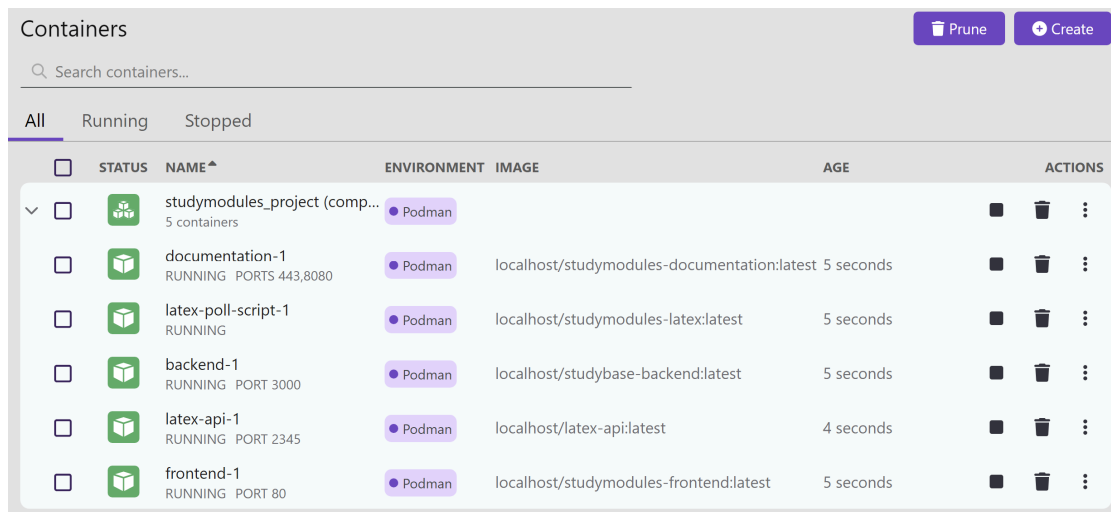


Abbildung 29: Podman Desktop

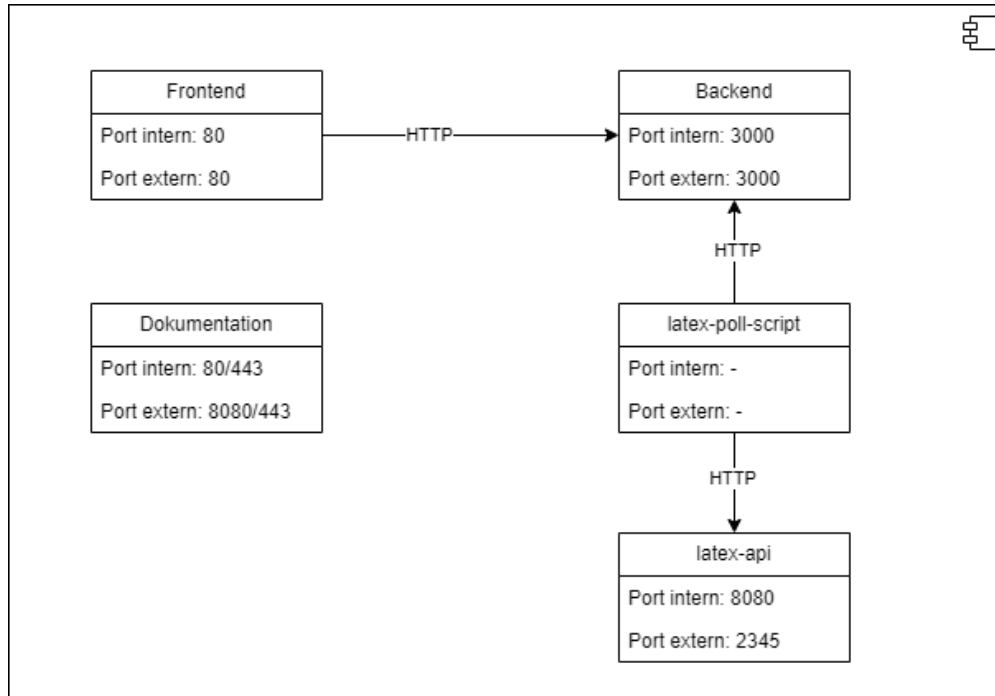


Abbildung 30: Diagramm der eingesetzten Dockercontainer

---

## **4.5. Zwischenfazit**

Nachdem Frontend und Backend implementiert wurden und eine Dokumentation erstellt wurde, besteht eine erste lauffähige Version des Systems. Diese Version kann im folgenden Kapitel 5 evaluiert werden, um herauszufinden, ob alle Anforderungen umgesetzt wurden.

# 5. Review

In diesem Kapitel wird das erstellte System überprüft. Es soll herausgefunden werden, ob das neue System einsetzbar ist. Hierzu wird ein Interview geführt und die in Kapitel 2 aufgestellten Anforderungen überprüft.

## 5.1. Interview mit modulverantwortlicher Person

Im Laufe der Implementierungsphase wurde ein Prototyp einer modulverantwortlichen Person vorgestellt, um einschätzen zu können, ob das entstehende System eine Erleichterung des bisherigen Arbeitsprozesses darstellen könnte. Aus dem Interview ergaben sich kleinere Anpassungen an den Anforderungen, jedoch konnten keine groben Fehler am Gesamtsystem festgestellt werden. Ergebnis des Interviews war, dass die Anwendung benutzbar wirkt und den Arbeitsprozess vermutlich verbessern wird.

## 5.2. Abweichungen zum Prototypen

Um sicherzustellen, dass keine wichtigen Details aus den Entwürfen übersehen wurden, wurde das neue System in diesem Abschnitt mit den Entwürfen aus Abschnitt 3.2 verglichen.

Das in Abbildung 5 gezeigte Menü enthält in der tatsächlichen Implementierung andere Einträge. Es gibt beispielsweise in der aktuellen Version des Systems noch keine Möglichkeit, einen neuen Studiengang oder eine neue Abteilung anzulegen. Dies muss aktuell noch per Datenbankzugriff erledigt werden. Da das System im ersten Schritt jedoch nur von der Abteilung Informatik genutzt werden soll, ist dies zunächst in Ordnung. Eine Implementierung entsprechender Funktionen sollte zudem nicht aufwändig sein, da es bereits ähnliche Implementierungen für das Erstellen von Modulen und Teilmodulen gibt. Des Weiteren entfällt die geplante Benutzerverwaltung (Abbildung 16), da in Zukunft eventuell direkt das LDAP der Hochschule verwendet werden könnte. Stattdessen gibt es im Drawer jetzt die Möglichkeit, PDF-Kompilierungsanträge und Teilmodule zu verwalten.

Die Filter in Abbildung 10 wurden direkt in die Tabellenüberschrift integriert (Abbildung 31). Außerdem wurden weitere Spalten eingeführt, um Use Case 2 abbilden zu können.

