JOHANNES KEPLER
UNIVERSITÄT LINZ

Entwicklung eines Prototypen für die wissensbasierte und vertraulichkeits-bewahrende Aufgabenzuteilung in intelligenten tutoriellen Systemen

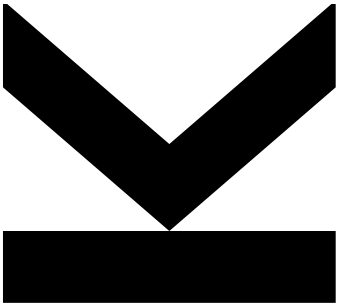
Eingereicht von
Victoria Edelsbacher, BSc

Angefertigt am
Institut für Wirtschaftsinformatik – Data & Knowledge Engineering

Betreuer
o.Univ.-Prof. DI Dr. Michael Schrefl

Mitbetreuung
Mag. Dr. Bernd Neumayr

Oktober 2021



Masterarbeit

zur Erlangung des akademischen Grades

Master of Science

im Masterstudium

Wirtschaftsinformatik

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

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

.....	.....
Datum	Unterschrift

Kurzfassung

In der heutigen Zeit kann man sich das Lernen ohne digitale Hilfsmittel kaum mehr vorstellen. Um Studierende beim Lernprozess zu unterstützen, werden in vielen Bereichen Lernplattformen für die Bereitstellung von Lernmaterialien sowie zur Lernfortschrittskontrolle eingesetzt. Da jedoch jeder Mensch individuell lernt, wird immer mehr versucht, das Lernen mithilfe individueller Aufgabenlisten an die Bedürfnisse der Studierenden anzupassen. Hierfür kommen intelligente tutorielle Systeme zum Einsatz. Für die Individualisierung der Aufgabenlisten wird jedoch eine große Menge an Daten über die Studierenden gesammelt. Daher spielt Privatsphäre eine besondere Rolle in diesem Bereich.

In dieser Masterarbeit wurde ein experimenteller Standalone-Prototyp entwickelt, welcher eine wissensbasierte und vertraulichkeitsbewahrende Zuteilung von Aufgaben in intelligenten tutoriellen Systemen realisiert. Hierbei wurde darauf geachtet, die Privatsphäre der Studierenden zu bewahren, indem die Informationen über die Studierenden nur von ihnen selbst einsehbar sind und nur das notwendige Minimum an Informationen mit den Lehrenden geteilt wird. Des Weiteren wurden verschiedene Einstellungsmöglichkeiten für die Studierenden entwickelt, mit welchen sie das Lernen an ihre individuellen Bedürfnisse anpassen können.

Abstract

In today's society it is hard to imagine learning without the digital world. In order to support students in the learning process, learning platforms are used in many areas to provide learning materials and to monitor the learning progress. However, since not all students learn in the same way, more and more attempts are being made to adapt learning to the needs of the students by means of individual task lists. Intelligent tutorial systems are used for this purpose. In order to individualise the task lists, a large amount of student data is collected. Therefore privacy plays a special role in this area.

In this master thesis an experimental standalone prototype was developed, which realises a knowledge-based and privacy-preserving assignment of tasks in intelligent tutorial systems. In doing so, the privacy of the students was preserved by ensuring that information about the students is only accessible by the students themselves and that only the minimum necessary information is shared with the teachers. In addition, various configuration options have been developed for the students, with which they can adapt the learning process to their individual needs.

Inhaltsverzeichnis

Kapitel 1

Einführung

In diesem Kapitel wird ein erster Einblick in die Arbeit gegeben. Dabei wird zuerst auf die Motivation der Arbeit eingegangen. Anschließend wird die Problemstellung der Arbeit präsentiert und eine mögliche Lösungsidee vorgestellt. Zudem werden die Ziele und Anforderungen sowie der Aufbau der Arbeit beschrieben.

