JOHANNES KEPLER
UNIVERSITÄT LINZ

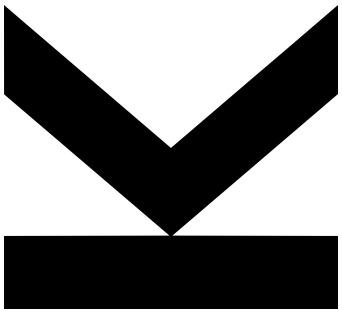
Eingereicht von
Johann Reichl
Matrikelnummer:
K01256381
Studienkennzahl:
033 521

Einreichung am
Institut für Systemsoftware

Beurteiler
o.Univ.-Prof. Dr. Dr.h.c.
Hanspeter Mössenböck

Oktober 2023

Automatisches Ausfüllen von Prüfungsrastern



Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Science

im Bachelorstudium

Informatik (UK 033/521)

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

JOHANNES KEPLER
UNIVERSITÄT LINZ

Kurzfassung

In dieser Arbeit wird ein innovatives Online-Tool präsentiert, welches das Ausfüllen von Prüfungsrastern für Studierende vereinfacht, indem es Prüfungen automatisch entsprechend den Studienplanspezifikationen zuordnet. Mit der zunehmenden Komplexität der Studienpläne gerät die aktuell verwendete domänenspezifische Sprache (DSL), repräsentiert durch eine attributierte Grammatik (ATG), an ihre Grenzen. Um den anspruchsvolleren Studienplanspezifikationen gerecht zu werden, wird in dieser Arbeit die bestehende DSL so erweitert, dass sie in der Lage ist, auch komplexere Anforderungen zu erfassen. In diesem Kontext werden die wesentlichen Schritte und Konzepte systematisch erläutert, die zur Erreichung dieses Projektziels notwendig sind.

Abstract

This thesis presents an innovative online tool that simplifies the completion of exam grids for students by automatically assigning exams to match study plan specifications. As study plans become more complex, the current domain-specific language (DSL) represented by an attributed grammar (ATG) is reaching its limits. In order to cope with the more sophisticated study plan specifications, this work extends the existing DSL to be capable of handling more complex requirements. In this scope, the essential steps and concepts necessary to achieve this project goal are systematically explained.

JOHANNES KEPLER
UNIVERSITÄT LINZ

Inhaltsverzeichnis

JOHANNES KEPLER
UNIVERSITÄT LINZ

Abbildungsverzeichnis

JOHANNES KEPLER
UNIVERSITÄT LINZ

Tabellenverzeichnis

JOHANNES KEPLER
UNIVERSITÄT LINZ

1 Einleitung/Motivation

Das Institut für Systemsoftware bietet ein Online-Tool an, welches den Prozess des automatischen Ausfüllens von Prüfungsrastern übernimmt. Zuvor mussten Studierende dies manuell am Ende ihres Studiums erledigen. Dieses Tool weist bereits abgelegte Prüfungen automatisch den entsprechenden Abschnitten hinzu. Studenten haben somit über den gesamten Studienzeitraum hinweg einen Überblick sowohl über bereits absolvierte als auch über ausstehende Prüfungen. Die Regeln für die Zuordnung der Prüfungen zu den bestimmten Bereichen im Studienplan werden durch eine domänenspezifische Sprache (DSL = Domain Specific Language) definiert.

Während die aktuelle DSL ausreicht, um die Bedingungen in Studienplänen wie Informatik abzubilden, gibt es andere Studienrichtungen die aufgrund ihrer Komplexität nicht beschrieben werden können. Das Ziel dieser Arbeit besteht darin, die DSL so zu erweitern, dass sie eine breitere Vielfalt an Studienplänen erfassen kann. Insbesondere sollen die Bachelor-Studiengänge Informatik und Künstliche Intelligenz (AI) sowie die entsprechenden Master-Studiengänge in die Erweiterung einbezogen werden. Darüber hinaus soll das vorhandene Programm modifiziert werden, um diese erweiterten Bedingungen in der automatischen Prüfungszuordnung zu berücksichtigen.

Die DSL wird durch eine attributierte Grammatik (ATG) spezifiziert, aus der mithilfe des Compilergenerators Coco/R sowohl ein Scanner als auch ein Parser generiert werden. Diese ATG enthält neben den syntaktischen Regeln auch semantische Aktionen, die den Parser so ergänzen, dass die zugehörigen Objekte für die Nutzung im Frontend vorbereitet und übertragen werden können. Im Frontend werden dann die übertragenen Daten bzw. Objekte gemäß der Spezifikation weiterverarbeitet, um letztlich das Prüfungsraster inklusive aller möglichen Zuordnungen korrekt darzustellen.

In den folgenden Kapiteln beleuchten wir den Background, Analysieren das System und befassen uns eingehend mit der Implementierung dieser Aufgabe. Anschließend wird eine Anleitung präsentiert, eine Evaluierung der Software vorgenommen und ein abschließender Ausblick gegeben.

2 Background/Analyse

Die heutige technologische Landschaft bietet eine Fülle von Möglichkeiten, um komplexe Herausforderungen und Aufgaben effektiv zu bewältigen. Die Verwendung der richtigen Technologien und Architekturen ist dabei von zentraler Bedeutung, um sowohl aktuelle Anforderungen gerecht zu werden, als auch zukünftige Erweiterungen zu ermöglichen.

Für die zu realisierende Erweiterung stützen wir uns auf bereits genutzte und bewährte Technologien und Architekturen, die nicht nur populär, sondern auch bestens für die vorgesehene Anwendung geeignet sind. Als zentrale Datenquelle wird dabei die Oracle-Datenbank verwendet, die vom Informationsmanagement (IM) der JKU bereitgestellt und betreut wird. Mit Java und dem Framework Spring Boot werden im Backend relevante Tabellen zu Entities transformiert, welche die grundlegende Datenstruktur darstellen. Diese dienen als Basis, um die Struktur der in der DSL definierten Spezifikationen zu formen. Das Frontend, realisiert mit Angular, übernimmt die anschließende Verarbeitung und visuelle Darstellung dieser Daten.

In diesem Kapitel beleuchten wir die eingesetzten Technologien und Kernstrukturen, die dem Tool zugrunde liegen. Dabei wird bei den Datenstrukturen lediglich ein Überblick gegeben, um den Umfang dieser Arbeit nicht zu übersteigen.

2.1 Datenbank: Oracle

In der Welt der relationalen Datenbankmanagementsysteme nimmt Oracle eine führende Rolle ein was Performance, Skalierbarkeit, Zuverlässigkeit und Sicherheit angeht. Diese Merkmale sind nicht nur Schlagworte, sondern bilden vielmehr das Fundament von Oracles Reputation als eines der global renommiertesten Datenbanksysteme [1].

Die Oracle-Datenbank des IM stellt für die Anwendung eine geeignete Grundlage dar. Für die Entwicklungsarbeiten wurde ausschließlich die Test-Datenbank verwendet, die etwa fünfmal langsamer ist als die Produktiv-Datenbank. Dies hatte zur Folge, dass der Programmstart durch das Einlesen der für das Erstellen der Curricula erforderlichen Daten rund 10 Minuten

in Anspruch nahm. Um diesen Engpass zu beheben, wurde vor der Implementierung der eigentlichen Aufgabe, eine Performance-Optimierung durchgeführt, die den Programmstart während der Entwicklungsphase signifikant beschleunigte. Weitere Einzelheiten werden dazu im späteren Verlauf in Abschnitt 5.1.1 präsentiert. Bestehende Datenstrukturen in Form von Entities wurden nur geringfügig modifiziert. Für eine detaillierte Datenanalyse kam der SQL Developer zum Einsatz. Nachstehend sind die relevanten Tabellen mit ihren zugehörigen Entity-Namen und Beschreibungen aufgelistet, welche bereits die Brücke zum Backend in Java schlagen:

Tabellenname	**Entityname**	**Beschreibung**
VIEWSTUPLA	StudyplanId	Statistikdaten
TABCOUVAR	CourseVariant	Klassenvariante
TABSTUPLA	Studyplan	Studienplan Kopfdaten
TABSTUPLANAM	StudyplanName	Logtabelle zu Kennzahlensystem
TABSTUPLASEC	StudyplanSection	Studienplan Abschnitt
TABSUB	Subject	Studienplan Prüfungsfach
TABSUBTEX	SubjectText	Studienplan Fach-Texte
TABSUBREL	SubjectRelation	Studienplan Zuordnung/Voraussetzung
TABCOUSEM	CourseSemester	Veranstaltung - Abhaltung
TABSINEXA	SingleExam	Einzelprüfungsdaten
TABUSEANNSOU	UsedAnnotationSource	Fachquelle
TABUSEANN	UsedAnnotation	Verwendungsannotierungen
TABPAREXA	PartExam	Teilprüfungsdaten
TABREC	Recognition	Teilprüfung-Anrechnung
TABSEMVAL	SemesterValidity	Studienplan Semester-Gültigkeit

Tabelle 2.1: Entity-Mapping der Datenbanktabellen

In der oben gezeigten Tabelle ist anzumerken, dass nicht alle aufgeführten Einträge tatsächliche Tabellen darstellen. Der erste Eintrag ist nämlich eine Sicht (*View*). Ein eindeutiges Erkennungsmerkmal dieser Aufstellung ist, dass alle Tabellen mit "TAB" beginnen, während Sichten mit "VIEW" anfangen. Bei genauer Betrachtung des Spalteninhalts für die Tabellennamen wird deutlich, dass der erste Eintrag mit "VIEW" startet. Dies verweist auf eine Sicht, die in diesem Fall auf eine Tabelle basiert. Diese spezifische Sicht stellt nur einen Bruchteil der Spalten aus der zugrunde liegenden Tabelle dar, liefert jedoch alle essenziellen Informationen, die das Programm benötigt.

2.2 Backend: Java mit Spring Boot

Im Backend des Systems spielt Java in Kombination mit Spring Boot eine zentrale Rolle. Java bietet durch seine Robustheit und Plattformunabhängigkeit eine zuverlässige Basis [2]. Spring Boot, ein Framework zur Vereinfachung der Anwendungsentwicklung (siehe Abschnitt 2.2.3), ergänzt das, indem es den Entwicklungsaufwand minimiert und eine nahtlose Integration mit Datenbanken ermöglicht. Für die Entwicklung des Backends wurde IntelliJ IDEA Ultimate als integrierte Entwicklungsumgebung (IDE) verwendet. Sie bietet durch ihre umfangreichen Funktionen und Integrationen einen signifikanten Vorteil für den Entwicklungsprozess. Zudem ermöglicht Spring Boot das direkte Einbetten von Servern wie Tomcat, wodurch ein WAR-Datei-Deployment überflüssig wird. Dies, kombiniert mit weiteren Features macht Spring Boot zu einer effizienten Wahl für dieses Projekt [3]. Zusammen bieten Java und Spring Boot in Verbindung mit IntelliJ eine robuste und benutzerfreundliche Plattform für die Backend-Entwicklung.

Im weiteren Verlauf dieses Abschnitts werden wir tiefer in die technischen Aspekte des Backends eintauchen. Zuerst beschäftigen wir uns mit den Entities, die bereits im vorherigen Abschnitt kurz angeschnitten wurden. Anschließend beleuchten wir die Datenintegration im Zusammenspiel mit den jeweiligen Repository-Interfaces, welche sowohl bei der Erstellung der Curricula als auch beim Abrufen der notwendigen Informationen für die Studierenden eine wichtige Rolle spielen. Im letzten Teil dieses Abschnitts wird dann noch Spring Boot präsentiert und der Einsatz als Kommunikationswerkzeug zum Frontend erläutert.

2.2.1 Entities

Nachdem im vorherigen Abschnitt die Datenbanktabellen und ihre zugehörigen Entities in der Tabelle 2.1 vorgestellt wurden, ist es nun entscheidend, die Beziehungen und Struktur dieser Entities zu betrachten. Die Verknüpfungen der Entities geben Einblick in die logische Organisation der Daten, wie sie im Backend repräsentiert und genutzt werden. In Tools wie diesem sind solche Beziehungen unerlässlich, um sowohl Datenintegrität als auch den effizienten Zugriff und die Handhabung der Daten zu gewährleisten. Abbildung 2.1 illustriert diese Struktur und die Beziehungen, sodass ein umfassendes Bild des Datenmodells entsteht.