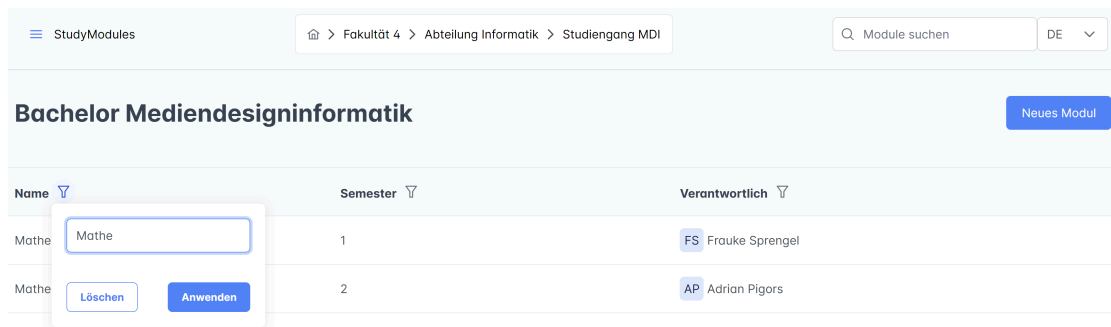


Abbildung 31: Filtermöglichkeit

Die in Abbildung 14 und Abbildung 15 zu sehenden Steuerelemente zum Hinzufügen von Übersetzungen wurden zugunsten einer besseren Usability ersetzt (siehe Unterabschnitt 4.2.8).

Alle nicht genannten Abbildungen aus Abschnitt 3.2 ähneln den tatsächlichen Implementierungen.

Bei dem Vergleich sind keine unerklärten Abweichungen gefunden worden. Alle vorgenommenen Änderungen verbessern entweder das System oder sind nicht für die Erfüllung der wesentlichen Use Cases erforderlich, sodass davon ausgegangen wird, dass alle wichtigen Anforderungen an das Design erfüllt wurden.

### 5.3. Vergleich der PDFs

Da die Generierung eines Modulhandbuchkataloges im PDF-Format ein zentraler Bestandteil dieser Arbeit war, sollte das Ergebnis genauer geprüft werden. Hierzu wurden aktuelle Modulhandbücher von der Website der Hochschule mit den neu generierten Handbüchern aus dem System verglichen. Bei der ersten Überprüfung (siehe Abbildung 32 und Abbildung 33 [73]) sind dabei noch einige Aufgaben aufgefallen, die im Anschluss behoben wurden. So fehlten beispielsweise bei den Semesterwochenstunden ein Komma und das Suffix „SWS“. Bei den längeren Texten (z. B. „Inhalt“) fällt außerdem auf, dass die neuen Texte im Blocksatz dargestellt sind, während die ursprünglichen Texte lediglich linksbündig ausgerichtet sind.

**Teilmodul BIN-102-01 Programmieren 1**

Untertitel	(BIN-PR1)
Verantwortliche(r)	Garmann, Robert, Prof. Dr.
Sprache	Deutsch
Zuordnung zu Curricula	BIN
Veranstaltungsart, SWS	Vorlesung mit Übung, 4 SWS
Credits	6
Präsenzstunden / Selbststudium	68 h / 112 h
Empfehlungen zum Selbststudium	siehe Literatur
Empfohlene Voraussetzungen	keine
Studien-/ Prüfungsleistungen	Prüfung (Klausur oder mündliche Prüfung) und experimentelle Arbeit
Gruppengröße	70

**Angestrebte Lernergebnisse**

Algorithmische Kompetenz: eine konkrete Problemstellung analysieren und algorithmisch lösen können, grundlegende Algorithmen und Datenstrukturen zur Lösung von Problemen einsetzen  
Realisierungs-Kompetenz: Beherrschung des imperativen Programmierparadigmas unter Nutzung von Objektbibliotheken, Erstellen und Testen von Programmen unter Einsatz entsprechender Werkzeuge

**Inhalt**

Einführung in die Grundlagen der objektorientierten Programmierung anhand der Programmiersprache Java, deren Sprachkonstrukte mit einigen wichtigen Bibliotheken vorgestellt werden. Viele praktische Beispielaufgaben vertiefen den Stoff.  
Behandelt werden u.a. Grundlagen der Programmierung: Problem, Algorithmus, Programm, Grundlagen der objektorientierten Programmierung: Pakete, Klassen, Objekte, Einfache und strukturierte Datentypen, Kontrollstrukturen, Ein-/Ausgabe, Behandlung von Ausnahmen, Abstraktion, Rekursion

Abbildung 32: Ursprüngliches PDF

**Unterm modul BIN-102-01 Programmieren 1**

<b>Untertitel</b>	(BIN-PR1)
<b>Verantwortliche(r)</b>	Prof. Dr. Robert Garmann
<b>Sprache</b>	Deutsch
<b>Zuordnung zu Curricula</b>	BIN, MDI
<b>Veranstaltungsart, SWS</b>	Vorlesung mit Übung 4
<b>Credits</b>	6
<b>Präsenzstunden / Selbststudium</b>	68 h / 112 h
<b>Studiensemester</b>	1
<b>Empfehlungen zum Selbststudium</b>	siehe Literatur
<b>Empfohlene Voraussetzungen</b>	keine
<b>Studien- / Prüfungsleistungen</b>	Prüfung (Klausur oder mündliche Prüfung) und experimentelle Arbeit
<b>Gruppengröße</b>	70

**Angestrebte Lernergebnisse**

Algorithmische Kompetenz: eine konkrete Problemstellung analysieren und algorithmisch lösen können, grundlegende Algorithmen und Datenstrukturen zur Lösung von Problemen einsetzen  
 Realisierungs-Kompetenz: Beherrschung des imperativen Programmierparadigmas unter Nutzung von Objektbibliotheken, Erstellen und Testen von Programmen unter Einsatz entsprechender Werkzeuge

**Inhalt**

Einführung in die Grundlagen der objektorientierten Programmierung anhand der Programmiersprache Java, deren Sprachkonstrukte mit einigen wichtigen Bibliotheken vorgestellt werden. Viele praktische Beispielaufgaben vertiefen den Stoff. Behandelt werden u.a. Grundlagen der Programmierung – Problem, Algorithmus, Programm, Grundlagen der objektorientierten Programmierung – Pakete, Klassen, Objekte, Einfache und strukturierte Datentypen, Kontrollstrukturen, Ein-/Ausgabe, Behandlung von Ausnahmen, Abstraktion, Rekursion

Abbildung 33: Neues PDF

Nachdem die kleineren Anpassungen vorgenommen und mit einer erneuten Überprüfung verifiziert wurden, ähnelt das neue PDF nun dem ursprünglichen PDF. Die Nutzung des neuen PDFs sollte dementsprechend möglich sein.