1.1 Motivation

Das Lernen spielt sich zunehmend in der digitalen Welt ab, der Wissenserwerb ist in vielen Bereichen kaum mehr vorstellbar ohne digitale Unterstützung [1]. Daher wurden in den letzten Jahren zunehmend Lernplattformen für die Unterstützung des Lernens eingesetzt, Studierenden werden dabei Lernunterlagen über eine Lernplattform zur Verfügung gestellt [1]. Eine erweiterte Form von Lernplattformen sind intelligente tutorielle Systeme (ITS). Hierbei wird versucht, beim Lernen auf die individuellen Bedürfnisse der Studierenden einzugehen [1].

Einer der Grundgedanken von intelligenten tutoriellen Systemen ist, dass nicht alle Menschen gleich lernen, sondern es viele verschiedene Faktoren gibt, welche das Lernen der Studierenden beeinflussen [2]. Daher sollte auf diese Faktoren eingegangen werden, um die Lernerfahrungen möglichst positiv und effektiv zu gestalten. Zudem kann mit Lernplattformen das Lernen ortsunabhängig durchgeführt werden und ermöglicht so ein effektiveres Zeitmanagement für die Lernenden [2].

Bereits 1970 hat Jaime Carbonell in einem Artikel das Programm SCHOLAR vorgestellt, welches als das erste intelligente tutorielle System gilt [3]. In SCHOLAR wurde eine natürliche Sprache verwendet, um die Fragen der Lernenden zu beantworten oder wiederum selbst eine Frage zu stellen und auf eine Rückmeldung zu reagieren [3]. Dieses System stellte bereits unterschiedlichen Studierenden unterschiedliche Fragen, anstatt nur einem

strikten Lernpfad zu folgen [3].

Die Erstellung eines individuellen Lernpfads setzt Wissen über die Lernenden voraus. In dieser Masterarbeit wird untersucht, wie in intelligenten tutoriellen Systemen den Lernenden individuell Aufgaben zugeteilt werden können und gleichzeitig ihre Privatsphäre bewahrt werden kann.

1.2 Problemstellung

Um Grundkenntnisse einer Datenbanksprache (z.B. SQL) zu erwerben, sollen Studierende schrittweise die wesentlichen Sprachkonstrukte, z.B. GROUP BY oder OUTER JOIN, dieser Datenbanksprache kennen und anwenden lernen. Dies gelingt vor allem durch das Lösen von Aufgaben (Übungsbeispielen), wie dem Formulieren von Abfragen ausgehend von textuellen Aufgabenbeschreibungen. Als Lernziel wird in dieser Masterarbeit nicht eine fortgeschrittene Beherrschung der Datenbanksprache um komplexe Problemstellungen selbstständig lösen zu können gesehen, sondern lediglich die selbstständige, erfolgreiche Anwendung der einzelnen Sprachkonstrukte. In diesem Sinne entspricht jedes Sprachkonstrukt einem Lernziel. Die Zuordnung von Aufgaben zu Studierenden soll dynamisch abhängig von Lernfortschritt, Lernziel und User-Präferenzen erfolgen. Damit die Aufgaben individuell zugeteilt werden können, muss privates Wissen über die Studierenden gespeichert werden.

1.3 Lösungsidee

Das schrittweise Erarbeiten der Sprachkonstrukte bedeutet in dieser Masterarbeit, dass die Sprachkonstrukte mit den Aufgaben nach und nach eingeführt werden. Die Schwierigkeit der mit einer Aufgabe neu hinzukommenden Sprachkonstrukte soll so gewählt sein, dass die Aufgabe für Studierende herausfordernd, aber auch beherrschbar ist, um gleichermaßen Langeweile und Frustration zu vermeiden. Die Schwierigkeit einer Aufgabe wird auch als *relative Schwierigkeit* (relativ in Bezug zu bereits gelösten Aufgaben) angesehen. Die gewünschte relative Schwierigkeit der einzelnen Aufgaben in einer Aufgabenliste ist individuell abhängig vom Lerntyp der Studierenden. Bei der Generierung individueller Aufgabenlisten soll die individuell gewünschte relative Schwierigkeit berücksichtigt werden.