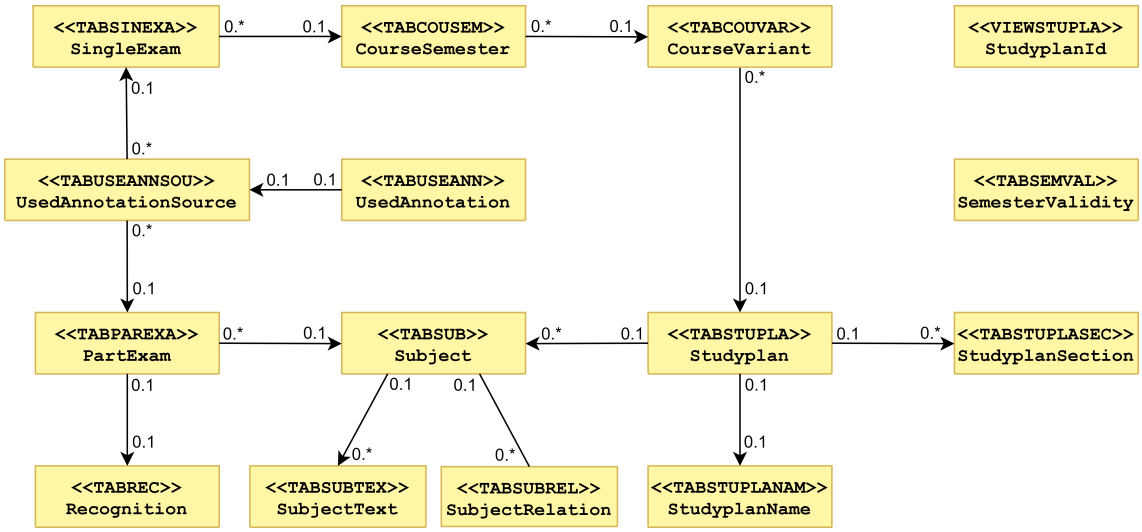


Abbildung 2.1: Struktur der Entities

Zum Erstellen aller Diagramme, die primär aus Klassendiagrammen bestehen und einen besseren Überblick über die Datenstruktur im Frontend und Backend bieten, wurde Draw.io verwendet. Die nachfolgenden Informationen zu diesem Abschnitt basieren auf Inhalten aus der Hibernate Dokumentation [4].

In dem dargestellten Diagramm repräsentieren die Verbindungselemente spezifische Beziehungen zwischen den Klassen. Fast alle Verbindungen sind durch Pfeile dargestellt. Diese symbolisieren eine unidirektionale Beziehung, bei der eine Klasse Zugriff bzw. Kenntnis von einer anderen Klasse hat, die andere Klasse jedoch nicht umgekehrt. Das bedeutet, dass in den meisten Fällen eine Klasse auf eine andere zugreift, ohne dass die zweite Klasse Informationen über die erste besitzt. Es gibt jedoch eine Ausnahme. Eine Linie ohne Pfeil zeigt eine bidirektionale Beziehung an, und zwar zwischen `Subject` und `SubjectRelation`. Hier haben beide beteiligten Klassen Zugriff aufeinander und sind sich gegenseitig bekannt. Durch diese klare Unterscheidung wird die Abhängigkeitsrichtung zwischen den Klassen deutlich. Zudem sind zwei Klassen im Diagramm dargestellt, die keine Verbindungen aufweisen. Dies deutet darauf hin, dass diese Klassen in diesem Kontext keine direkten Abhängigkeiten oder Beziehungen zu anderen Klassen besitzen.

Zusätzlich zu den bereits erklärten Verbindungen sind in dem Diagramm Kardinalitäten zu erkennen, welche die Beziehungen zwischen den Entities weiter konkretisieren. In einem Entity-Relationship-Diagramm geben Kardinalitäten an, wie viele Instanzen einer Entität

in Beziehung zu einer Instanz einer anderen Entität stehen können. In diesem Kontext wurden keine expliziten Einschränkungen in den Annotierungen der Entities vorgenommen, weshalb stets eine optionale Kardinalität vorliegt. Mehr zu diesen Annotierungen wird später im Kapitel Evaluierung im Abschnitt 5.1 erläutert.

Im Diagramm sind folgende Kardinalitäten zu finden:

0.1 ⟶ 0.1: `OneToOne` Beziehung
Sie zeigt an, dass genau eine Instanz eines Entity-Typs mit genau einer Instanz eines anderen Entity-Typs verknüpft sein kann.

0.1 ⟶ 0.*: `OneToMany` Beziehung
Diese symbolisiert, dass eine Instanz eines Entity-Typs mit beliebig vielen Instanzen eines anderen Entity-Typs in Beziehung stehen kann.

0.* ⟶ 0.1: `ManyToOne` Beziehung
Hier wird anzeigt, dass mehrere Instanzen eines Entity-Typs mit genau einer Instanz eines anderen Entity-Typs verknüpft sein können.

Ein praktisches Beispiel hierfür ist das Entity `Subject`, das über eine `OneToMany` Beziehung eine Liste von `SubjectRelation`-Objekten besitzt:

```
@OneToMany
@JoinColumn(name = "SUBRELID")
private List<SubjectRelation> relations;
```

Die umgekehrte Richtung im Entity `SubjectRelation` zeigt eine `ManyToOne` Beziehung zum `Subject`:

```
@ManyToOne
@JoinColumn(name = "SUBID")
Subject subject;
```

Die Nutzung von Entity-Beziehungen, wie `OneToMany`, `ManyToOne` und `OneToOne`, trägt maßgeblich zu einem effizienten Datenmanagement im Tool bei. Diese Beziehungen spiegeln die Struktur und die Zusammenhänge der Daten wider, ohne dass Entwickler sich direkt mit den SQL-Details auseinandersetzen müssen. Sie stellen eine Brücke zwischen dem objektorientierten Modell des Tools und der relationalen Datenbankstruktur dar, in der sie auf intuitive Weise abgebildet werden.

2.2.2 Datenintegration

Die Datenintegration stellt sicher, dass Informationen im System effizient fließen und in einem nutzbaren Format bereitgestellt werden. In diesem Tool, das mit diversen Datenquellen und Beziehungen arbeitet, ist es wichtig, dass man eine klare und funktionierende Integrationsstrategie hat. In diesem Abschnitt wird aufgezeigt, wie das System Daten integriert, verarbeitet und bereitstellt.

Um die Prozesse und die Struktur der Datenintegration besser zu visualisieren und zu veranschaulichen, wurde die folgende Abbildung erstellt:

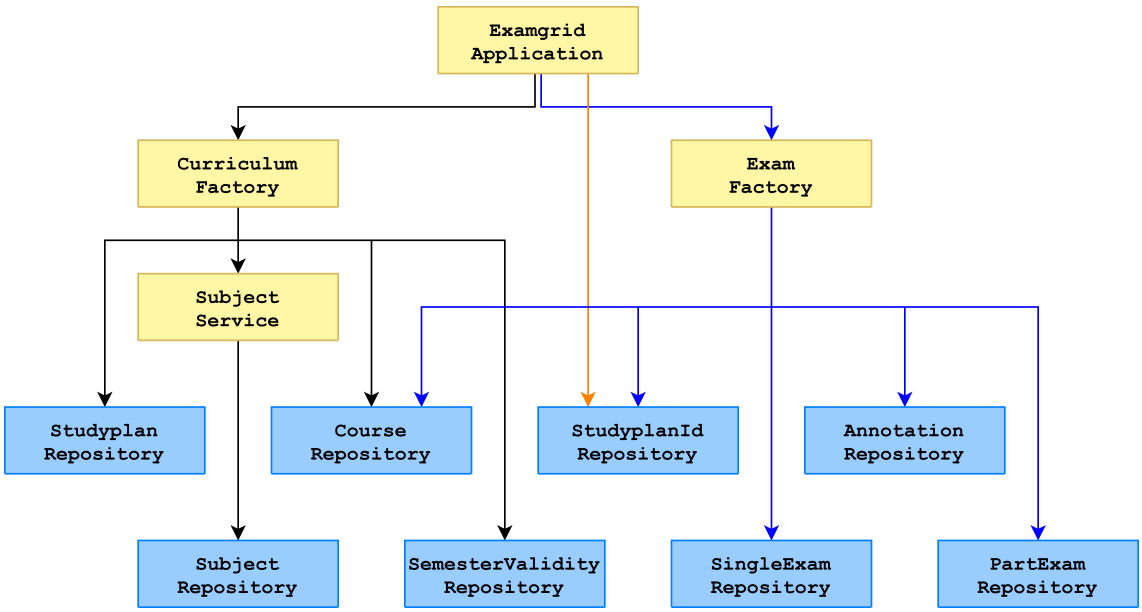


Abbildung 2.2: Architekur der Datenintegrationsprozesse

In dieser Grafik symbolisieren gelbe Elemente Klassen, wohingegen blaue Elemente für Interfaces stehen. Die farbliche Kodierung der Pfeile gibt Aufschluss über ihre jeweilige Funktion: Schwarze Pfeile stehen für Backend-Operationen, während blaue Pfeile und der orange Pfeil Anforderungen aus dem Frontend repräsentieren.

Ganz oben in der Mitte der Grafik befindet sich die `ExamgridApplication`, die das Herzstück und Hauptprogramm des Tools im Backend darstellt. Von hier aus werden sämtliche Datenanforderungen gesteuert und an die entsprechenden Einheiten weitergeleitet.

Links im Diagramm wird die `CurriculumFactory` durch schwarze Pfeile mit ihren zugehörigen Komponenten dargestellt. Sie ist spezialisiert auf die Erstellung und Aufbereitung von Curricula für einen aus der ATG generierten Parser. Für diesen Prozess zieht sie Informationen aus verschiedenen Repositories und dem `SubjectService` heran. Der `SubjectService` entstand im Zuge von Maßnahmen zur bereits erwähnten Performance-Optimierung. In diesem Zusammenhang wurden das `SubjectTextRepository` und das `SubjectRelationRepository` entfernt und durch den `SubjectService` ersetzt. Innerhalb eines Curriculums werden diverse Themenbereiche zusammengefasst. Im Rahmen dieses Projekts werden diese Themenbereiche oft als Fächer, Fachgebiete, Kategorien oder auch als *Subjects* bezeichnet, auf die im Abschnitt 2.4 im Kontext der DSL erneut Bezug genommen wird. Diese *Subjects* weisen häufig eine hierarchische Struktur auf und umfassen eine Menge an Kursen oder Lehrveranstaltungen. In grafischer Hinsicht könnte man sich die *Subjects* als Knoten (*Nodes*) und die Kurse als Endpunkte (*Leafs*) eines Baums vorstellen.

Im rechten Teil des Diagramms ist die `ExamFactory` zu sehen, die mittels blauer Pfeile mit ihren zugehörigen Repositories verknüpft ist. Sie erfüllt zwei zentrale Aufgaben: Einerseits die Abfrage spezifischer Details zu Studienplänen, insbesondere Studienkennzahlen, und andererseits das Abrufen abgeschlossener Kurse und Prüfungen von Studenten. Wenn Daten vom Frontend angefordert werden, kommt die `ExamFactory` zum Einsatz. Ein paralleles Beispiel für Frontend-Anforderungen zeigt sich bei der Verbindung zwischen `ExamgridApplication` und `StudyplanIdRepository`, die durch einen orangen Pfeil repräsentiert wird. Im Gegensatz zu den blauen Pfeilen wird hierbei die `ExamFactory` übersprungen und spezifisch für die Erzeugung des Studienraster-PDFs und das Senden des Rasters an den Prüfungs- und Anerkennungsservice (PAS) direkt auf die `StudyplanIdRepository`, zur Ermittlung der Studienkennzahl, zugegriffen, ohne Umwege über zusätzliche Klassen oder Services.

Nachdem nun die Struktur der Datenintegration und ihre Interaktionen mit dem Frontend dargelegt wurde, wird ein tieferer Blick auf das zugrundeliegende Spring Boot Framework geworfen, die diese Kommunikation ermöglicht. Dieses Framework dient als Rückgrat für das Backend und orchestriert die Datenbereitstellung für das Frontend, das in Angular entwickelt wurde. Im folgenden Abschnitt wird näher auf die Funktionsweise und Besonderheiten von Spring Boot eingegangen und wie es im Kontext dieses Projekts eingesetzt wird.