## 5.4. Überprüfung, ob Anforderungen erfüllt sind

Im Folgenden wird überprüft, welche Anforderungen erfüllt sind und welche Anforderungen in Zukunft noch umgesetzt werden müssen. Dies hilft für die spätere Einschätzung, ob das System bereits eingeführt werden könnte, oder ob es noch kritische offene Aufgaben gibt. Die Anforderungen enthalten gegebenenfalls einen Verweis auf Abbildungen oder Passagen im vorangegangenen Text, um die aufgestellten Behauptungen zu beweisen.

Anf.	Umsetzung	Bewertung
<i>Use Case 1</i>		
F1	Durch die Nutzung von Routen kann ein Studiengang mithilfe eines Links aufgerufen werden.	Erfüllt Code 19
F2	In der Übersicht aller Studiengänge der Abteilung kann ein PDF für jeden Studiengang aufgerufen werden.	Erfüllt Abbildung 18
<i>Use Case 2</i>		
F3	Es können alle Module eines Studienganges angezeigt werden.	Erfüllt Abbildung 19
F4	In allen Spalten steht eine Filterfunktion bereit.	Erfüllt Abbildung 31
F5	Die Suchfunktion wurde erfolgreich implementiert.	Erfüllt Unterabschnitt 4.2.6
F6	Es gibt für jedes Modul eine Detailansicht mit allen verfügbaren Informationen.	Erfüllt Abbildung 20
<i>Use Case 3</i>		
F7	Im seitlichen Menü gibt es einen Login-Button.	Erfüllt
F8	Im seitlichen Menü gibt es einen Logout-Button.	Erfüllt
F9	Wurde nicht umgesetzt, da Accounts in der Einführungsphase erst direkt in der Datenbank verwaltet werden. Eventuell erfolgt später eine Anbindung an die SSO-Accounts der Hochschule.	Nicht erfüllt
F10	Wurde nicht umgesetzt (siehe Umsetzung F9)	Nicht erfüllt
F11	Personen, die für ein Modul verantwortlich sind, sowie die studiengangsverantwortliche Person können Module bearbeiten.	Erfüllt Abbildung 24
F12	Es wurden verschiedene Plausibilitätschecks eingebaut. Fehlerhafte Felder werden rot markiert. Der User erhält weitere Informationen über einen Tooltip.	Erfüllt Abbildung 26

Anf.	Umsetzung	Bewertung
<i>Use Case 4</i>		
F13	Module können bearbeitet werden.	Erfüllt Abbildung 24
F14	Wurde aus zeitlichen Gründen nicht priorisiert.	Nicht erfüllt
F15	Studiengänge können verwaltet werden	Erfüllt Abbildung 21
F16	Studiengänge können dupliziert werden. Dabei werden darin enthaltene Module und Teilmodule ebenfalls dupliziert. Ansprechpartner werden nicht dupliziert, sondern auf die bestehenden Einträge verwiesen.	Erfüllt Abbildung 21
F17	Studiengänge können im Menü in der Abteilungsübersicht ausgeblendet werden.	Erfüllt Abbildung 21
F18	Wenn ein User angemeldet ist, werden ausgeblendete Studiengänge angezeigt. Die für nicht angemeldete User ausgeblendeten Studiengänge erhalten eine Markierung, die darauf hinweist.	Erfüllt Abbildung 34
F19	Wurde nicht umgesetzt (siehe Umsetzung F9)	Nicht erfüllt
F20	Teilmodule können verwaltet werden.	Erfüllt
F21	Voraussetzungen haben keine eigenständige Verwaltungsmaske, sondern sind ein Teil der Modulbearbeitungsmaske und können dort bearbeitet werden.	Erfüllt
<i>Use Case 5</i>		
F22	Die Änderungen an einem Modul werden automatisch protokolliert. Dabei ist der Benutzer angegeben, sowie alle modifizierten Felder.	Erfüllt Abbildung 35 Unterabschnitt 4.1.2
F23	Wurde aus zeitlichen Gründen nicht priorisiert.	Nicht erfüllt
F24	Wurde aus zeitlichen Gründen nicht priorisiert.	Nicht erfüllt
F25	Wurde aus zeitlichen Gründen nicht priorisiert.	Nicht erfüllt
<i>Use Case 6</i>		
F26	Wurde aus zeitlichen Gründen nicht priorisiert.	Nicht erfüllt