Um die Generierung individueller Aufgabenlisten zu ermöglichen, werden die Aufgaben entsprechend der einzusetzenden Sprachkonstrukte in Aufgabenklassen eingeteilt, wobei jedes Sprachkonstrukt einer Aufgabenklasse entspricht und eine Aufgabe auch mehreren Aufgabenklassen zugeordnet sein kann. Mit der Lösung einer Aufgabe erreichen Studierende das von der Aufgabenklasse dieser Aufgabe repräsentierte Lernziel. Wenn eine Aufgabe zu mehreren Aufgabenklassen gehört, erreichen Studierende mit der Lösung der Aufgabe auch mehrere Lernziele.

Die Aufgabenklassen (Sprachkonstrukte, Lernziele) werden in Hierarchien (siehe Abbildung 1.1) geordnet, z.B. die Aufgabenklasse für das Sprachkonstrukt *OUTER JOIN* (hier in der Rolle der Child- oder Sub-Klasse) ist unter der Aufgabenklasse für das Sprachkonstrukt *JOIN* (hier in der Rolle als Parent- oder Super-Klasse). Wenn eine Aufgabe einer Subklasse zugeordnet ist, dann ist sie automatisch auch allen direkten und indirekten Superklassen zugeordnet. Eine Aufgabe der Klasse *OUTER JOIN* gehört also auch zur Klasse *JOIN* und mit der Lösung dieser Aufgabe erreicht man beide Lernziele.

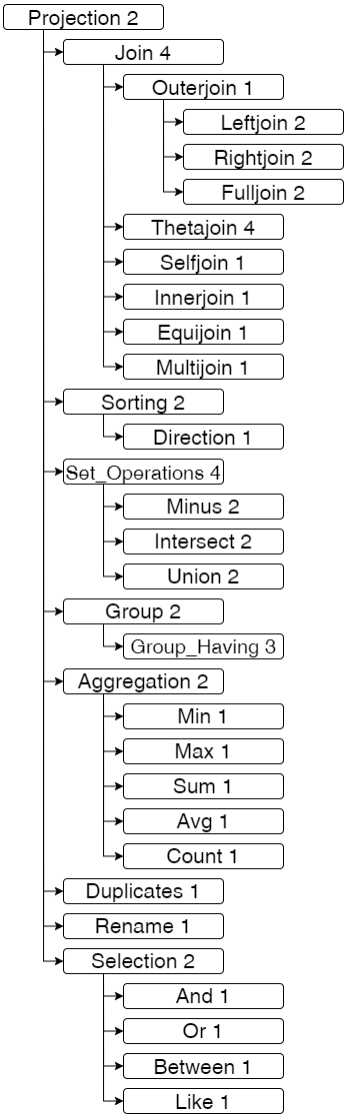


Abbildung 1.1: Gewichtete Generalisierungshierarchie

Um nun die relative Schwierigkeit von Aufgaben in Bezug auf die einzusetzenden Sprachkonstrukte und die bereits vorher erreichten Lernziele bestimmen zu können, wird jeder Aufgabenklasse eine Schwierigkeit (ausgedrückt als natürliche Zahl) zugeordnet. Die relative Schwierigkeit einer Aufgabe ergibt sich aus der Summe der Schwierigkeiten der Aufgabenklassen, zu denen die Aufgabe direkt oder indirekt gehört, wobei die Schwierigkeiten von Aufgabenklassen, zu denen bereits eine Aufgabe gelöst wurde, nicht mitgezählt werden.

Die Aufgabenklasse *COUNT* hat eine Schwierigkeit von 1, wenn jedoch noch keine Aufgabe mit einer Aufgabenklasse aus *AGGREGATION* gelöst wurde, wird die relative Schwierigkeit von *AGGREGATION* zu *COUNT* hinzugezählt. Damit ergibt sich eine gesamte relative Schwierigkeit für *COUNT* von 3. Wurde zudem die Aufgabenklasse *PROJECTION* noch nicht gelöst, muss auch diese relative Schwierigkeit hinzugezählt werden. Damit erreicht *COUNT* eine gesamte relative Schwierigkeit von 5. Beim Generieren einer individuellen Aufgabenliste sollen die Aufgaben nun so gewählt werden, dass die relative Schwierigkeit jeder Aufgabe möglichst der gewünschten relativen Schwierigkeit der Studierenden entspricht.