2.2.3 Spring Boot

Spring Boot ist ein Framework, das die Entwicklung eigenständiger, sofort einsetzbarer Java-Anwendungen erleichtert, indem es komplexe Konfigurationen ablöst und eine leichtgängige Einrichtung für vielfältige Funktionen bietet [3]. Dank Spring Boot wird in unserem Projekt die Kommunikation mit dem Angular-Frontend effizient realisiert. Nachfolgend ein Überblick über diese Kommunikation:

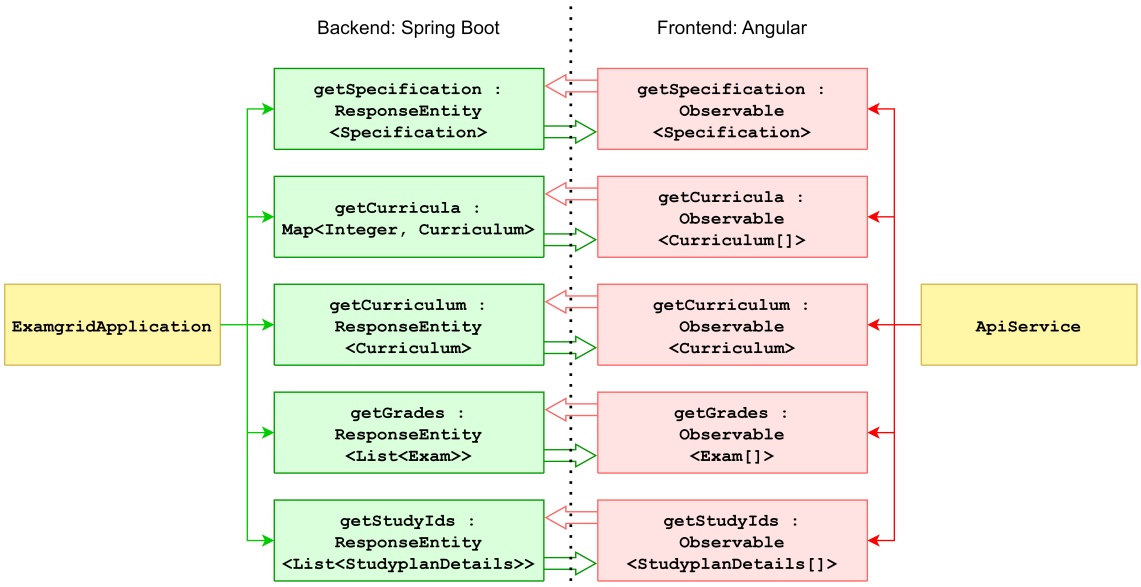


Abbildung 2.3: RESTful Kommunikationsarchitektur: Spring Boot ↔ Angular

Das dargestellte Diagramm zeigt die Interaktion zwischen dem Spring Boot Backend und dem Angular Frontend. Die bereits vorhandenen RESTful-Services ermöglichen es dem Frontend, Daten anzufordern. Die bereitgestellten Daten, die aus den zuvor beschriebenen Komponenten stammen (siehe Abbildung 2.2), werden als JSON-Objekte an das Frontend übermittelt. In Angular kann dies dann unkompliziert verarbeitet werden [5].

Ein zentraler Vorteil dieser Architektur ist die klare Trennung von Frontend und Backend, sodass Modifikationen auf einer Seite kaum Einfluss auf die andere haben. Im Laufe dieses Projekts wird trotz anstehenden Änderungen in der Datenstruktur, wie im Kapitel Implementierung beschrieben, die Kommunikation zwischen Backend und Frontend gleich bleiben. Abschließend bestätigt die Erfahrung aus diesem Projekt, dass Spring Boot aufgrund seiner Funktionalität und Handhabung hervorragend für das Backend geeignet ist.

2.3 Frontend: Angular

Angular ist ein populäres Open-Source-Framework für die Webentwicklung, das von Google entwickelt wurde. Es wurde mit dem Ziel konzipiert, Entwicklern den Aufbau von umfangreichen, erweiterbaren und leistungsfähigen Webapplikationen zu vereinfachen. Durch die Nutzung von TypeScript als primäre Programmiersprache ermöglicht Angular einen gut strukturierten und typsicheren Entwicklungsprozess. Dank seiner modularen Struktur und den fortschrittlichen Datenbindungstechniken wird es in vielen Webprojekten bevorzugt. Unterstützt durch einen reaktiven Programmierstil, lassen sich mit Angular dynamische und responsive Webanwendungen realisieren. Detaillierte Informationen und weiterführende Erklärungen zu Angular lassen sich in den angegebenen Quellen nachlesen [6][7].

Um die Architektur und den Aufbau des in diesem Projekt verwendeten Angular-Frontends zu veranschaulichen, wird im folgenden Diagramm die Struktur dargestellt:

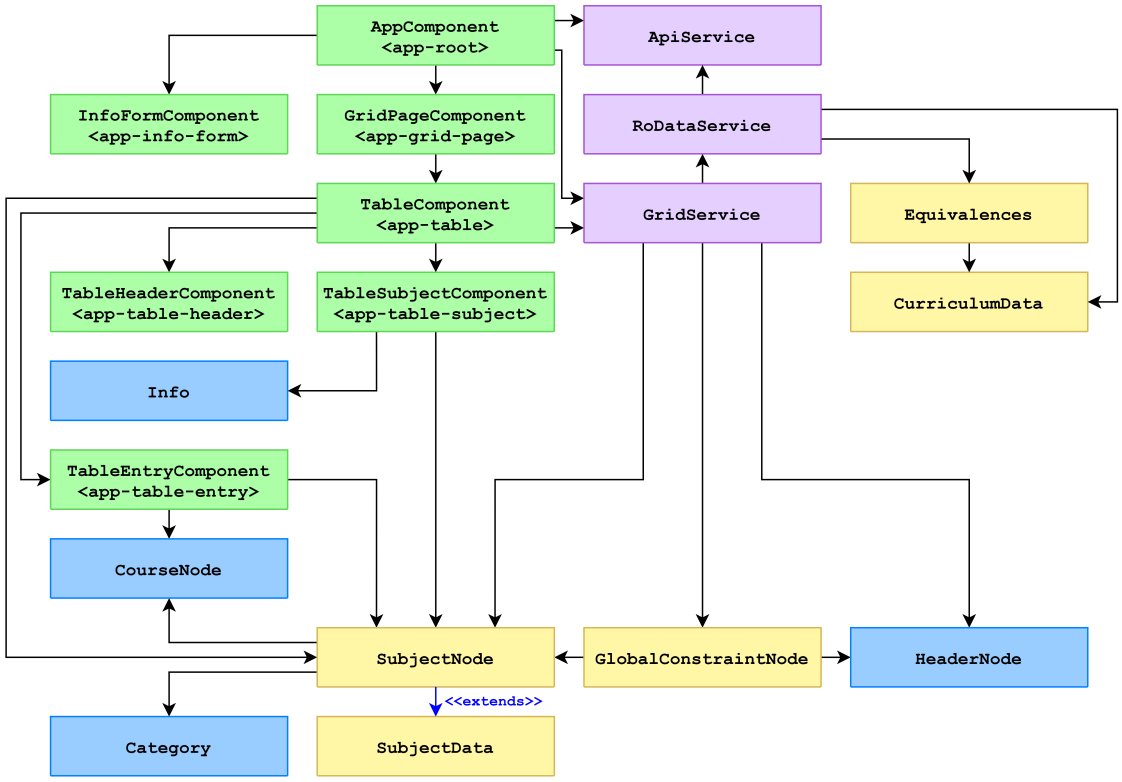


Abbildung 2.4: Architektur des Angular-Frontends

Die grün dargestellten Komponenten bilden Kern der grafischen Darstellung in dieser Anwendung. Jede Komponente in Angular verfügt in der Regel über vier Dateien: eine HTML-Datei für das UI-Layout, eine Styling-Datei (in unserem Fall SCSS), eine TypeScript-Datei für die dahinterliegende Logik und eine Test-Datei. Diese Komponenten repräsentieren unterschiedliche Bereiche der Prüfungsraster-Webpage, wobei die Pfeile im Diagramm ihre Interaktionen und Abhängigkeiten aufzeigen.

In Lila sind Serviceklassen hervorgehoben, die sowohl Daten als auch Funktionen bereitstellen. Die Klasse `ApiService` ist speziell für den Datenaustausch und die Kommunikation mit dem Backend verantwortlich. Über die im vorherigen Abschnitt gezeigte Darstellung hinaus (siehe Abbildung 2.3) verfügt dieser Service über Methoden zur Raster-PDF-Generierung und zum Senden des Rasters an den Prüfungs- und Anerkennungsservice (PAS). Der `RoDataService` verwendet den `ApiService`, um Daten vom Backend zu beziehen und entsprechend aufzubereiten. Der `GridService` transformiert diese Daten weiter, sodass sie von den grünen Komponenten konsumiert werden können. Er kümmert sich auch um die Initialisierung der visuell dargestellten Informationen und das Aktualisieren der Inhalte nach Benutzerinteraktionen, insbesondere die Zuordnung bereits abgelegter Prüfungen, was für die Erweiterung in diesem Projekt essentiell ist.

Die gelben Entities sind als Datenverarbeitungsklassen konzipiert. Basierend auf den Serviceklassen werden diese Klassen instanziiert. Jede von ihnen bringt eine spezifische Logik mit, um Daten effizient für das Frontend aufzubereiten. Sie beinhalten Funktionen, die Daten filtern, sortieren, gruppieren oder in anderer Weise modifizieren und strukturieren. Dabei liegt ein besonderer Fokus in der Logik, um die Spezifikationen der Studienpläne ideal abzubilden und zu formatieren, sodass sie insbesondere von den grünen Angular-Komponenten optimal dargestellt werden können.

In blau sind vier der insgesamt 27 Interfaces dargestellt, die in diesem Zusammenhang wichtig für die Darstellung der Daten im Frontend sind. In TypeScript, und damit auch in Angular, spielen diese Interfaces eine zentrale Rolle. Sie definieren die erwarteten Datenformate oder -strukturen für verschiedene Teile dieser Anwendung und stellen sicher, dass die Daten korrekt und konsistent über die Anwendung hinweg gehandhabt werden.

Das Diagramm bietet lediglich einen groben Einblick in die Architektur des Angular-Frontends. Mit allen Details wäre es erheblich umfangreicher.

2.4 Domänenspezifische Sprachen und ihre Anwendung

Domänenspezifische Sprachen, auch als DSLs (aus dem Englischen: *Domain Specific Language*) bekannt, sind spezialisierte Programmiersprachen, die für klar definierte Anwendungsbereiche entwickelt werden. Sie unterscheiden sich von allgemeinen Programmiersprachen insbesondere durch ihre maßgeschneiderte Ausrichtung auf die spezifischen Anforderungen einer bestimmten Domäne und sind oft weniger komplex und klarer gestaltet. In der Praxis werden DSLs häufig von Experten eines Gebiets entwickelt, um spezielle Anforderungen effektiv zu adressieren [8].

Im Rahmen dieses Projekts wurden die DSLs vom Leiter des Instituts für Systemsoftware, Prof. Mössenböck, bereitgestellt. Diese Arbeit basiert auf diesen DSLs, der zugehörigen attributierten Grammatik (ATG) und drei neuen Klassen, die in direktem Zusammenhang mit der ATG stehen. Wie bereits in der Einleitung/Motivation angedeutet, ist das Hauptziel dieser Arbeit die Realisierung der Erweiterung für die vier Studiengänge: Bachelorstudium Informatik, Masterstudium Computer-Science, Bachelorstudium Artificial Intelligence und Masterstudium Artificial Intelligence. Im weiteren Verlauf dieses Kapitels wird speziell das Masterstudium Artificial Intelligence vorgestellt, dessen Curriculum komplexer ist als die Curricula der anderen Studiengänge und somit einen guten Einblick in die Modellierungsmöglichkeiten der DSL und in die Struktur dieses Studiengangs bietet.

2.4.1 DSL und Struktur des Masterstudiums Artificial Intelligence

Nachfolgend wird die domänenspezifische Sprache (DSL) präsentiert, die für das Masterstudium Artificial Intelligence verwendet wird. Diese Listings bieten einen Einblick in die Kursstrukturen, ihre Besonderheiten und die Anforderungen des Studiengangs.

```
1 // Raster993.eg:
2 curriculum 066 993 // Masterstudium Artificial Intelligence (Update: 2023-08-25)
3 base studies 033 536 // Bachelorstudium Artificial Intelligence
4
5 subject "Machine Learning and Perception"
6 subject "Seminar and Practical Training"
7 subject "AI and Society"
```

Dieses Listing zeigt die ersten sieben Zeilen des Studienplans für das Masterstudium Artificial Intelligence mit der Studienkennzahl 066 993, welches das Bachelorstudium

Artificial Intelligence (Studienkennzahl 033 536) voraussetzt und führt die ersten drei Fächer dieses Studiengangs auf. Innerhalb dieser DSL dienen Begriffe wie `curriculum`, `base studies` und `subject` als Schlüsselwörter, um bestimmte Strukturen oder Anweisungen im Studienplan zu definieren.