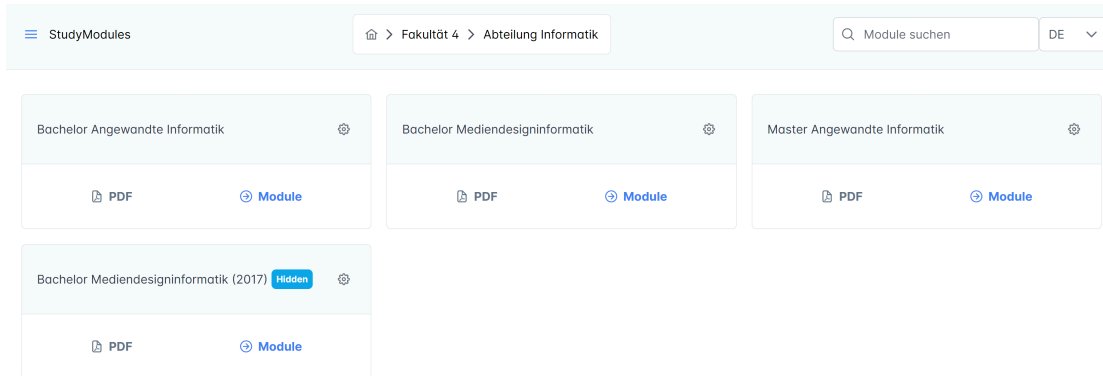


Abbildung 34: Ausgeblendeter Studiengang

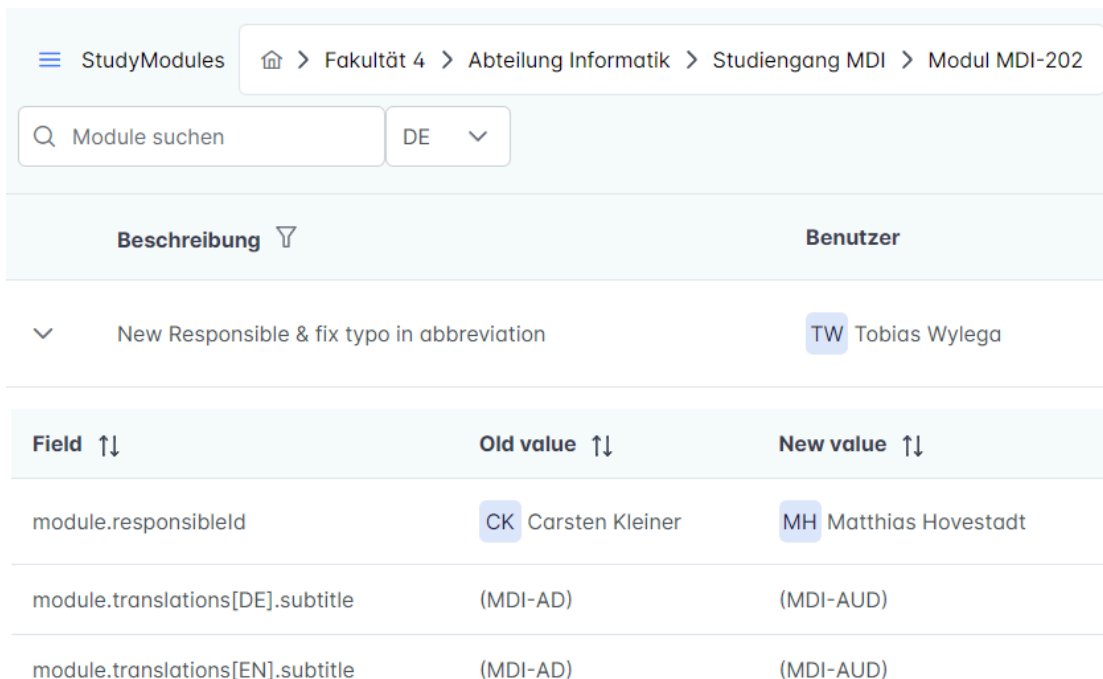


Abbildung 35: Änderungshistorie

Anf.	Umsetzung	Bewertung
<i>Änderbarkeit</i>		
N1	Frontend und Backend nutzen wiederverwendbare Services (z. B. UserService zum Abrufen der Daten aller User). Außerdem gibt es im Frontend wiederverwendbare Komponenten (z. B. ResponsibleAvatarComponent)	Erfüllt
N2	Durch Erfüllung der Anforderungen N1, N3, N4, N5 und N6 sollte der vorliegende Code gut testbar sein, sodass die Anforderungen N8 und N9 umgesetzt werden können.	Erfüllt
N3	Bei der Entwicklung wurde darauf geachtet, dass es für die verschiedenen Arten von Daten jeweils eigene Services gibt. Es gibt nicht eine große Klasse „DatabaseAccess“, sondern einen ModuleService, einen SubModuleService, einen RequirementService und so weiter.	Erfüllt
N4	Aus der Trennung von Frontend und Backend ergibt sich eine lose Kopplung, da das Frontend keine Abhängigkeit zur Datenbank hat, weil nur über die REST-Schnittstelle kommuniziert wird. Außerdem kommen die meisten Komponenten im Quellcode mit einer geringen Anzahl an Abhängigkeiten aus. Als Beispiel ist hier das ModuleGridComponent zu nennen, welches als zentrales Element der Anwendung die Module in einer Tabelle auf der Oberfläche anzeigt. Diese Komponente hat Abhängigkeiten zum ModuleService (um alle Module abzurufen), zum Router, der Activated Route und dem CourseService (um den ausgewählten Studiengang aus der URL auslesen und dessen Namen anzeigen zu können), zum AuthService (um zu ermitteln, ob der User eingeloggt ist) und zum LanguageService (um die Website in der gewünschten Sprache zu zeigen). Diese Anzahl an Abhängigkeiten ist gerechtfertigt, weil die Komponente eine zentrale Rolle in der Anwendung spielt und daher mit verschiedenen Bereichen der Anwendung interagieren muss.	Erfüllt
N5	Mithilfe eines externen Tools (Qodana [74]) wurde die zyklomatische Komplexität der einzelnen Methoden betrachtet. Hierbei wurden keine Methoden mit einer höheren Komplexität als 10 gefunden.	Erfüllt

<b>Anf.</b>	<b>Umsetzung</b>	<b>Bewertung</b>
N6	In Frontend und Backend werden Abhängigkeiten mithilfe von Dependency Injection eingesetzt.	Erfüllt Unterabschnitt 4.1.2 Unterabschnitt 4.2.3
N8	Bisher nicht umgesetzt, aber durch N2 vorbereitet.	Vorbereitet
N9	Bisher nicht umgesetzt, aber durch N2 vorbereitet.	Vorbereitet
<i>Benutzbarkeit</i>		
N10	Der aktuelle Pfad wird in der Anwendung angezeigt. Durch Anklicken eines Elementes kann zurückgesprungen werden (beispielsweise von der Detailansicht eines Moduls zurück zur Auflistung aller Module)	Erfüllt Abbildung 20
N11	Das Löschen von Datensätzen muss vom User bestätigt werden. Hierzu erscheint ein Popup mit den Buttons „Ja“ und „Nein“.	Erfüllt Abbildung 36
N12	Diese Anforderung wurde nicht umgesetzt. Das Löschen eines Datensatzes könnte in Zukunft angepasst werden, sodass nur eine Eigenschaft „deleted“ auf True gesetzt wird. Beim Laden von Daten werden nur Datensätze geladen, die „deleted=false“ sind. Die Oberfläche könnte angepasst werden, sodass angemeldete User auch gelöschte Elemente ansehen können.	Konzept liegt vor
N13	Das Generieren eines PDFs dauert recht lange. Hier wird ein Statusindikator eingesetzt, der dem User anzeigt, in welchem Status sich der Kompilierungsauftrag befindet. Abgesehen davon gibt es keine Stellen in der Anwendung, die eine erhöhte Ladezeit haben. Ein Ladebalken wurde daher nicht eingebaut, das Ziel einer guten Benutzbarkeit aber trotzdem erreicht.	Erfüllt Abbildung 23
N14	Mögliche Fehler werden abgefangen und mithilfe einer verständlichen Fehlermeldung an den User übermittelt.	Erfüllt Abbildung 37
N15	Bisher nicht umgesetzt, aber durch N14 vorbereitet.	Vorbereitet
N16	Die Übersicht der Studiengänge, aller Module und die Moduldetails wurden für mobile Endgeräte optimiert.	Teilweise erfüllt Unterabschnitt 4.2.2
N17	Wurde aus zeitlichen Gründen nicht priorisiert.	Nicht erfüllt
N18	Ohne eine Studie oder Ähnliches ist es schwer zu beweisen, dass das System selbsterklärend ist. In der Implementierung wurde darauf geachtet, möglichst selbsterklärende Beschriftungen, Icons und Steuerelemente	Vermutlich erfüllt Abbildung 24