Lehrende stellen eine Aufgabensammlung (früher auch als Übungszettel bezeichnet) für Studierende nun als Menge von zu erreichenden Lernzielen, also Aufgabenklassen, zur Verfügung. Studierende können sich entsprechend ihrer Präferenzen daraus ihre individuelle Aufgabenliste generieren lassen. Diese individuelle Aufgabenliste wird in dieser Masterarbeit auch als Lernpfad bezeichnet.

Bei der Generierung von Aufgabenlisten ist auch noch zu beachten, dass jede Aufgabe zu einer Aufgabenfamilie gehört. Eine Aufgabenfamilie ist eine Menge von Aufgaben, die einen Teil der Aufgabenstellung gemeinsam haben. Im Falle von Aufgaben zu SQL-Abfragen teilen die Aufgaben einer Aufgabenfamilie das Datenbankschema, auf das die Abfragen ausgeführt werden sollen. Ein Wechsel zwischen Aufgabenfamilien innerhalb einer Aufgabenliste kann als störend empfunden werden.

Ein weiterer wichtiger Aspekt ist die Privatsphäre der Studierenden. Bei der Generierung der Aufgabenliste sowie bei der Überprüfung der Zielerreichung soll diese gewahrt werden. Dies geschieht, indem die Einstellungen der Studierenden nur für sie selbst verfügbar sind und nicht geteilt werden. Zudem erfolgt die Generierung der Aufgabenliste nur lokal bei den Studierenden. Die anschließende Abarbeitung der Aufgabenliste erfolgt anonym. Studierende bekommen nach dem erfolgreichen Absolvieren einer Aufgabe eine signierte Bestätigung von einer zentralen Instanz.

Um nachvollziehen zu können, welche Aufgaben bereits von den Studierenden gelöst wurden, müssen Studierende nur einen Nachweis erbringen, dass die Aufgabe gelöst wurde, jedoch nicht wie sie bei der Lösung vorgegangen sind. Dabei gibt es für jede vorgegebene Aufgabenklasse eine Bestätigung der Lösung samt Lösung, jedoch nicht für etwaige Vorschritte, welche aufgrund der Einstellungen der Studierenden durchgeführt wurden. Des Weiteren wird bei der Lernfortschrittskontrolle nicht vermerkt, wie viele Schritte notwendig waren, um das Ziel zu erreichen, es wird nur das notwendige Minimum an Informationen zur Lernfortschrittskontrolle verwendet.

1.4 Ziele und Anforderungen

Ziel dieser Arbeit ist es, einen experimentellen Standalone-Prototypen zu entwickeln, mit welchem eine individuelle Aufgabenzuteilung in intelligenten tutoriellen Systemen wissensbasiert und vertraulichkeitsbewahrend durchgeführt wird. Dieser Prototyp soll als selbstständige Web-Anwendung laufen und am Beispiel des Wissensgebiets SQL-Abfragen umgesetzt werden. Folgende Anforderungen soll der Standalone-Prototyp erfüllen.

A1 Individualisierung Aufgabenliste

Der Prototyp soll verschiedene Einstellungsmöglichkeiten haben, mit welchen die Aufgabenliste an die Bedürfnisse der Studierenden angepasst werden kann.

A1.1 Einstellungsmöglichkeit Aufgabenfamilie
Jede Aufgabe wird einer Aufgabenfamilie zugeordnet. Eine Aufgabenfamilie entspricht einem Datenbankschema und besteht aus mehreren zusammenhängenden Datenbanktabellen. Studierende soll die Möglichkeit haben anzugeben, wie wichtig es ihnen ist, immer dieselbe Aufgabenfamilie zu verwenden oder ob verschiedene Aufgabenfamilien vorkommen können.