Ein weiteres Schlüsselwort, das in den nächsten Listings auftauchen wird, ist `do`. Es spezifiziert Voraussetzungen (*Requirements*), die ein bestimmtes `subject` erfüllen muss. Zum Beispiel wäre die Kombination von `subject "AI and Society"` mit `do "AI and Society"` in der von uns definierten DSL redundant. Der Grund dafür ist, dass `subject "AI and Society"` bereits impliziert, dass alle Kurse, die unter dieses Fachgebiet fallen, absolviert werden müssen. Das Schlüsselwort `do` bekommt erst dann Bedeutung, wenn man eine spezifische Auswahl aus der Gesamtheit der Kurse eines Fachgebiets treffen oder bestimmte Einschränkungen (*Constraints*) festlegen möchte, wie es in unserem nächsten Listing gezeigt wird:

```
7  subject "Elective Tracks"
8     hint "If the elective track is 'Symbolic AI and Mathematical Foundations'
          choose also a focus area"
9     choose "Elective Tracks" $track
10    choose "Focus Area" $focus from ("Symbolic AI", "Mathematical Foundations")
11    if ($track == "Symbolic AI and Mathematical Foundations") {
12       do 21 ects from $track
13       do at least 13.5 ects from $focus
14       if ($focus == "Symbolic AI") do (
15          993TASMKPLV22,  // VL Knowledge Representation and Learning
16          993TASMKPLU22,  // UE Knowledge Representation and Learning
17          993TASMPRAV22,  // VL Planning and Reasoning in Artificial Intelligence
18          993TASMPRAU22   // UE Planning and Reasoning in Artificial Intelligence
19       )
20    } else
21       do $track
```

In diesem Teil der DSL kommen wir zur komplexesten Struktur, die zugleich den Hauptgrund für die Erweiterung der DSL darstellt, sowie den Anstoß für dieses Bachelorprojekt liefert. Die derzeitige DSL kann die hier dargestellten verschachtelten *Constraints* nicht abbilden, da ihre ATG nicht mächtig genug ist, um solch komplexe Strukturen zu beschreiben.

Das oben gezeigte Listing illustriert das `subject "Elective Tracks"` zusammen mit seinen bedingten *Constraints*. Der `hint` liefert einen Hinweis, der in der Benutzeroberfläche des Tools für dieses Fach aufscheinen soll. Dann folgt eine spezielle bedingte Mehrfach-

Auswahl, die durch die `choose`-Schlüsselwörter repräsentiert wird. Die Besonderheit besteht darin, dass die Auswahl für den Fokusbereich `"Focus Area"` von der vorherigen Entscheidung in der Auswahl `"Elective Tracks"` abhängt. Wenn ein Student `"Symbolic AI and Mathematical Foundations"` wählt, müssen die spezifischen Bedingungen, die innerhalb des entsprechenden `if`-Statements aufgeführt sind, erfüllt werden. Wird eine andere Option gewählt, tritt der `else`-Zweig in Kraft, bei dem sämtliche Kurse des gewählten `tracks` absolviert werden müssen. Innerhalb des genannten `if`-Statements sind zudem zwei `do`-Anweisungen und ein weiteres `if`-Statement vorhanden, welche die Auswahl des `"Elective Tracks"` weiter präzisieren.

Die Sprache generell ist klar strukturiert und für Personen, selbst ohne Fachkenntnisse oder Programmiererfahrung, relativ gut verständlich. Zum Beispiel bedeutet `do 21 ects from $track` in Kombination mit der getroffenen Auswahl sinngemäß auf Deutsch: „Mache 21 ECTS aus dem gewählten `"Elective Track"`“. Im darauffolgenden Befehl erweitert `at least` die Bedeutung, sodass auch mehr als die spezifizierten ECTS in dem gegebenen Kontext gemacht werden können. Bei einem weiteren `if`-Statement in Zeile 14 geht es hierbei um den gewählten Fokus. Wenn dieser auf `"Symbolic AI"` fällt, sind aus dem Pool von mindestens 13.5 ECTS die vier spezifischen Pflichtkurse, repräsentiert durch ihre Klassencodes, zu absolvieren.

Im Zuge der Darstellung unserer DSL gibt es eine zugrundeliegende Struktur, die im Hintergrund abgebildet wird. Wir werden uns diese Struktur im Detail, insbesondere den Abschnitt von Zeile 11 bis 21, auf der folgenden Seite genauer anschauen. Diese Struktur wurde in einer ähnlichen Form als Orientierungshilfe für das Projekt zur Verfügung gestellt.

Innerhalb dieser Struktur sind insbesondere `do`- und `if`-Anweisungen in orange als `Requirements` dargestellt. Jedes `Requirement` besitzt einen zugehörigen abstrakten Syntaxbaum (*abstract syntax tree*, AST) und verweist auf weitere `Requirements`, die sich an der zuvor dargestellten DSL-Struktur orientieren. Bei der Erstellung des ASTs kommen spezifische Knoten, auch *Nodes* genannt, zum Einsatz. Innere Knoten dieser Struktur repräsentieren in der Regel Operatoren, wie beispielsweise „`==`“ oder „`+`“. Die Blätter des ASTs, auch als *Leafs* bezeichnet, enthalten hingegen Operanden: Sie können einen `String`, einen Klassencode oder eine spezifische ID, die mit einem $-Symbol beginnt, beinhalten. Diese ID verweist auf eine Auswahl, die von einem Studenten getroffen wurde. Die Implementierung und weitere Aspekte dieser Struktur werden im Kapitel Implementierung behandelt.

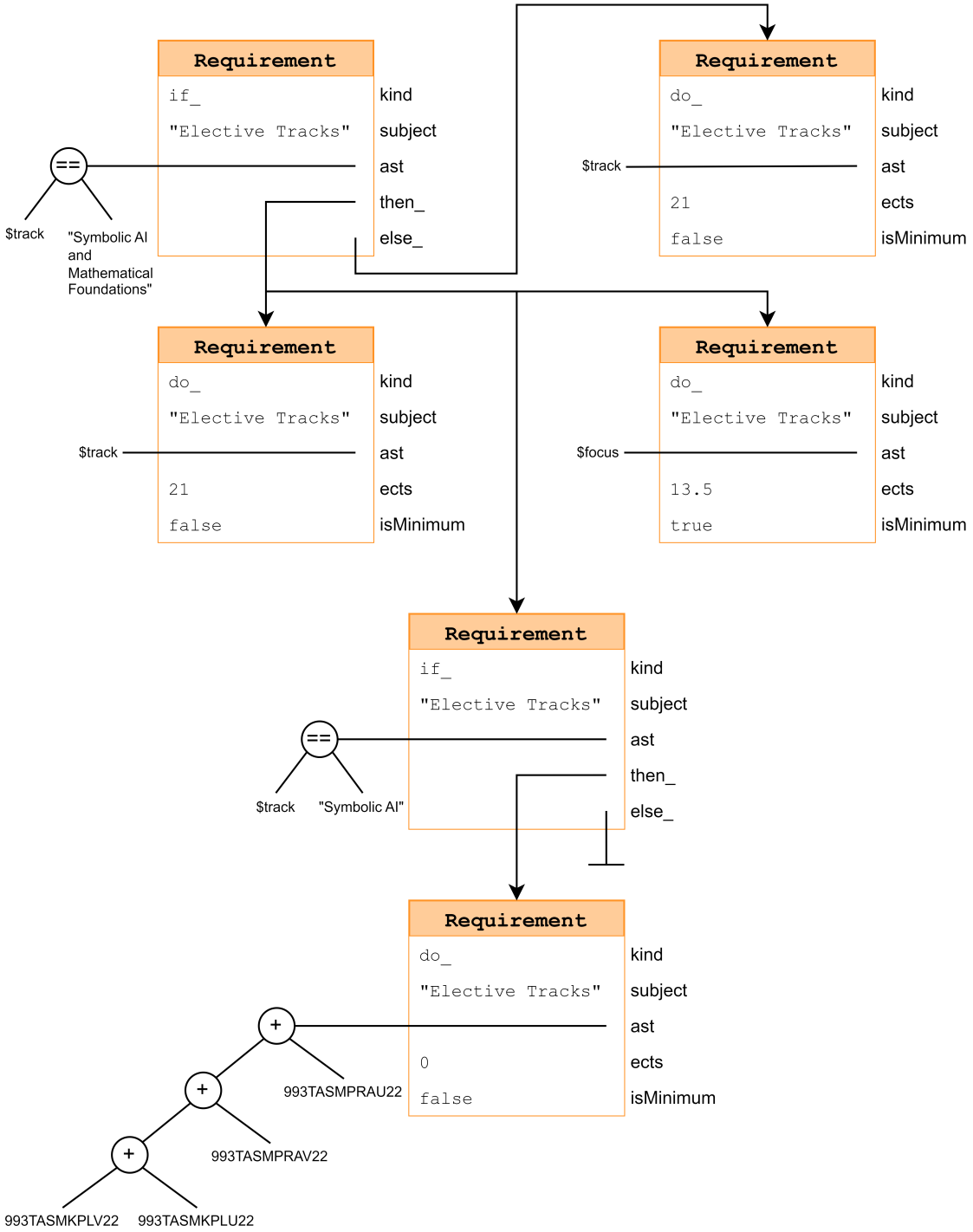


Abbildung 2.5: DSL Struktur des `"Elective Tracks"`

Ein weiterer Punkt im Studienplan ist die `"Area of Specialization"`, welche im nachstehenden Listing zu sehen ist:

```
22 subject "Area of Specialization"
23    hint "choose the number of ECTS to complement your elective track to 27 ECTS"
24    choose "chosen ECTS" #specialization from (10.5, 9, 6, 15)
25    do #specialization ects
```

In diesem Abschnitt wird dem Studierenden die Auswahlmöglichkeit gegeben, um die Anzahl der ECTS des Wahlfachs anzupassen, um insgesamt 27 ECTS zu errreichen.

Zum Abschluss des Masterstudiums Artificial Intelligence gibt es noch das `"Master's Thesis Seminar"` und einen Bereich für freie Wahlkurse, wie im folgenden Listing dargestellt:

```
26 subject "Master's Thesis Seminar"
27 subject "Free Electives"
28    do 12 ects from *
```

Dies zeigt, dass Studierende innerhalb der `"Free Electives"` insgesamt 12 ECTS aus beliebigen Kursen absolvieren müssen, um ihre Studienanforderungen zu erfüllen.

Abschließend werden globale Beschränkungen und Äquivalenzen im Studienplan definiert, wie im folgenden Abschnitt aufgeführt:

```
29 check
30    ects($track) + #specialization = 27
31    otherwise "Elective Tracks and Area of Specialization must total 27 ECTS"
32 equivalences  // 993
33 //--- from 2019W to 2020W
34 993MLPEEAIK19 -> (993MLPEEAIV20, 993MLPEEAIU20) // Explainable AI 2KV -> 1VO+1UE
35 921PECOCOVK13 -> (921PECOCOVV20, 921PECOCOVU20) // Computer Vision 3KV -> 2VO+1UE
36 ...
```

Die unter `check` spezifizierte globale Bedingung stellt sicher, dass die Kombination aus dem gewählten `track` und der `"Area of Specialization"` insgesamt 27 ECTS ergibt. Die Äquivalenzen zeigen, welche Kurse aus älteren Curricula mit welchen Kursen des neuen Curriculums äquivalent (und somit anrechenbar) sind.

Nachdem wir die DSL und ihre Struktur analysiert haben, werden wir uns nun den praktischen Aspekten zuwenden und beleuchten dazu die Schritte und Überlegungen, die bei der Implementierung dieser Konzepte anstehen.

3 Implementierung

Nachdem die domänenspezifische Sprache (DSL) und ihre Struktur im vorangegangen Kapitel vorgestellt wurde, wenden wir uns nun ihrer technischen Umsetzung zu. Dabei liegt der Fokus nicht nur auf der Implementierung der erweiterten DSL, sondern auch auf Modifikationen des am Institut für Systemsoftware entwickelten Tools, das den automatischen Ausfüllprozess von Prüfungsrastern übernimmt.

In diesem Kapitel fokussieren wir uns zunächst auf die Anpassungen und Erweiterungen der attributierten Grammatik (ATG). Hierbei werden die wichtigsten Produktionen der ATG vorgestellt und es wird gezeigt, wie die DSL durch semantische Aktionen in die Datenstrukturen des Backends übersetzt wird. Parallel dazu wird die Datenstruktur im Backend umgestaltet, um die in der DSL beschriebenen Informationen in klare und strukturierte Datenobjekte zu überführen. Im weiteren Verlauf des Kapitels behandeln wir auch die notwendigen Anpassungen im Frontend, insbesondere in Bezug auf die Angular-Anwendung.

3.1 Anpassung/Erweiterung der attributierten Grammatik (ATG)

Die attributierte Grammatik bildet das Fundament unserer DSL. In dieser Sektion gehen wir auf Produktionen der ATG ein, die aufgrund ihres entscheidenden Beitrags zur Struktur der DSL näher betrachtet werden. Der Schwerpunkt liegt hierbei sowohl auf der Modifikation dieser Produktionen als auch auf den Ausbau der zugehörigen semantischen Aktionen. Zur Verbesserung der Übersichtlichkeit und um die Unterscheidung klarer zu gestalten, werden die nachfolgenden Listings in einem erweiterten Format präsentiert. Dadurch lassen sich die Syntaxelemente, die auf der linken Seite dargestellt werden, leichter von den semantischen Aktionen auf der rechten Seite unterscheiden.