Anf.	Umsetzung	Bewertung
	zu verwenden. In den Bearbeitungsmasken gibt es eine Live-Vorschau, sodass ersichtlich ist, welches Eingabefeld was verändert. Außerdem wurden an verschiedenen Stellen Tooltips, Dialoge und Einblendungen verwendet, um möglicherweise unklare Details genauer zu erklären.	
<i>Effizienz</i>		
N19	Alle Seiten im Frontend laden innerhalb einer Sekunde. Die geladenen Datenmengen werden reduziert, indem beim Laden der Modulübersicht zum Beispiel nicht direkt alle Informationen eines Moduls geladen werden. Diese werden erst geladen, sobald die Detailansicht geöffnet wird.	Erfüllt Abschnitt 3.3
N20	Während der Implementierung des Systems konnte kein Zustand ermittelt werden, zu dem das Backend nach einem Fehler nicht innerhalb einer Minute neu startet.	Erfüllt
N21	Das Frontend wird mithilfe einer GitHub-Action bei jedem Push auf den Testserver deployt. Das Backend wird mithilfe eines Deploy-Skripts und einem Cronjob automatisch deployt. Diese Methodik ist auch für den Livebetrieb möglich.	Vorbereitet
N22	Es wurde versucht, die Arbeitsabläufe möglichst einfach zu gestalten. Ohne größeren Aufwand ist es nicht nachprüfbar, ob diese Anforderung vollständig erledigt ist.	Vermutlich erfüllt
<i>Funktionalität</i>		
N23	Die Anwendung und die PDFs die generiert werden stehen in Englisch und Deutsch bereit.	Erfüllt Unterabschnitt 4.2.3
N24	Der Code ist so vorbereitet, dass ohne großen Aufwand weitere Sprachen hinzugefügt werden können. Aktuell liegen die Modulhandbücher nur in Englisch und Deutsch vor, weshalb die neue Anwendung auch nur in Deutsch und Englisch entwickelt wurde.	Vorbereitet
N25	In den Bearbeitungsmasken werden Eingabefelder verwendet, welche den User bei der Eingabe unterstützen. Wenn beispielsweise eine Zahl erwartet wird, können keine Buchstaben eingegeben werden. Außerdem werden, wenn möglich, Dropdowns statt Textfeldern genutzt.	Erfüllt Abbildung 24
N26	Das neue PDF ähnelt dem bisherigen PDF.	Erfüllt Abschnitt 5.3

<b>Anf.</b>	<b>Umsetzung</b>	<b>Bewertung</b>
N27	Nur autorisierte Benutzer können datenverändernde Endpunkte verwenden.	Erfüllt
<i>Übertragbarkeit</i>		
N28	Es gibt eine Dokumentation, welche unter anderem erklärt, wie das System einzurichten ist.	Erfüllt Abschnitt 4.3
N29	Das neue System kann mithilfe von Podman-Containern deployt werden.	Erfüllt Abschnitt 4.4
N30	Durch die Verwendung von Prisma ist der Zugriff auf die Datenbank abstrahiert, daher kann die tatsächliche Datenbank ohne größeren Aufwand ausgetauscht werden. Der LaTeX-Kompilierungs-Server wird über einen einzelnen REST-Endpunkt angesprochen und ist dadurch nur sehr lose gekoppelt. Das Frontend arbeitet hingegen mit mehreren REST-Endpunkten des Backends und ist daher zwar austauschbar, jedoch wäre das verglichen mit den zuvor genannten Komponenten aufwändig.	Erfüllt
<i>Zuverlässigkeit</i>		
N31	Fehler, die während des Testens aufgefallen sind, haben die Systeme nicht zum Absturz gebracht.	Vermutlich erfüllt
N32	Beim Erledigen der Use Cases sind keine Fehler aufgefallen.	Erfüllt
N33	Wurde aus zeitlichen Gründen nicht priorisiert.	Nicht erfüllt
<i>Technische Anforderungen</i>		
N34	Das Frontend ist eine neue Anwendung.	Erfüllt
N35	Das Frontend nutzt Angular.	Erfüllt
N36	Das bestehende Backend wurde erweitert.	Erfüllt
N37	Das Backend nutzt (weiterhin) NestJS und Prisma.	Erfüllt
N38	Die bestehende Datenbank wurde erweitert.	Erfüllt



Abbildung 36: Rückfrage beim Löschen

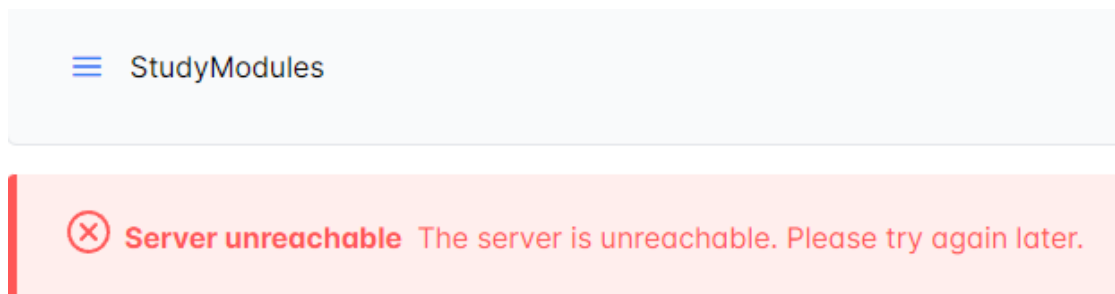


Abbildung 37: Fehlermeldung vom error.service.ts

## 5.5. Zwischenfazit

Nachdem die aufgestellten Anforderungen überprüft wurden, kann nun eine Aussage zum neuen System getroffen werden.

Von den 64 Anforderungen wurden 15 Anforderungen nicht erfüllt. Unter den nicht erfüllten Anforderungen befinden sich keine Anforderungen mit dem Schlüsselwort „muss“. Es wurden also alle Anforderungen, die für das System zwingend erforderlich sind, erfüllt. Aus diesem Grund ist davon auszugehen, dass das System für die in Abschnitt 2.5 erstellten Use Cases nutzbar ist (mit Ausnahme des nur teilweise erfüllten Use Case 5 und des ausgelassenen Use Case 6).



# 6. Fazit

Das in Kapitel 1 aufgestellte Ziel, eine Webanwendung zu erstellen, welche die Bearbeitung und Veröffentlichung von Modulhandbüchern erleichtert, wurde durch eine klare Struktur angegangen. Nachdem in Kapitel 2 die Zielgruppen und Anforderungen ermittelt wurden, konnte in Kapitel 3 das System geplant werden. In Kapitel 4 wurden die Anforderungen umgesetzt, was in Kapitel 5 bestätigt wurde.