A1.2 Einstellungsmöglichkeit Schwierigkeitsgrad
Es gibt verschiedene Aufgaben, welche die Studierenden absolvieren können, um eine Aufgabenklasse zu erreichen. Studierende sollen die Möglichkeit haben, die Geschwindigkeit, mit welcher neue Aufgabenklassen in den Aufgaben hinzukommen, an ihre Bedürfnisse anzupassen.

A1.3 Anpassung Aufgabenliste
Jede Aufgabe hat einen relativen Schwierigkeitsgrad. Studierende können ihren bevorzugten relativen Schwierigkeitsgrad für die Aufgabenliste festlegen. Sollte eine Aufgabenliste für die Studierenden zu schwer oder zu leicht sein, sollen sie die Möglichkeit haben, eine neue Aufgabenliste mit angepasstem relativen Schwierigkeitsgrad generieren zu lassen.

A2 Wissensstand der Studierenden

Um das Lernen der Studierenden besser unterstützen zu können, sammeln Lernplattformen Wissen zum Lernfortschritt der Studierenden. Studierende sollen sich dieses Wissen ansehen können. Folgende Anforderungen sollen daher in Bezug auf das Wissen zum Lernfortschritt der Studierenden erfüllt werden.

A2.1 Wissensstand Aufgabenklasse
Die Studierenden sollen sich ihren aktuellen Wissensstand in Bezug auf die Aufgabenklassen ansehen können.

A2.2 Wissensstand und Aufgabenlistengenerierung
Der Wissensstand aus bereits abgeschlossenen Aufgaben soll bei der Aufgabenlistengenerierung berücksichtigt werden. Dabei soll vereinfachend davon ausgegangen werden, dass Studierende die Aufgabenklassen nach einmaligem Üben beherrschen. Eine Aufgabenklasse soll daher nicht zweimal erlernt werden müssen.

A2.3 Vergangene Abfragen
Studierende sollen ihre verschiedenen Versuche für eine Übungsaufgabe einer Aufgabenklasse einsehen können. Dazu sollen sie sich alle ausgeführten Abfragen anzeigen lassen können.

A3 Wahrung der Privatsphäre

Eine zentrale Anforderung ist die Wahrung der Privatsphäre der Studierenden, wobei in dieser Masterarbeit die Benutzerpräferenzen und die Art des Lernens als Teil dieser Privatsphäre angesehen wird. Folgende Anforderungen sollen erfüllt werden.

A3.1 Privates Wissen über die Studierenden
Das Wissen über die Studierenden und ihre durchgeführten Übungsaufgaben soll nur von ihnen selbst einsehbar sein. Es soll sichergestellt werden, dass nur Studierende zu ihren eigenen privaten Daten Zugang haben.

A3.2 Lernfortschrittsbestätigung
Bei der Übermittlung der Lernfortschrittsbestätigung soll nur ein Minimum an Informationen über die Studierenden mitgesendet werden, welche für die Absolvierung einer Aufgabenklasse notwendig sind.

1.5 Aufbau der Arbeit

Diese Arbeit gliedert sich wie folgt.

- In diesem Kapitel wurde die Motivation, Problemstellung sowie Lösungsidee dieser Arbeit beschrieben. Zudem wurde auf die Ziele und Anforderung eingegangen.
- In Kapitel 2 werden die relevanten theoretischen Grundlagen, welche für das Verständnis der späteren Umsetzung notwendig sind, beschrieben. Hierbei wird auf Lernplattformen, intelligente tutorielle Systeme, Privatsphäre-bewahrende Systeme sowie Answer Set Programming eingegangen und ihre Zusammenhänge näher beschrieben.