Im Folgenden wird die Root-Produktion `Specification` der erweiterten ATG vorgestellt, die sowohl die grundlegenden Strukturelemente als auch die semantischen Aktionen aufzeigt:

```
Specification =
    "curriculum" CodePair<.out IntPair skz.>        (. Curriculum curriculum = curricula.stream().filter(c ->
                                                        c.getSkz() == skz.getSecond()).findAny().orElse(null); .)
                                                    (. if (curriculum == null) SemErr("Unknown SKZ: " + t.val); return; .)
                                                    (. List<IntPair> bases = List.of(); .)
    [ "base" "studies" CodePairList<.out bases.> ]
                                                    (. StudyMap successors = new StudyMap(); .)
    [ "successor" "studies" SuccessorMap<.out successors.> ]
                                                    (. specification = new Specification(utils.toPairList(bases),
                                                        utils.toMap(successors), skz.getFirst(), curriculum); .)
                                                    (. utils.init(); .)
    {
        SubjectHeader<out SubjectHeader header>     (. specification.getEntries().add(header); .)
        |
        SubjectSpec<out SubjectSpec subject>        (. specification.getEntries().add(subject); .)
    }
    {
        GlobalConstraint<out GlobalConstraint constraint>
                                                    (. specification.getGlobalConstraints().add(constraint); .)
    }
    [
        "equivalences"                              (. List<Equivalence> equivalences = new ArrayList<>(); .)
        {
            Equivalence<out Equivalence equivalence>
                                                    (. equivalences.add(equivalence); .)
        }                                           (. specification.getEquivalences().addAll(equivalences); .)
    ]
.
```

Die `Specification`-Produktion illustriert als Einstiegspunkt die Kern-Konstruktion der gesamten DSL sowie Details der Implementierung. Jede semantische Aktion, gekennzeichnet durch `(. ... .)`, spielt an Ort und Stelle eine entscheidende Rolle und gewährleistet, dass die in der DSL bereitgestellten Informationen korrekt verarbeitet und in entsprechende Datenobjekte umgewandelt werden.

Nach der Hauptproduktion `Specification` wollen wir nun weitere zentrale Produktionen betrachten. Eines der bedeutendsten ist die Produktion `SubjectSpec`:

```
SubjectSpec<out SubjectSpec subject> =              (. String alias = null, hint = null;
                                                        List<SelectionSpec> selections = null;
                                                        List<Requirement> requirements = null; .)
    "subject" Subject<out curSubject>
    [ Alias<out alias> ]
    [ Hint<out hint> ]
    {
        Selection<out SelectionSpec selection>      (. if (selections == null) selections = new ArrayList<SelectionSpec>();
```

```
9                                                   selections.add(selection); .)
10      }
11      (
12          Require<out requirements>               (. dump(requirements); .)
13          |                                       (. Requirement requirement = new Requirement(do_);
14                                                      requirement.subject = curSubject;
15                                                      requirements = new ArrayList<Requirement>(1);
16                                                      requirements.add(requirement);
17                                                      requirement.ast = new Node(string, curSubject); .)
18      )                                           (. subject = new SubjectSpec(curSubject, alias, hint, selections,
19                                                      requirements); .)
20  .
```

Die `SubjectSpec`-Produktion modelliert die Spezifikationen von Studienfächern und integriert verschiedene Attribute wie Alias, Hinweise (`Hint`), Auswahlmöglichkeiten und Anforderungen. Mit der Einführung der erweiterten ATG wurde insbesondere die Mehrfach-Auswahlmöglichkeit (Zeile 7-10) hinzugefügt, die eine signifikante Erweiterung für die Spezifikation von Studienfächern darstellt. Ein besonderes Augenmerk gilt aber vor allem auch den Anforderungen: Bei Verwendung des Schlüsselworts `do` in der DSL tritt dabei die `Require`-Produktion in Kraft. Andernfalls wird standardmäßig ein `do`-Requirement generiert. Dies spiegelt auch die im Abschnitt 2.4.1 erläuterte Bedeutung von `do` wider. Ausgehend von dieser Produktion erlebt das Backend eine signifikante Veränderung in seiner Datenstruktur. Ein entsprechendes Diagramm hierzu wird später in Abschnitt 3.2 vorgestellt.

Ein weiteres Kernelement unserer ATG ist die Produktion `Require`. Diese Produktion modelliert die unterschiedlichen Anforderungen, die an einen Studiengang oder einem Fachgebiet gestellt werden können. Sie zeichnet sich durch ihre Vielseitigkeit und Anpassungsfähigkeit aus, die durch verschiedene Schlüsselwörter und semantische Aktionen zum Ausdruck kommt:

```
1   Require<.out List<Requirement> list.> =         (. List<Requirement> lst; .)
2                                                   (. Requirement req = new Requirement(do_); req.subject = curSubject;
3                                                       list = new ArrayList<Requirement>(1); list.add(req); .)
4       (
5           "do"
6           (
7               Courses<out req.ast>
8               |
9               [
10                  "at" "least"                    (. req.isMinimum = true; .)
11              ]
```

```
            (
                number                              (. req.ects = convertEcts(t.val); .)
                |
                NumID<out req.ectsID>
            )
            "ects"
            (
                "from" Courses<out req.ast>
                |                                   (. req.ast = new Node(string, curSubject); .)
            )
        )
        [ "otherwise" String<out req.msg> ]
        |
        "if"                                        (. req.kind = if_; .)
        "("
            Condition<out req.ast>                  (. if (!isBoolean(req.ast)) SemErr("boolean condition expected"); .)
        ")"
        Require<out req.then_>
        [ "else" Require<out req.else_> ]
        | "module" Courses<out req.ast>             (. req.kind = module_; .)
        |
        "{"                                         (. list.clear(); .)
            {
                Require<out lst>                    (. list.addAll(lst); .)
            }
        "}"
    )
.
```

Die Produktion `Require` bietet mehrere Pfade, die es ermöglichen, unterschiedliche Anforderungsarten zu beschreiben, beispielsweise basierend auf ECTS oder Bedingungen. Die Verwendung von semantischen Aktionen an strategisch gut gewählten Stellen sorgt dafür, dass die relevanten Daten aus der DSL korrekt erfasst und in Backend-Strukturen umgewandelt werden. Besonders bemerkenswert ist die Möglichkeit, verschachtelte Anforderungen rekursiv mittels `{ Require }` zu definieren, was zusätzliche Flexibilität bei der Beschreibung komplexer Studienvorgaben bietet, wie im Listing `subject "Elective Tracks"` des letzten Kapitels (siehe 2.4.1) bereits gezeigt wurde.

Parallel zu den beschriebenen Anforderungen stellt die Produktion `Selection` ein weiteres entscheidendes Element unserer DSL dar, die vor allem für die Realisierung der Master-Studiengänge von hoher Bedeutung ist.

```
Selection<out SelectionSpec selection> =            (. selection = null;
                                                       String caption, subjectID, numID, s;
                                                       List<String> subjectsList = new ArrayList<>();
```

```
                                                    List<Integer> numbersList = new ArrayList<>(); .)
    "choose" Caption<out caption>
    (
        SubjectID<out subjectID>
        [
            "from"
            "("
                Subject<out s>                      (. subjectsList.add(s); .)
                {
                    "," Subject<out s>              (. subjectsList.add(s); .)
                }
            ")"
        ]                                           (. selection = new SelectionSpec(caption, subjectID, null,
                                                        subjectsList, null); .)
        |
        NumID<out numID>
        "from"
        "("
            number                                  (. numbersList.add(convertEcts(t.val)); .)
            {
                "," number                          (. numbersList.add(convertEcts(t.val)); .)
            }
        ")"                                         (. selection = new SelectionSpec(caption, null, numID, null,
                                                        numbersList); .)
    )
.
```

Die Selection-Produktion zeigt, wie Auswahlmöglichkeiten in unserer DSL modelliert werden. Sie erlaubt es, eine Auswahl basierend auf bestimmten Fächern oder ECTS zu definieren. Durch diese Funktion kann das Tool präziser auf die individuellen Vorstellungen des Benutzers abgestimmt werden.

Die hier vorgestellten Produktionen stellen nur einen Ausschnitt der Gesamtergänzungen dar, die im Zuge der Anpassung und Erweiterung der ATG vorgenommen wurden. Eine umfassende Übersicht findet sich in der Datei `Specification.atg` die im Abgabeordner in den Projektdateien zu finden ist. Aufgrund der Neugestaltung und Erweiterung der DSL muss auch die Datenstruktur im Backend entsprechend angepasst werden. Genau diese Anpassung wird im folgenden Abschnitt behandelt.

3.2 Backend-Strukturanpassung

Die Erweiterung der domänenspezifischen Sprache (DSL) erforderte entsprechende Anpassungen und Aktualisierungen in der attributierten Grammatik (ATG), wie im letzten Abschnitt gezeigt. Dies zieht notwendigerweise eine Überarbeitung der Backend-Datenstruktur nach sich. Der Grund hierfür liegt in der Funktionsweise des von Coco/R generierten Parsers und Scanners, die dafür zuständig sind, den Text aus den DSL-Dateien, welche Studienplan-Spezifikationen enthalten, zu interpretieren und inhaltlich zu verarbeiten. Während dieses Vorgangs werden spezifische Datenobjekte identifiziert, die im Backend konsistent und geordnet gespeichert werden müssen.

Die in Abbildung 3.1 dargestellte Datenstruktur verdeutlicht die notwendigen Anpassungen im Backend. Die Klasse `Specification`, als direkter Übersetzer der DSL konzipiert, verknüpft sich dabei mit `GlobalConstraint` sowie `Equivalence` und beinhaltet zudem eine Liste von Einträgen (`entries`) für die Spezifikation der Studienfächer. Alle im Diagramm gezeigten Klassen, bis auf `Specification`, `Equivalence` und `SubjectHeader`, wurden grundlegend überarbeitet oder sind neu hinzugekommen. Zu den neuen Klassen zählen `Requirement` und `Node`, die beide durch eine rekursive Struktur geprägt sind. Damit können Anforderungen und der zugehörige abstrakte Syntaxbaum (AST) tief verschachtelt werden, wie beispielsweise bereits für das `subject "Elective Tracks"` im Abschnitt 2.4.1 erörtert wurde.

In der angepassten Datenstruktur sind die Knoten (`Nodes`) für die Erstellung eines AST verantwortlich. Diese repräsentieren unterschiedliche Typen, wie zum Beispiel `string`, `subjectID`, `number` und `class_`, die jeweils die *Leafs* des AST darstellen, wie auch bereits in Abbildung 2.5 gezeigt. Namen von Fächern oder darunterliegenden Bereichen werden mittels `string` bestimmt, wohingegen `subjectID` für mittels `choose` gewählte Fächer und `class_` für Klassencodes in der DSL stehen. Dort sieht man auch arithmetische Operatoren, dargestellt als innere Knoten, wie `plus (+)`, die in den vier Studienplänen zusammen mit der Operation `minus (-)` verwendet werden, um Kursmengen oder eine Menge an Klassencodes zu definieren. Die Operatoren in der Kombination `plus (+)` und `eql (==)` kommen hauptsächlich im Kontext der Bestimmung von ECTS zum Einsatz, wie z.B. beim `GlobalConstraint` im Abschnitt 2.4.1 gezeigt. Des Weiteren wird `eql (==)` für bedingte Abfragen verwendet, wie im Listing für das `subject "Elective Tracks"` illustriert. Diese spezifischen Festlegungen ermöglichen eine präzise Interpretation unserer DSL.