Das Ergebnis dieser Bachelorarbeit ist ein verbesserter Prozess zur Bearbeitung des Modulkatalogs. Die hierzu aufgestellten Anforderungen wurden größtenteils erfüllt. Unter den nicht erfüllten Anforderungen befinden sich keine Anforderungen, die die produktive Nutzung der Anwendung verhindern. Zwar wäre es gut, wenn die ausstehenden Arbeiten in Zukunft umgesetzt werden, jedoch kann mit dem bestehenden System bereits der aktuell bestehende Arbeitsablauf ersetzt werden. Das System ermöglicht das Anzeigen von Modulkatalogen als PDF und das Anzeigen von Moduldetails in einer modernen Oberfläche. Außerdem können die Informationen der Module im System bearbeitet werden, wobei das System mithilfe von Dropdowns und Plausibilitätschecks unterstützt. Das resultierende PDF wird automatisch generiert und den Interessierten zur Verfügung gestellt. Außerdem sind wesentliche Informationen zur Einrichtung und Weiterentwicklung des Systems in einer Dokumentation erfasst.

Durch die modernen Oberflächen sollte es Studierenden und studieninteressierten Personen leichtfallen, an die gewünschten Informationen zu bestimmten Modulen oder einem Studiengang zu gelangen. Für die dozierenden Personen an der Hochschule Hannover sollte die Aktualisierung der Modultexte dank der intuitiven Eingabefelder sowie der unterstützenden Plausibilitätsprüfung angenehmer sein. Die Generierung eines PDFs läuft jetzt weitestgehend automatisch, sodass der Studiengangsdekan für diese Aufgabe nun weniger Zeit benötigen sollte.

Alles in allem wurde durch die Erfüllung der wesentlichen Anforderungen und durch die Bereitstellung des neuen Systems das aufgestellte Ziel erreicht. Es gibt eine Webanwendung, die die Verwaltung und Erstellung von Modulhandbüchern ermöglicht.

---

---

# 7. Ausblick

Obwohl das neue System bereits funktionsfähig ist, gibt es dennoch Potenzial für weitere Optimierungen. Sowohl die erstellte Dokumentation als auch der Quellcode könnten in bestimmten Bereichen weiter verbessert werden. Einzelne Abschnitte im Code könnten noch redundanten Code enthalten und die Dokumentation enthält nicht alle in dieser Arbeit genannten Einzelheiten des Systems. Sollten bei der Weiterentwicklung verbesserungswürdige Stellen in Dokumentation oder Quellcode auffallen, können diese dokumentiert und verbessert werden. Die aktuelle Struktur des Systems sollte Erweiterungen und Anpassungen ermöglichen.

Der nächste Schritt könnte die Einführung des neuen Systems für die Abteilung Informatik sein. Hierzu könnte das System auf Hochschulservern eingerichtet werden. Im Anschluss müssten die Modulverantwortlichen zunächst einmal das generierte PDF, beziehungsweise die im System hinterlegten Daten, auf Vollständigkeit prüfen. Ein Link zu verschiedenen Seiten des Systems könnte dann auf der Website der Hochschule erstellt werden. Es wäre beispielsweise denkbar, auf der Unterseite der Abteilung Informatik auf die Studiengangübersicht des neuen Systems (Abbildung 21) zu verlinken.

Für die nähere Zukunft ist es außerdem denkbar, dass abgesehen von den bereits erfassten nicht erfüllten Anforderungen weitere Funktionen geplant werden. So könnte die moderne Anzeige von einzelnen Modulen um verschiedene Informationen erweitert werden. Es könnte beispielsweise eine Anbindung an den Stundenplan geben, sodass auf der Modulseite auch angezeigt wird, zu welchen Zeiten und an welchem Ort die Lehrveranstaltungen stattfinden. Auch wäre die Anbindung weiterer Abteilungen und Fakultäten denkbar. Dies ist durch die Datenstruktur bereits vorbereitet, jedoch könnte es erforderlich sein, die PDF-Dokumente optisch anders darzustellen. Auch wäre eine Anbindung an das HISinOne denkbar, wie bereits in Abschnitt 1.3 beschrieben. Zusätzlich zu den Modulhandbüchern und dem Anhang der Prüfungsordnung könnte ebenso die in Abschnitt 1.2 genannte Abbildung des Curriculums (Abbildung 1) generiert und im neuen System angezeigt werden.

In der Bearbeitungsansicht könnte statt der derzeitigen HTML-Vorschau das tatsächliche PDF angezeigt werden. Dies wurde durch die Umstellung auf TypeScript bereits vorbereitet und könnte eine sinnvolle Änderung sein, um zum einen eine genauere Vorschau zu ermöglichen und zum anderen die Code-Qualität zu verbessern, indem Abhängigkeiten verringert werden.

Das neu entworfene System könnte zudem Vorlage für weitere Entwicklungen sein. Die genutzten Technologien sowie die entworfene Struktur könnten für eine Vielzahl von Projekten interessant sein.

---

---

# Anhang

## Quellcode

Der in dieser Arbeit entstandene Quellcode ist an den folgenden Stellen zu finden.

### Frontend & Entwicklerdokumentation

<https://gitlab.gwdg.de/lernanwendungen/studymodules>

Branch: main

Commit-Hash: 8217a61d2fd15ef2b38cb5a5b47fb2351902017b

### Backend & LaTeX-Skript

<https://gitlab.gwdg.de/lernanwendungen/studybase>

Branch: studymodules\_tobi

Commit-Hash: 9cdd4a94fdad896e86df201a1f8c212c0587231b

---

## Interview-Fragen

**Häufigkeit von Änderungen** *Falls diese Informationen nicht vorliegen - ist ein geeigneter Ansprechpartner an der Hochschule bekannt?*

1. Wie häufig ändern sich einzelne Details zu einem Modul? Details sind z.B. Verantwortliche(r), Angestrebte Lernergebnisse, Gruppengröße...
  - welche Details ändern sich am häufigsten?
2. Wie häufig gibt es neue Module / Wie häufig werden alte Module entfernt?
3. Wie häufig gibt es neue Studiengänge / Wie häufig werden alte Studiengänge entfernt

### Design

4. muss neues Modulhandbuch 1zu1 so aussehen wie altes Modulhandbuch? Oder darf ein neues Design verwendet werden, wenn trotzdem alle Informationen übersichtlich erkennbar sind

### Workflow

5. Wenn es Änderungen am Inhalt eines Modulhandbuches gibt, was passiert dann genau?
  - 5.2 Wer ist für welche Aufgabe zuständig?
6. Werden die englischen Handbücher nach dem selben Prinzip manuell erstellt, oder passiert das automatisch?
7. Welche Schwierigkeiten & Probleme haben Sie mit dem aktuellen Prozess?