- Anschließend beginnt mit Kapitel 3 *Architektur und konzeptuelles Modell* der praktische Teil der Arbeit. In diesem Kapitel werden die allgemeine Architektur der Anwendung, die verwendeten Plattformen und das konzeptuelle Modell beschrieben.
- In Kapitel 4 wird auf die konkrete technische Umsetzung des Backends eingegangen. In diesem Kapitel werden das Maintenance-Service, das Task-Service und das Task-Assignment-Generator-Service beschrieben.
- Anschließend werden in Kapitel 5 die Implementierungsdetails des Frontends (Web-Anwendung) beschrieben. Das Frontend wird in Studierenden-Sicht und Administrator-Sicht unterteilt.
- Kapitel 6 beschreibt das Kernelement des Prototyps, den Task-Assignment-Generator, welcher für die wissensbasierte Zuteilung der Übungsaufgaben zuständig ist.
- In Kapitel 7 wird auf die einzelnen Anforderungen noch einmal eingegangen und diese in Bezug auf die Erreichung in den einzelnen Kapiteln betrachtet.
- Das abschließende Kapitel 8 gibt eine generelle Zusammenfassung der Arbeit und beschreibt das weitere Vorgehen.

Kapitel 2

Hintergrund

In den nachfolgenden Abschnitten wird beschrieben, was man unter Lernplattformen, intelligenten tutoriellen Systemen, einem individuellen Lernpfad und Privatsphäre-bewahrenden Systemen versteht und wie diese Begriffe zusammenhängen. Zudem wird Answer Set Programming beschrieben und näher auf das "Constraint Satisfaction Problem" eingegangen. Es wird versucht ein Basiswissen aufzubauen, welches für die spätere technische Umsetzung notwendig ist.

2.1 Lernplattformen

In diesem Abschnitt wird ein allgemeiner Überblick gegeben, was Lernplattformen sind und welche Funktionalitäten sie zur Verfügung stellen und es werden drei der bekanntesten Lernplattformen vorgestellt.

Allgemeines

Generell versteht man unter einer Lernplattform ein System, mit welchem Lerninhalte sowohl zur Verfügung gestellt als auch erarbeitet werden können. Es dient als Schnittstelle zwischen den ÜbungsleiterInnen und den Studierenden [4]. Auf Lernplattformen werden, meist unter einer gemeinsamen Visualisierung, mehrere Teilsysteme eingebunden und stellen somit diverse Funktionalitäten zur Verfügung [5]. Sie bieten damit den Vorteil, dass nicht mehrere verschiedene Systeme verwendet werden müssen, sondern etwa die Kommunikation, Verwaltung und Absolvierung von Tests direkt in einem System erfolgen können [4].

Mit Lernplattformen wird den Studierenden das Lernen von überall und zu jeder Zeit er-

möglicht, Kursinhalte können direkt über den Webbrowser abgerufen werden und eine Anwesenheit an einem speziellen Ort ist nicht mehr notwendig [5]. Studierende können so individuell entscheiden, wann sie sich welche Lerninhalte aneignen wollen und können das Lernen an ihre persönlichen Bedürfnisse anpassen [5]. Des Weiteren haben auch ÜbungsleiterInnen den Vorteil, dass sie mit Lernplattformen die Kursinhalte ortsunabhängig zur Verfügung stellen können [5].

Um das volle Potenzial von Lernplattformen zu nutzen, binden viele Systeme zusätzlich intelligente tutorielle Systeme ein [2]. In solchen Systemen werden die Lerninhalte mithilfe eines individuellen Lernpfads zur Verfügung gestellt und können so das Lernen für die Studierenden personalisieren [2]. In Abschnitt 2.2 werden intelligente tutorielle Systeme genauer erklärt und in Abschnitt 2.3 wird auf den Lernpfad näher eingegangen.

Funktionalitäten

Eine der wesentlichsten Funktion von Lernplattformen ist das Erstellen und Präsentieren von Lerninhalten, womit ÜbungsleiterInnen ihre Kursinhalte digital bereitstellen und verwalten können [5]. ÜbungsleiterInnen können so entscheiden, wann und wie den Studierenden Inhalte zur Verfügung gestellt werden und ob die Inhalte nur für eine gewisse Dauer oder für einen gewissen Kurs verfügbar sein sollen [5].