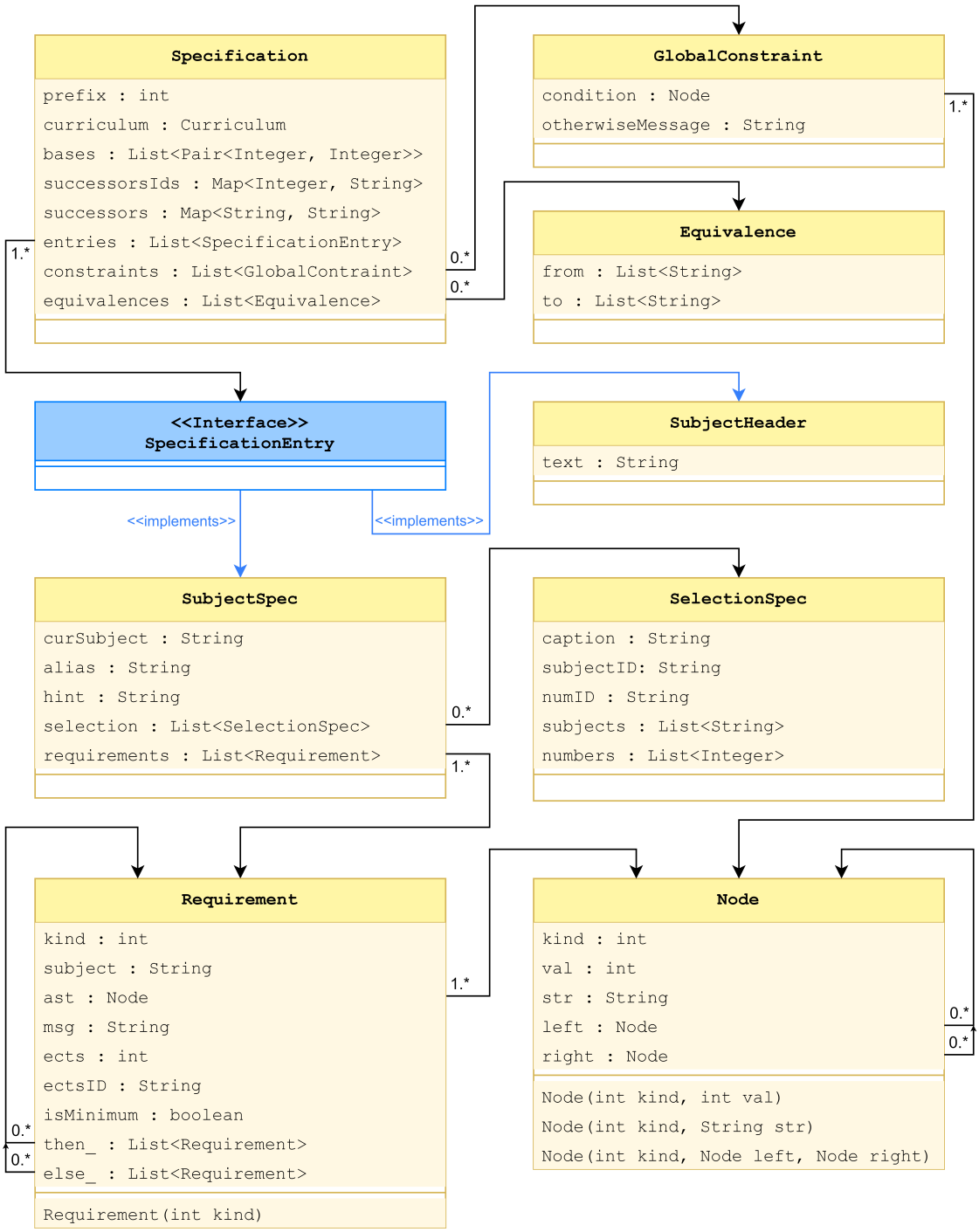


Abbildung 3.1: Überarbeitete Datenstruktur auf Basis der erweiterten ATG

Durch die eingehenden Erläuterungen zur aktualisierten ATG und der modifizierten DSL sollte nun ein klareres Gesamtbild unseres Programms entstehen. Mit der vorgestellten Überarbeitung des Backends ist die Grundlage gelegt, um die angepasste DSL effizient zu interpretieren und im Zusammenspiel mit dem Angular-Frontend umzusetzen.

3.3 Anpassungen im Angular-Frontend

In diesem Abschnitt konzentrieren wir uns auf das zugrundeliegende Konzept und die Modifikationen im Angular-Frontend, die sich aufgrund der ATG-Erweiterung und der Backend-Umstrukturierung ergeben. Während in Abschnitt 2.3 eine Einführung in die Angular-Architektur für dieses Tool gegeben wurde, beschäftigen wir uns hier mit den Maßnahmen zur Überarbeitung und Aktualisierung des Frontends.

Zu Beginn wurde die Backend-Datenstruktur im Angular-Frontend abgebildet bzw. entsprechend erweitert. Im Anschluss daran waren Anpassungen an über 30 verschiedenen Stellen im Code erforderlich. Diese Modifikationen dienten nicht nur der Behebung diverser Fehler, sondern auch der Schaffung einer stabilen Grundlage für die Datenverarbeitungsklassen, wie in Abschnitt 2.3 beschrieben. Mit dieser Basis konnte schließlich die Evaluierung und Implementierung der neuen Datenstrukturlogik realisiert werden. Im Zuge dessen wurden auch die `RequirementKind`- und `NodeKind`-Definitionen eingeführt, um die Logik der Datenstruktur effizient zu evaluieren und zu implementieren.

3.3.1 Definitionen von `RequirementKind`

Integer	Kind	Beschreibung	Beispiel aus DSL
0	DO	Zu erfüllende Anforderung	`do 21 ects from $track`
1	IF	Abfragebedingung	`if ($focus == "Symbolic AI")`

Tabelle 3.1: Übersicht über die `RequirementKind`-Definitionen

Tabelle 3.1 veranschaulicht die neu eingeführten `RequirementKind`-Definitionen im Angular-Frontend. Bei den aus dem Backend gelieferten Datenobjekten wird generell der `kind` als Integer-Wert übermittelt. Mit den hier vorgestellten Definitionen wird diesem Integer-Wert eine konkrete Bedeutung zugeordnet, wodurch der Kontext und die Absicht des jeweiligen Datenobjekts klarer hervorgehoben wird.

3.3.2 Definitionen von `NodeKind`

Integer	Kind	Beschreibung	Beispiel aus DSL
0	NUMBER	Integer Wert	`3, 7.5, 27`
1	STRING	Name eines Fachbereichs	`"Elective Tracks"`
2	SUBJECT_ID	ID für gewählten Bereich	`$track`
3	NUM_ID	ID für gewählten ECTS	`#specialization`
4	CLASS	Klassencode eines Kurses	`993TASMKPLV22`
5	ECTS	Anzahl der ECTS	`21, 13.5, 27`
6	PLUS	Plus Operator	`+`
7	MINUS	Minus Operator	`-`
8	AND	And Operator	`and`
9	OR	Or Operator	`or`
10	EQL	Equal Operator	`==, =`
11	NEQ	Not-Equal Operator	`!=`
12	LSS	Less Operator	`<`
13	LEQ	Less or Equal Operator	`<=`
14	GTR	Greater Operator	`>`
15	GEQ	Greater or Equal Operator	`>=`

Tabelle 3.2: Übersicht über die `NodeKind`-Definitionen

Tabelle 3.2 bietet eine detaillierte Übersicht über die `NodeKind`-Definitionen, die im Rahmen des Angular-Frontends genutzt wurden bei der Implementierung. Diese Typeneinteilung ermöglicht es, verschiedene Datenobjekte und Operatoren im System effizient zu identifizieren und zu verarbeiten. Ähnlich wie bei den `RequirementKind`-Definitionen wird jeder `NodeKind` durch einen spezifischen Integer-Wert repräsentiert. Diese Werte sind nicht nur essentiell für die interne Logik und Datenverarbeitung, sondern tragen auch dazu bei, den Code lesbarer und wartbarer zu gestalten. Die in der Tabelle gezeigten Beispiele aus der DSL können im umfassenderen Kontext in Abschnitt 2.4.1 nachgelesen werden. Durch die Einführung dieser Definitionen kann man nun schnell den Typ und den Zweck eines bestimmten `Node` in der Datenstruktur erfassen, um die entsprechende Logik dahinter zu implementieren.

3.3.3 Logik-Implementierung

Nach der Definition und Modellierung der Anforderungstypen, repräsentiert durch die `RequirementKind`- und `NodeKind`-Enumerationen, besteht der nächste Schritt darin, die eigentliche Logik zur Verarbeitung dieser Anforderungen im Angular-Frontend zu implementieren. Diese Implementierung ist essenziell, um sicherzustellen, dass die im Backend definierten Datenstrukturen und Anforderungen korrekt im Frontend interpretiert und visualisiert werden.

Ein zentraler Bestandteil dieser Logik ist die Methode `checkAndProcessRequirements`. Sie durchläuft alle Anforderungen, die in einem bestimmten Studienplan (repräsentiert durch `this.spec.requirements`) definiert sind, und verarbeitet sie je nach ihrem spezifischen Typ. Der folgende Code gibt einen Überblick über diesen Prozess:

```
private checkAndProcessRequirements() {
  for (const req of this.spec.requirements || []) {
    switch (req.kind) {
      case RequirementKind.DO:
        this.processDoRequirement(req);
        break;
      case RequirementKind.IF:
        this.processIfRequirement(req);
        break;
    }
  }
}
```

Wie im obigen Codeausschnitt ersichtlich, werden die Anforderungen basierend auf ihrem `RequirementKind` (entweder `DO` oder `IF`) verarbeitet. Spezifische Methoden wie `processDoRequirement` und `processIfRequirement` werden aufgerufen, welche lediglich den Beginn einer komplexen Kette von aufeinanderfolgenden Methodenaufrufen repräsentieren. Einige dieser Methoden sind rekursiv, ähnlich wie in Abschnitt 3.2 erläutert. Die ausgearbeitete Logik stellt sicher, dass alle visuellen Darstellungen im Frontend korrekt zur Anzeige gebracht werden. Genauere Details dazu können in der Klasse `SubjectData` im Projektordner der Abgabe eingesehen werden.

Nachdem die zentrale Verarbeitungslogik weitgehend in der Klasse `SubjectData` umgesetzt wurde, traten weitere Herausforderungen im Zuge der Implementierung des Masterstudiums

Artificial Intelligence auf. Eine solche Herausforderung war die Integration einer Mehrfach-Auswahl, wie im Listing des `subject "Elective Tracks"` im Abschnitt 2.4.1 illustriert, bei der auch die `SubjectNode`-Klasse sowie verschiedene Angular-Komponenten (siehe Abbildung 2.4) modifiziert werden mussten. Entscheidend bei der Implementierung dabei war die Einführung des `DropdownMenu`, welches folgendermaßen definiert wurde:

```
export interface DropdownMenu {
  ref: number;
  caption: string;
  subjectID?: string;
  numID?: string;
  selectionOptions: SelectionOption[];
  selectedOption: SelectionOption | null;
  showDropdown: boolean;
}

export interface SelectionOption {
  ref: number;
  optionName: string;
}
```

Das `DropdownMenu` enthält neben den nötigen Attributen auch eine Liste von Auswahlmöglichkeiten, repräsentiert durch `selectionOptions`, sowie die aktuell getroffene Auswahl des Benutzers in `selectedOption`. Das `showDropdown`-Attribut steuert dabei die Sichtbarkeit des Dropdown-Menüs. Ein anspruchsvoller Punkt dabei war die Implementierung der dynamischen Anzeige der Dropdown-Menüs basierend auf den in der DSL festgelegten Bedingungen, von dessen Abhängigkeit das spezifische Dropdown-Menü ein- oder ausgeblendet wird. Dies ermöglicht eine flexible Benutzeroberfläche, die nur relevante Auswahlmöglichkeiten anzeigt und je nach Situation weitere Auswahlen zur Verfügung stellt.

Eine weitere Herausforderung stellte die Implementierung des `GlobalConstraintNode` dar, die eine komplette Überarbeitung erforderte. Wie in Abbildung 3.1 demonstriert, besteht ein `GlobalConstraint` aus einer `condition` und einer `otherwiseMessage`. Diese Struktur wurde im Angular-Frontend entsprechend abgebildet:

```
export interface GlobalConstraint {
  condition: Node;
  otherwiseMessage: string;
}
```

So wie ein `Requirement` durch einen AST in seinem `ast`-Attribut repräsentiert wird, wird die `condition` im `GlobalConstraint` ebenfalls durch einen AST dargestellt. Dieser AST muss evaluiert werden, um zu bestimmen, ob der Text in `otherwiseMessage` angezeigt werden soll oder nicht. Dazu wurde folgende Methode konstruiert:

```
1  private evaluateAstCondition(condition: ConditionNode): boolean {
2    if (!condition) return false;
3
4    const leftValue = condition.left ? this.evaluateAstForEcts(condition.left) :
         undefined;
5    const rightValue = condition.right ? this.evaluateAstForEcts(condition.right) :
         undefined;
6
7    if (typeof leftValue === "boolean" || leftValue === undefined || typeof
         rightValue === "boolean" || rightValue === undefined) {
8      return false;
9    }
10
11   switch (condition.kind) {
12     case NodeKind.EQL:
13       return leftValue === rightValue;
14     case NodeKind.NEQ:
15       return leftValue !== rightValue;
16     case NodeKind.LSS:
17       return leftValue < rightValue;
18     case NodeKind.LEQ:
19       return leftValue <= rightValue;
20     case NodeKind.GTR:
21       return leftValue > rightValue;
22     case NodeKind.GEQ:
23       return leftValue >= rightValue;
24     default:
25       console.warn(‘Unexpected AST kind: ${condition.kind}‘);
26       return false;
27   }
28 }
```

Die oben gezeigte `evaluateAstCondition`-Methode dient dazu, den im `GlobalConstraint` definierten AST zu evaluieren. Der `condition`-Knoten, obwohl vom Datentyp `Node` in der allgemeinen Definition, wird hier im Parameter als `ConditionNode` repräsentiert. Die Methode ruft in der Zeile vier und fünf `evaluateAstForEcts` auf, um die entsprechenden ECTS-Werte für den linken und rechten Kindknoten des übergebenen `ConditionNode` zu ermitteln. Falls in deren Kindknoten weitere Ausdrücke wie die Operatoren PLUS oder

MINUS vorkommen, wird `evaluateAstForEcts` rekursiv aufgerufen, um deren Werte zu berechnen.