### Ideen

8. Haben Sie Vorstellungen / Anforderungen / Ideen für das neue System?

# Literaturverzeichnis

- [1] E. Commission, S. Directorate-General for Education Youth, und Culture, *ECTS users' guide 2015*. Publications Office of the European Union, 2015. doi: doi/10.2766/87192.
- [2] C.-D. Hachmeister, U. Müller, und F. Ziegele, „Im Blickpunkt: Zu viel Vielfalt? Warum die Ausdifferenzierung der Studiengänge kein Drama ist“, Juli 2016.
- [3] *Niedersächsisches Gesetz- und Verordnungsblatt*, Bd. 5321. 2019.
- [4] Hochschule Hannover - Fakultät II, „Modulhandbücher – Fakultät II“. Zugegriffen: 12. März 2024. [Online]. Verfügbar unter: <https://f2.hs-hannover.de/studium/studierende/modulhandbuecher>
- [5] „Website der Fakultät IV“. Zugegriffen: 18. März 2024. [Online]. Verfügbar unter: <https://f4.hs-hannover.de/>
- [6] Dezernat 3, *Besonderer Teil der Prüfungsordnung für den Bachelor-Studiengang Mediendesigninformatik (MDI) mit dem Abschluss Bachelor of Science der Fakultät IV – Wirtschaft und Informatik, Abteilung Informatik der Hochschule Hannover*, Bd. 7/2023. 2023.
- [7] „HISinOne-Campus: Prüfungen und Veranstaltungen“. Zugegriffen: 12. März 2024. [Online]. Verfügbar unter: <https://www.his.de/hisinone/pruefungen-und-veranstaltungen>
- [8] „HIS eG – Das Softwarehaus der Hochschulen“. Zugegriffen: 12. März 2024. [Online]. Verfügbar unter: <https://www.his.de/>
- [9] „Erfahrungsbericht Universität Hohenheim“. Zugegriffen: 12. März 2024. [Online]. Verfügbar unter: <https://www.his.de/erfahrungsberichte/universitaet-hohenheim>
- [10] „Mein persönliches Hochschulportal - Hochschule Hannover“. Zugegriffen: 12. März 2024. [Online]. Verfügbar unter: <https://campusmanagement.hs-hannover.de/>
- [11] S. Kleuker, *Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten*. Wiesbaden: Springer Fachmedien Wiesbaden, 2013. doi: 10.1007/978-3-658-00642-6.
- [12] C. Rupp, *Requirements-Engineering und -Management: aus der Praxis von klassisch bis agil*, 6., aktualisierte und erweiterte Auflage. München: Hanser, 2014.
- [13] „Lernanwendungen Allerkamp – Fakultät IV“. Zugegriffen: 8. April 2024. [Online]. Verfügbar unter: <https://f4.hs-hannover.de/ueber-uns/personen/lehrende/abteilung-informatik/professorinnenprofessoren/lernanwendungen-allerkamp/>

- [14] „NestJS - A progressive Node.js framework“. Zugegriffen: 16. März 2024. [Online]. Verfügbar unter: <https://nestjs.com/>
- [15] „Angular - Workspace and project file structure“. Zugegriffen: 19. März 2024. [Online]. Verfügbar unter: <https://angular.io/guide/file-structure>
- [16] J. Goll, *Entwurfsprinzipien und Konstruktionskonzepte der Softwaretechnik*. Wiesbaden: Springer Fachmedien Wiesbaden, 2018. doi: 10.1007/978-3-658-20055-8.
- [17] R. C. Martin, *Clean code: a handbook of agile software craftsmanship*, Repr. in Robert C. Martin series. Upper Saddle River, NJ Munich: Prentice Hall, 2012.
- [18] T. Studer, *Relationale Datenbanken: Von den theoretischen Grundlagen zu Anwendungen mit PostgreSQL*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019. doi: 10.1007/978-3-662-58976-2.
- [19] „Entity Relationship Diagram | Pintora“. Zugegriffen: 6. August 2024. [Online]. Verfügbar unter: <https://pintorajs.vercel.app/docs/diagrams/er-diagram/>
- [20] J. Tidwell, *Designing interfaces: patterns for effective interaction design*, Dritte Auflage. Sebastopol, CA: O'Reilly, 2020.
- [21] „Hochschule Hannover“. Zugegriffen: 12. Mai 2024. [Online]. Verfügbar unter: <https://www.hs-hannover.de/>
- [22] S. Tilkov, M. Eigenbrodt, S. Schreier, und O. Wolf, *REST und HTTP: Entwicklung und Integration nach dem Architekturstil des Web*, 3., aktualisierte und erweiterte Auflage. Heidelberg: dpunkt.verlag, 2015.
- [23] „Relations (Reference) | Prisma Documentation“. Zugegriffen: 5. August 2024. [Online]. Verfügbar unter: <https://www.prisma.io/docs/orm/prisma-schema/data-model/relations>
- [24] „Prisma Schema Overview | Prisma Documentation“. Zugegriffen: 5. August 2024. [Online]. Verfügbar unter: <https://www.prisma.io/docs/orm/prisma-schema/overview>
- [25] „Controllers: Documentation | NestJS“. Zugegriffen: 7. Juni 2024. [Online]. Verfügbar unter: <https://docs.nestjs.com/controllers>
- [26] „Dependency injection in Angular“. Zugegriffen: 1. Juni 2024. [Online]. Verfügbar unter: <https://angular.dev/guide/di>
- [27] „Documentation | NestJS - A progressive Node.js framework“. Zugegriffen: 16. März 2024. [Online]. Verfügbar unter: <https://docs.nestjs.com/>

- [28] „Destructuring assignment - JavaScript | MDN“. Zugegriffen: 21. Juli 2024. [Online]. Verfügbar unter: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring\\_assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)
- [29] „Relation queries (Concepts) | Prisma Documentation“. Zugegriffen: 21. Juli 2024. [Online]. Verfügbar unter: <https://www.prisma.io/docs/orm/prisma-client/queries/relation-queries>
- [30] „Transactions and batch queries (Reference) | Prisma Documentation“. Zugegriffen: 21. Juli 2024. [Online]. Verfügbar unter: <https://www.prisma.io/docs/orm/prisma-client/queries/transactions>
- [31] „Object.keys() - JavaScript | MDN“. Zugegriffen: 24. Juli 2024. [Online]. Verfügbar unter: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Object/keys](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys)
- [32] „Array.prototype.filter() - JavaScript | MDN“. Zugegriffen: 24. Juli 2024. [Online]. Verfügbar unter: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Array/filter](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter)
- [33] „RxJS - lastValueFrom“. Zugegriffen: 8. August 2024. [Online]. Verfügbar unter: <https://rxjs.dev/api/index/function/lastValueFrom>
- [34] „Array.prototype.reduce() - JavaScript | MDN“. Zugegriffen: 24. Juli 2024. [Online]. Verfügbar unter: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Array/reduce](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce)
- [35] „YtoTech/latex-on-http“. Zugegriffen: 3. Juni 2024. [Online]. Verfügbar unter: <https://github.com/YtoTech/latex-on-http>
- [36] „NestJS - Streaming files“. Zugegriffen: 3. Juni 2024. [Online]. Verfügbar unter: <https://docs.nestjs.com/techniques/streaming-files>
- [37] „Typst: Compose papers faster“. Zugegriffen: 24. Juli 2024. [Online]. Verfügbar unter: <https://typst.app/>
- [38] „npm | Home“. Zugegriffen: 10. August 2024. [Online]. Verfügbar unter: <https://www.npmjs.com/>
- [39] „npm: Threats and Mitigations | npm Docs“. Zugegriffen: 6. August 2024. [Online]. Verfügbar unter: <https://docs.npmjs.com/threats-and-mitigations#by-typosquatting%E2%80%93dependency-confusion>
- [40] „Open-Source-Sicherheit verständlich erklärt“. Zugegriffen: 6. August 2024. [Online]. Verfügbar unter: <https://snyk.io/de/series/open-source-security/>
- [41] „Angular powered Bootstrap“. Zugegriffen: 1. Juni 2024. [Online]. Verfügbar unter: <https://ng-bootstrap.github.io/>