Lernplattformen bieten verschiedene Werkzeuge an, um die Kommunikation zwischen den ÜbungsleiterInnen und den Studierenden zu ermöglichen und zu vereinfachen [4]. Dies kann sowohl synchron (in Echtzeit) mithilfe von Nachrichtendiensten funktionieren oder asynchron in Form von Kursankündigungen, die direkt auf der Lernplattform sichtbar sind [5]. Auch die Kommunikation per E-Mail, welche mithilfe der Lernplattform versendet werden, ist sehr häufig in Gebrauch [5]. Des Weiteren werden bei einigen Lernplattformen Wikis, Blogs und File-Sharing-Systeme für die Unterstützung eingesetzt [5].

Unter administrativen Tätigkeiten auf Lernplattformen versteht man das Erstellen und Verwalten von Benutzerkonten, sowie die generelle Verwaltung von Kursen und Lerninhalten [5]. AdministratorInnen haben Einsicht in das gesamte System und können so bei Problemen die ÜbungsleiterInnen und Studierenden unterstützen [5]. Zudem sollen sie durch das Überwachen der Systeme die allgemeine Sicherheit der Lernplattform erhöhen und können bei unbefugten Aktivitäten bei der Nachverfolgung helfen [5].

Für das Überprüfen der Leistung von Studierenden werden spezielle Tools auf Lernplattformen eingesetzt [5]. Am häufigsten werden Tools als Unterstützung beim Erstellen und Durchführen von Tests eingesetzt. Diese Tools können entweder selber entwickelt werden oder es können bereits existierende Tools in die Lernplattform integriert werden [5]. Der Einsatz von solchen Tools kann es den ÜbungsleiterInnen soweit vereinfachen, dass sie nur mehr eine Fragensammlung zusammen stellen müssen, mit welchem anschließend das Tool automatisch einen Test zusammen stellt und diesen direkt überprüft [5].

Beispiele für Lernplattformen

Bevor standardisierte Lernplattformen entwickelt wurden, mussten ÜbungsleiterInnen selbst HTML bzw. Programmierkenntnisse beherrschen, um Kursinhalte online bereitzustellen [5]. Mit der Vielzahl an verschiedenen, bestehenden Lernplattformen ist es möglich, ein System zu finden, welches für die individuellen Bedürfnisse passend ist [5]. Man unterscheidet dabei zwischen webbasierten Lernplattformen und Systemen, die auf einem Endgerät installiert werden müssen (Desktop Applikationen). Des Weiteren gibt es Systeme, welche als Cloud-Lösungen oder als Local-Services angeboten werden [5]. Zudem gibt es Systeme, welche Open-Source-Projekte sind und Systeme, welche Closed-Source-Projekte sind [5].

Eine der bekanntesten Lernplattformen ist *Moodle*. Es ist eine Open-Source-Anwendung, mit welcher Kurse über das Web zur Verfügung gestellt werden können [6]. Moodle bietet dabei zum einen durch Diskussionsforen die Möglichkeit zur Interaktion zwischen den Studierenden und Lehrenden, als auch die Möglichkeit, dass Tests direkt auf der Lernplattform durchgeführt werden können [6]. Moodle unterstützt zudem mehr als 120 Sprachen [6].

Blackboard ist eine international bekannte Lernplattform. Es bietet seine webbasierte Lernplattform sowohl als Cloud-Lösung als auch als Local-Services an [7]. Blackboard bietet eine sehr breit gefächerte Produktlinie an, dies resultiert daraus, dass von Blackboard mehrere Lernplattformen übernommen wurden [5]. Zu den Basisfunktionalitäten von Blackboard zählen die Kommunikation für Studierende und Lehrende, sowie die Verwaltung von verschiedenen Kursen [7].

Canvas ist eine Cloud-basierte, Open-Source-Lernplattform. Es werden über 200 verschiedene Tools für ÜbungsleiterInnen und Studierende angeboten, welche unter anderem die Kommunikation, Bewertung sowie Verwaltung von Kursen unterstützen [8]. Zudem bietet Canvas eine iOS- und Android-App an, mit welcher direkt auf die Lernplattform zugegriffen werden kann [8].