Ein anschauliches Beispiel für die Anwendung dieser Methode findet sich in der DSL-Spezifikation, wie im Abschnitt 2.4.1 (Zeile 30 im Listing) gezeigt. Dort enthält ein `GlobalConstraint` die Bedingung `ects($track) + #specialization = 27`. Hierbei entspricht das Gleichheitszeichen (`=`) dem `NodeKind.EQL`. Die Methode `evaluateAstForEcts` ermittelt die Anzahl der geleisteten ECTS für den spezifizierten Track, die direkt aus dem zugehörigen `SubjectNode` extrahiert werden (in diesem Kontext, dem `subject "Elective Tracks"`). Gleichzeitig werden auch die absolvierten ECTS des `subject "Area of Specialization"` berücksichtigt. Durch die Methode `evaluateAstCondition` wird dann (in Zeile 13 in diesem Fall) abschließend überprüft, ob die Summe der ECTS aus beiden Bereichen genau 27 beträgt.

Zum Schluss dieses Kapitels kann festgehalten werden, dass die gezeigte Implementierung und die zugrundeliegenden Konzepte lediglich einen Ausschnitt der gesamten technischen Umsetzung darstellen. Dennoch geben die vorgestellten Inhalte einen wertvollen Einblick in die Struktur, den Funktionsumfang und die Herausforderungen des Programms. Ebenso wird deutlich, dass die Kombination aus der erweiterten ATG, der Überarbeitung des Backends und den umfangreichen Anpassungen im Angular-Frontend ein leistungsstarkes System ergeben, das den spezifischen Anforderungen des Projekts in vollem Umfang gerecht wird. Die ausführlichen Details dazu können im Projektordner der Abgabe eingesehen werden.

4 Anleitung

In den vorigen Kapiteln wurde die Implementierung und die zugrunde liegende Architektur des Prüfungsrasters vorgestellt. Nun soll es darum gehen, das Tool aus Anwendersicht zu verstehen. Die folgende Anleitung vermittelt dazu einen kurzen Überblick über die Handhabung des Prüfungsrasters und stellt Hinweise zu ausgewählten Anzeigen und Funktionen bereit. Wie bereits in der Einleitung erwähnt, funktioniert das Prüfungsraster aktuell nur für die Studienrichtungen Bachelorstudium Informatik, Masterstudium Computer Science, Bachelorstudium Artificial Intelligence und Masterstudium Artificial Intelligence.

Zu Beginn wird die Webseite `https://examroster.jku.at` aufgerufen. Anschließend erfolgt der Login über Shibboleth mit Matrikelnummer und Passwort, analog wie beim Anmeldeprozess auf KUSSS, Moodle oder myJKU. Nach dem erfolgreichen Login gelangt man zur Übersichtsseite des Prüfungsrasters, wie in den Abbildungen 4.1 und 4.2 auf den nachfolgenden Seiten dargestellt. Für eine klare und verständliche Erläuterung wurde für diese Abbildungen eine Nummerierung von 1 bis 15 eingeführt. Im Folgenden werden die jeweiligen nummerierten Bereiche detailliert erklärt:

1. Zeigt das aktuelle Studium an. Es besteht zudem die Möglichkeit, ein eventuelles Zweitstudium oder ein zuvor absolviertes Bachelorstudium auszuwählen.
2. Hier können Nutzer das Prüfungsraster entweder als PDF exportieren oder direkt an den Prüfungs- und Anerkennungsservice (PAS) senden.
3. Im linken Bereich werden alle zugeordneten Prüfungen des Prüfungsrasters angezeigt. Der rechte Bereich listet alle nicht zugeordneten, aber erfolgreich absolvierten oder anerkannten Prüfungen auf.
4. Spaltenüberschriften für die in Punkt 9 aufgeführten Kurse oder für die in Punkt 10 aufgeführten Prüfungen.
5. Überschriften zur besseren Unterteilung der Fachgebiete bzw. Kategorien.
6. Ein-/Ausklappbare Fachgebiete bzw. Kategorien des jeweiligen Studienplans.
7. Allgemeine Hinweise zu den jeweiligen Fachgebieten bzw. Kategorien.
8. Spezifische Hinweise oder Warnungen zu den jeweiligen Fachgebieten.

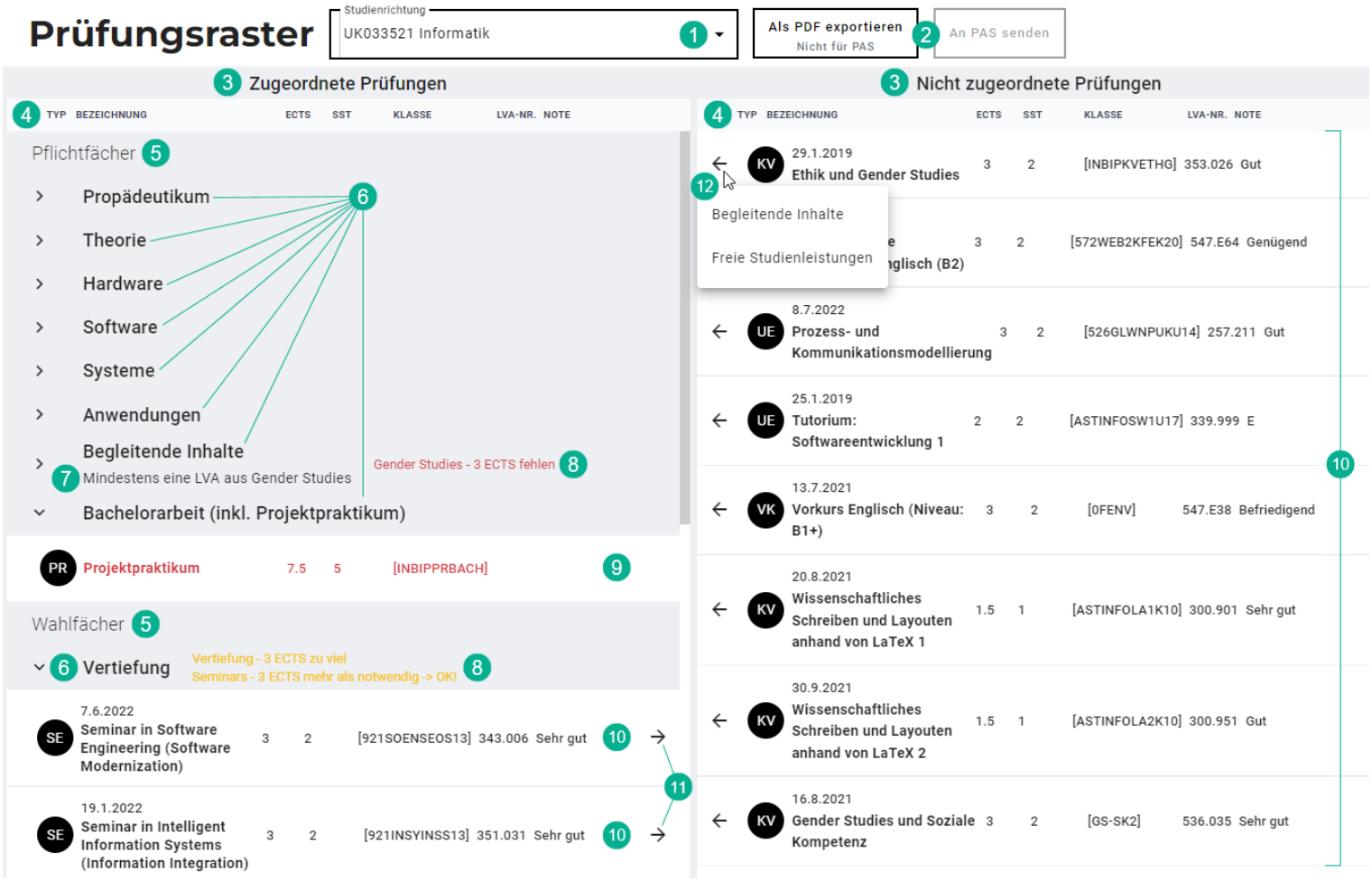


Abbildung 4.1: Überblick der Anzeigen und Funktion am Beispiel des Bachelorstudiums Informatik

9 Nicht absolvierte, obligatorische Kurse sind generell in rot markiert.

10 Bereits absolvierte Kurse oder Prüfungen.

11 Mit diesem Pfeil können zugeordnete Prüfungen in den Bereich der nicht zugeordneten Prüfungen verschoben werden. Dies ist jedoch nur möglich, wenn sie nicht obligatorisch sind.

12 Dieser Pfeil ermöglicht es, nicht zugeordnete Prüfungen einem Fachgebiet zuzuweisen. Wenn eine Prüfung mehreren Fachgebieten zugeordnet werden kann, werden entsprechende Auswahlmöglichkeiten präsentiert.

In Abbildung 4.2 wird anhand des Masterstudiums Artificial Intelligence auf weiterführende wichtige Aspekte eingegangen:

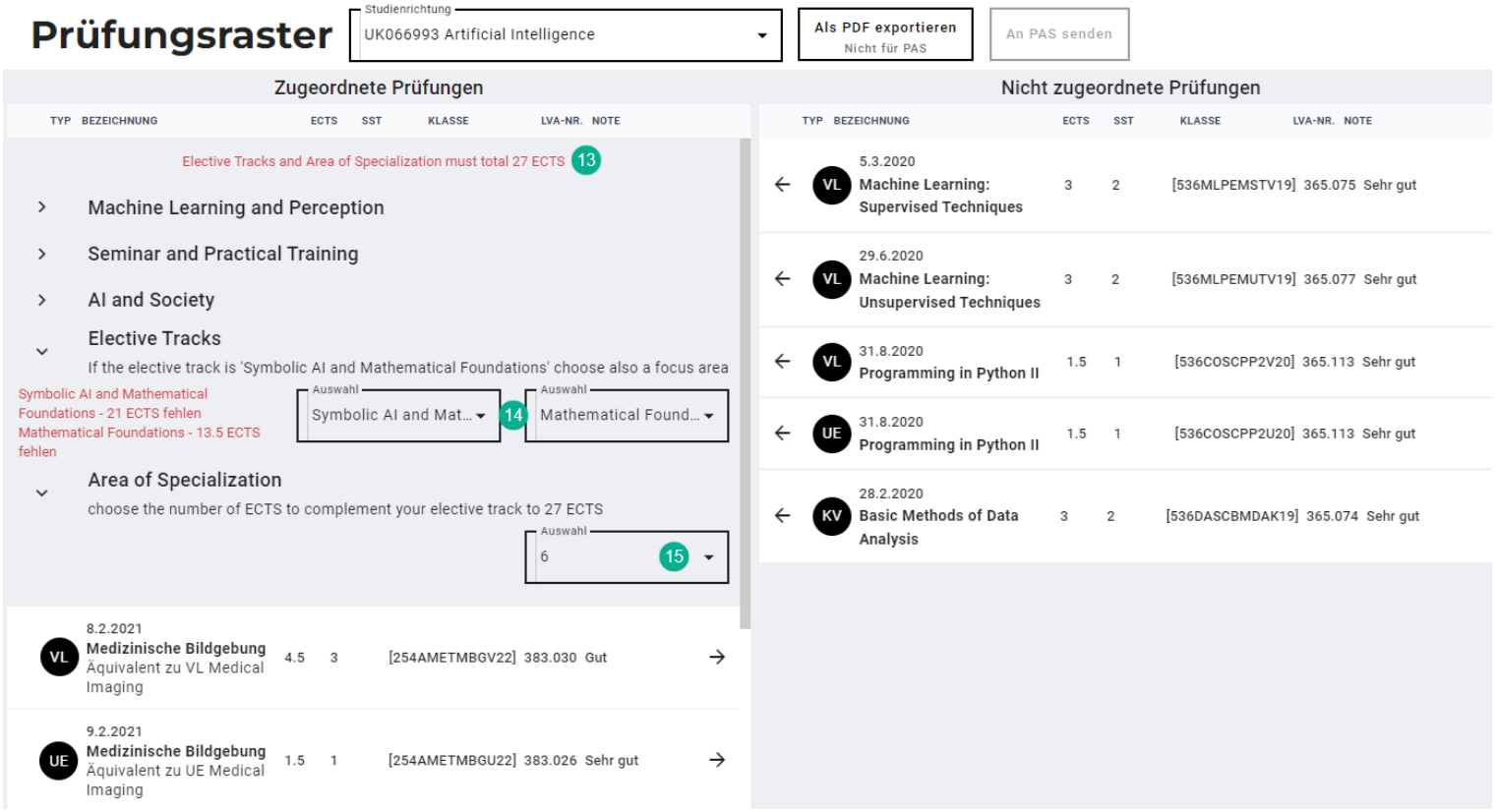


Abbildung 4.2: Überblick der Anzeigen und Funktion am Beispiel des Masterstudiums AI

13 Allgemeiner Hinweis zu übergreifenden Bedingungen von Fachgebieten im Hinblick auf die Studienplan-Spezifikation. Dieser Hinweis wird solange angezeigt, bis er erfüllt ist.