- [42] „PrimeNG - Angular UI Component Library“. Zugegriffen: 1. Juni 2024. [Online]. Verfügbar unter: <https://primeng.org/>
- [43] „How to add CSS“. Zugegriffen: 5. August 2024. [Online]. Verfügbar unter: [https://www.w3schools.com/css/css\\_howto.asp](https://www.w3schools.com/css/css_howto.asp)
- [44] „Sass: Syntactically Awesome Style Sheets“. Zugegriffen: 5. August 2024. [Online]. Verfügbar unter: <https://sass-lang.com/>
- [45] „Sass: Style Rules“. Zugegriffen: 5. August 2024. [Online]. Verfügbar unter: <https://sass-lang.com/documentation/style-rules/#nesting>
- [46] „Angular - ViewEncapsulation“. Zugegriffen: 5. August 2024. [Online]. Verfügbar unter: <https://angular.io/api/core/ViewEncapsulation>
- [47] „CSS grid layout - CSS: Cascading Style Sheets | MDN“. Zugegriffen: 5. August 2024. [Online]. Verfügbar unter: [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_grid\\_layout](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout)
- [48] „unset - CSS: Cascading Style Sheets | MDN“. Zugegriffen: 5. August 2024. [Online]. Verfügbar unter: <https://developer.mozilla.org/en-US/docs/Web/CSS/unset>
- [49] A. Ertel und K. Laborenz, *Responsive Webdesign - Konzepte, Techniken, Praxisbeispiele*, 2. Aufl.
- [50] „Angular Table Component“. Zugegriffen: 5. August 2024. [Online]. Verfügbar unter: <https://primeng.org/table#responsive-stack>
- [51] „flex-direction - CSS: Cascading Style Sheets | MDN“. Zugegriffen: 5. August 2024. [Online]. Verfügbar unter: <https://developer.mozilla.org/en-US/docs/Web/CSS/flex-direction>
- [52] „RxJS - BehaviorSubject“. Zugegriffen: 1. Juni 2024. [Online]. Verfügbar unter: <https://rxjs.dev/api/index/class/BehaviorSubject>
- [53] „Angular Internationalization“. Zugegriffen: 1. Juni 2024. [Online]. Verfügbar unter: <https://angular.dev/guide/i18n>
- [54] Google, „Example Angular Internationalization application“. Zugegriffen: 31. Mai 2024. [Online]. Verfügbar unter: <https://angular.dev/guide/i18n/example>
- [55] „ngx-translate/core“. Zugegriffen: 1. Juni 2024. [Online]. Verfügbar unter: <https://github.com/ngx-translate/core>
- [56] „Transloco Angular i18n“. Zugegriffen: 1. Juni 2024. [Online]. Verfügbar unter: <https://jsverse.github.io/transloco/>
- [57] „@ngneat/transloco“. Zugegriffen: 1. Juni 2024. [Online]. Verfügbar unter: <https://www.npmjs.com/package/@ngneat/transloco>

- [58] „Angular - HTTP Client: Interceptors“. Zugegriffen: 1. Juni 2024. [Online]. Verfügbar unter: <https://angular.dev/guide/http/interceptors>
- [59] „Angular - Component Lifecycle“. Zugegriffen: 30. Juli 2024. [Online]. Verfügbar unter: <https://v17.angular.io/guide/lifecycle-hooks>
- [60] „Map - JavaScript | MDN“. Zugegriffen: 3. August 2024. [Online]. Verfügbar unter: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Map](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map)
- [61] „Regular expressions - JavaScript | MDN“. Zugegriffen: 29. Juli 2024. [Online]. Verfügbar unter: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular\\_expressions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions)
- [62] „Build optimized websites quickly, focus on your content | Docusaurus“. Zugegriffen: 7. Juli 2024. [Online]. Verfügbar unter: <https://docusaurus.io/>
- [63] „Docusaurus Site Showcase | Docusaurus“. Zugegriffen: 7. Juli 2024. [Online]. Verfügbar unter: <https://docusaurus.io/showcase>
- [64] „KI-Suche, die versteht“. Zugegriffen: 7. Juli 2024. [Online]. Verfügbar unter: <https://www.algolia.com/de/>
- [65] „Create a new crawler | Algolia“. Zugegriffen: 7. Juli 2024. [Online]. Verfügbar unter: <https://www.algolia.com/doc/tools/crawler/getting-started/create-crawler/>
- [66] „Podman Desktop - Containers and Kubernetes | Podman Desktop“. Zugegriffen: 26. Juli 2024. [Online]. Verfügbar unter: <https://podman-desktop.io/>
- [67] „Docker: Accelerated Container Application Development“. Zugegriffen: 26. Juli 2024. [Online]. Verfügbar unter: <https://www.docker.com/>
- [68] „Docker Deployment | Docusaurus.community“. Zugegriffen: 26. Juli 2024. [Online]. Verfügbar unter: <https://docusaurus.community/knowledge/deployment/docker/>
- [69] „Dockerfile reference“. Zugegriffen: 26. Juli 2024. [Online]. Verfügbar unter: <https://docs.docker.com/reference/dockerfile/>
- [70] „index | Alpine Linux“. Zugegriffen: 26. Juli 2024. [Online]. Verfügbar unter: <https://alpinelinux.org/>
- [71] „Docker build cache | Docker Docs“. Zugegriffen: 26. Juli 2024. [Online]. Verfügbar unter: <https://docs.docker.com/build/cache/>
- [72] „Volumes | Docker Docs“. Zugegriffen: 26. Juli 2024. [Online]. Verfügbar unter: <https://docs.docker.com/storage/volumes/>
- [73] ilovepdf.com, „iLovePDF | Online PDF tools for PDF lovers“. Zugegriffen: 9. August 2024. [Online]. Verfügbar unter: <https://www.ilovepdf.com/>

[74] „Qodana: Static Code Analysis Tool by JetBrains“. Zugegriffen: 12. August 2024.  
[Online]. Verfügbar unter: <https://www.jetbrains.com/qodana/>