14 Auswahl des Schwerpunktbereichs. Je nach Studienplan-Spezifikation kann bei bestimmten Schwerpunktbereichen eine zusätzliche spezialisierte Auswahl erforderlich sein, so wie im gezeigten Beispiel.

15 Auswahlmöglichkeit für die Anzahl der ECTS, um den allgemeinen Anforderungen des Studienplans zu entsprechen.

Zum Schluss sei angemerkt, dass Rot Dargestelltes bei den zugeordneten Prüfungen bedeutet, dass noch etwas fehlt, um das Prüfungsraster zu vervollständigen, während Gelb darauf hinweist, dasss einem Fach zu viele Kurse oder ECTS zugeordnet wurden und daher Teile davon anderen Fächern zugeordnet werden sollten.

5 Evaluierung

Im Rahmen der Evaluierung wird die Struktur, die Performance sowie die Qualität des Codes beurteilt. Die gewählten Technologien, wie im Kapitel 2 behandelt, entsprechen dem aktuellen Stand der Technik und sind optimal gewählt für dieses Projekt. Im Folgenden wird nun näher auf spezifische Aspekte des Projekts eingegangen.

5.1 Backend

Die Einteilung der Ordner und Pakete des Projekts ist sinnvoll und übersichtlich organisiert. Die Darstellung der Entities ist logisch und strukturiert. Eine mögliche Verbesserung wäre das Verfeinern der Attribute und Kardinalitäten der verbundenen Entities. Beispielsweise könnten Annotationen wie `@NotNull` oder `@NotEmpty` hinzugefügt werden, basierend auf den Definitionen in den Datenbanktabellen, um klarzustellen, welche Attribute stets einen Wert besitzen und welche Listen nie leer sind. Ein gutes Beispiel hierfür ist der Studienplan (`Studyplan`), der in der Regel immer mindestens ein Fach (`Subject`) besitzt, während jedes Fach (`Subject`) stets mindestens eine Bezeichnung (`SubjectText`) in einer bestimmten Sprache enthält. Trotz dieser Beobachtungen wurde nur aus der Datenbank gelesen, sodass das Fehlen dieser *Annotations* nicht kritisch ist.

5.1.1 Performance-Optimierung im Backend

Zu Beginn des Projekts wurde festgestellt, dass der Aufbau der Curricula, wie im Abschnitt 2.1 erwähnt, unverhältnismäßig lange Zeit in Anspruch nahm. Insbesondere mussten nach jeder kleineren Anpassung in der ATG oder bei Backend-Strukturanpassungen länger als 10 Minuten gewartet werden, bevor mit dem Debuggen begonnen werden konnte. Dieser Umstand verlangsamte den Entwicklungsprozess erheblich. Um diesen Engpass zu beheben, wurde eine Performance-Optimierung durchgeführt. Die Zeitmessungen erfolgten mittels `System.currentTimeMillis()`, wobei die resultierenden Zeiten nach dem Erstellen der Curricula in der Konsole angezeigt wurden.

Nachfolgend eine Übersicht der durchgeführten Optimierungen und ihre Auswirkungen:

Phase	Zeit (mm:ss)	Anzahl Curricula
Vor Optimierungen	10:03	267
Nach Einführung einer Relation zwischen `CourseVariant` und `StudyPlan`, kombiniert mit einer Optimierung des Algorithmus und einer gezielten Filterung auf lediglich 5 benötigte Curricula	00:51	5
Nach Integration eines ThreadPools über `ExecutorService` zur parallelen Generierung von Curricula	00:04	5
Nach Behebung eines Bugs, bei dem bestimmte Kurse in den Curricula fehlten	00:14	5
Verwendung der lediglich 4 erforderlichen Curricula	00:07	4

Tabelle 5.1: Übersicht der Optimierungsphasen und Zeitmessungen beim Aufbau von Curricula

Ein bemerkenswerter Aspekt war ein kleiner Bug, verursacht in der zweiten Phase, der nach der Einführung des ThreadPool in der dritten Phase entdeckt wurde und ein paar fehlende Kurse verursachte. Dieser wurde behoben, wobei sichergestellt wurde, dass die resultierenden Curricula konsistent mit den ursprünglichen Daten vor der Optimierung sind. Diese Korrekturen und Verbesserungen beschleunigten den Entwicklungsprozess erheblich, was zu einer effizienteren Fehlerbehebung und Weiterentwicklung führte.

5.2 Frontend

Das Frontend zeichnet sich durch eine exzellente benutzerfreundliche Gestaltung aus, die sowohl in der Navigation als auch in der Gesamtheit sehr gut gelungen ist. Die Komponenten, Klassen und Interfaces sind durchdacht strukturiert und interagieren effizient miteinander. Vereinzelt fielen Namensgebungen im Code auf, die entweder zu generisch waren oder nicht intuitiv genug, um deren Funktionalität auf den ersten Blick zu verstehen. Während des Debugging-Prozesses wurde festgestellt, dass Methoden teilweise unnötig wiederholt aufgerufen werden, was zu einer gewissen Redundanz führt, aber keinen merklichen Einfluss auf die Performance hat. Abgesehen von diesen Kleinigkeiten zeigt das Frontend insgesamt eine solide Code-Qualität und stellt damit eine durchwegs zuverlässige und effiziente Funktionsweise sicher.

6 Ausblick

Diese Projektarbeit hat zur Entwicklung einer mächtigeren domänenspezifischen Sprache (DSL) beigetragen, die nun in der Lage ist, komplexere Studiengänge abzubilden. Das genaue Ausmaß der Mächtigkeit der aktuellen Implementierung wurde nicht weiter untersucht, da der Fokus auf die Realisierung der zuvor mehrfach erwähnten Studiengänge gerichtet war. In diesem Kapitel wird ein Ausblick auf mögliche Verbesserungen und zukünftige Erweiterungen des Tools gegeben.

Zuerst wäre es spannend zu sehen, welche weiteren Studiengänge sich durch reines Hinzufügen ihrer Spezifikationen in den vorgesehenen Projektordner abbilden lassen. Theoretisch sollten Studiengänge, die weniger komplex sind als das Masterstudium Artificial Intelligence, problemlos integrierbar sein, und das trifft auf viele Studienrichtungen zu. Allerdings wäre es dann für eine zuverlässige Überprüfung notwendig, entsprechende Testpersonen einzusetzen, um die korrekte Integration sicherzustellen. Eine solche Testperson könnte allerdings lediglich die Zuordnungen überprüfen, die sich im Rahmen des aktuellen Studienfortschritts konstruieren lassen. Ein potenzieller nächster Schritt in der Weiterentwicklung dieses Tools könnte die Implementierung einer spezialisierten Testumgebung sein, die es erlauben würde, eine Testperson flexibel in verschiedene Studiengänge einzutragen und dabei unterschiedliche Szenarien des Studienverlaufs zu simulieren.

Eine weitere wesentliche Herausforderung, die sich bei aktuellen sowie zusätzlich integrierten Studienplan-Spezifikationen ergibt, ist deren laufende Wartung. Studienpläne können sich von Jahr zu Jahr ändern, was regelmäßige Aktualisierungen erfordert. Insbesondere müssen Äquivalenzen, also die anrechenbaren Kursklassen für jede Studiengangsspezifikation, regelmäßig aktualisiert werden, um eine konsistente und korrekte Darstellung der Kurse in der Benutzeroberfläche zu gewährleisten. Eine potenzielle Erweiterung des Tools, die den Wartungsaufwand erheblich reduzieren würde, wäre die automatische Integration der anrechenbaren Kursklassen direkt aus der Datenbank. Dies würde den manuellen Prozess des Eintragens der Äquivalenzen in die DSL-Spezifikationen, wie es im letzten Listing in Abschnitt 2.4.1 demonstriert wurde, überflüssig machen und eine effizientere Aktualisierung der Äquivalenzen sicherstellen.

Derzeit ist das Tool primär für den deutschsprachigen Raum konzipiert. Es zeigt sich jedoch eine sprachliche Diskrepanz, da wesentliche Hinweise und Inhalte der Kurse aus dem Bachelor- und Masterstudium Artificial Intelligence, in Englisch verfasst sind. Diese aktuelle Zweisprachigkeit innerhalb der Software könnte für Benutzer verwirrend sein. Daher wäre ein logischer Schritt in der Weiterentwicklung des Tools, eine Mehrsprachigkeit zu implementieren, die eine einheitliche Darstellung der Inhalte in verschiedenen Sprachen ermöglicht. Dies würde nicht nur die Benutzerfreundlichkeit erhöhen, sondern auch den Zugang für ein internationales Publikum erweitern.

Ein weiterer Aspekt, der für die Weiterentwicklung des Tools in Betracht gezogen werden könnte, ist die Optimierung für mobile Endgeräte. Momentan ist die Anwendung noch nicht tauglich für Smartphones. Im Hochformat werden die Bereiche *Zugeordnete Prüfungen* und *Nicht Zugeordnete Prüfungen* nebeneinander angezeigt, was zu einer überladenen, abgeschnittenen und schwer lesbaren Ansicht führt und die Bedienbarkeit erheblich beeinträchtigt. Noch problematischer ist die Darstellung im Querformat, da hier das Scrollen nach unten nicht möglich ist, was die Sicht auf die Auswahl der Studienrichtung und einige Überschriften beschränkt. Ein Responsive Design, das sich dynamisch an verschiedene Bildschirmgrößen anpasst, wäre eine bedeutende Verbesserung, die die Benutzerfreundlichkeit und Zugänglichkeit des Tools erheblich steigern könnte. In einer weiterführenden Entwicklungsphase könnte die Bereitstellung einer dedizierten App für Mobilgeräte in Betracht gezogen werden, um die Interaktion mit dem Tool nahtloser und intuitiver zu gestalten, insbesondere für Benutzer, die vermehrt auf mobile Technologie angewiesen sind.

Im Weiteren, nachdem sich das Tool in der Praxis über einen gewissen Zeitraum hinweg bewährt hat, wäre eine Integration in die bestehende myJKU-Plattform ein sinnvoller nächster Schritt. Der Grund dafür liegt nicht nur in der technologischen Kompatibilität beider Systeme, sondern auch darin, dass das Design und der Stil der Benutzeroberfläche bereits harmonisch auf myJKU abgestimmt wurden. Eine solche Integration würde die Zugänglichkeit des Tools erheblich verbessern, da die Studierenden es in einer bereits bekannten Online-Umgebung nutzen könnten. Diese Maßnahme würde auch das Gesamterscheinungsbild der digitalen Dienste der Universität stärken.

JOHANNES KEPLER
UNIVERSITÄT LINZ

Literatur

[1] *Oracle Database Technologies*. URL: https://www.oracle.com/database/technologies (besucht am 21. 09. 2023) (siehe S. 2).

[2] *The Java Language Environment*. URL: https://www.oracle.com/java/technologies/introduction-to-java.html (besucht am 23. 09. 2023) (siehe S. 4).

[3] *Spring Boot*. URL: https://spring.io/projects/spring-boot (besucht am 24. 09. 2023) (siehe S. 4, 9).

[4] *An Introduction to Hibernate 6*. 19. Sep. 2023. URL: https://docs.jboss.org/hibernate/orm/6.3/introduction/html_single/Hibernate_Introduction.html (besucht am 26. 09. 2023) (siehe S. 5).

[5] *Building a RESTful Web Service*. URL: https://spring.io/guides/gs/rest-service (besucht am 28. 09. 2023) (siehe S. 9).

[6] *Ist Angular Framework noch aktuell? Geschichte und Wissenswertes*. URL: https://bmu-verlag.de/angular-framework (besucht am 29. 09. 2023) (siehe S. 10).

[7] *Introduction to the Angular docs*. URL: https://angular.io/docs (besucht am 30. 09. 2023) (siehe S. 10).

[8] *Domänenspezifische Sprachen*. URL: https://www.jetbrains.com/de-de/mps/concepts/domain-specific-languages/ (besucht am 01. 10. 2023) (siehe S. 12).