2.2 Intelligente tutorielle Systeme

Mithilfe von intelligenten tutoriellen Systemen soll das individuelle Lernen auf Lernplattformen ermöglicht werden [2]. Man versteht darunter ein System, welches für jeden Studierenden mithilfe von verschiedenen Daten, wie etwa zum Vorwissen, einen individuellen Lernpfad erstellt [2]. Bei einigen Systemen gibt es zudem Einstellungsmöglichkeiten, mit welchen der Lernpfad an den individuellen Lerntyp angepasst werden kann [2].

Um auf die individuellen Bedürfnisse der Studierenden eingehen zu können, muss die Leistung der Studierenden mitverfolgt werden [2]. Dazu müssen eine Vielzahl an Daten über die Studierenden gespeichert werden, was wiederum ein großes Risiko für Datenmissbrauch mit sich bringt [9]. Insofern spielt die Privatsphäre eine besondere Bedeutung im Zusammenhang mit intelligenten tutoriellen Systemen [9]. Im Abschnitt 2.4 wird näher beschrieben,

was man unter Privatsphäre-bewahrende Systeme versteht.

Der Begriff „intelligente tutorielle Systeme“ wurde erstmals von Sleeman und Brown in ihrem Buch 1982 verwendet [2]. Wenger entwickelte anschließend im Jahre 1987 die sogenannte „traditionelle Architektur für intelligente tutorielle Systeme“, welche sich aus Wissensmodell, Studierenden-Modell, Tutoren-Modell und Benutzerschnittstelle zusammen setzt [2]. Seit dem hat sich einiges verändert, jedoch ist die ursprüngliche Architektur, mit gewissen Erweiterungen und Anpassungen, noch gültig [2]. Abbildung 2.1 zeigt diese Architektur.

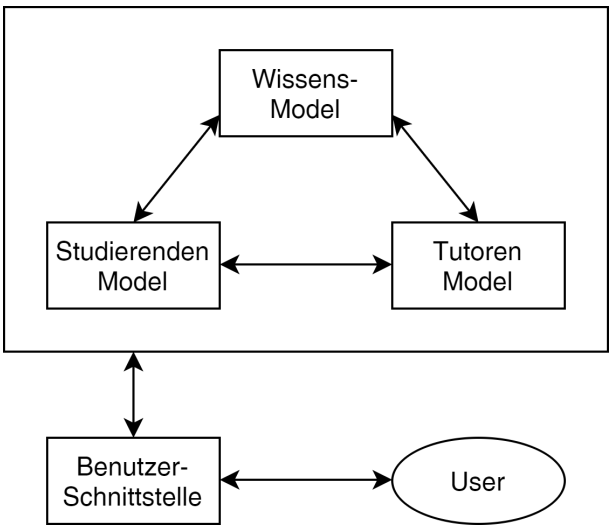


Abbildung 2.1: Architektur intelligente tutorielle Systeme [2]

Wissensmodell

Das Wissensmodell, oft auch als Expertenwissen bezeichnet, übernimmt verschiedene Aufgaben [2]. Zum einen wird es für die Repräsentation des zu vermittelnden Wissens verwendet und zum anderen dient es als Grundlage für die Beurteilung der Studierenden [2]. Damit ein intelligentes tutorielles System effektiv sein kann, muss es mit dem spezifischen Wissen des Wissensmodells ausgestattet werde [10]. Das abgebildete Wissen kann in drei Kategorien eingeordnet werden: deklaratives Wissen (Faktenwissen), prozedurales Wissen (praktisches Wissen) und heuristisches Wissen (Erfahrungs- und Problemlösungswissen) [10].

Das Wissensmodell kann mithilfe von semantischen Netzwerken und Ontologien abgebildet werden [2]. Es gibt drei verschiedene Ansätze für die Darstellung des Wissens:

- Black-Box Modell

Bei diesem Modell sind die Vorgänge des Programms nicht einsehbar, es werden nur die Endergebnisse übermittelt [10].

- Glas-Box Modell
  Bei diesem Modell können die einzelnen Schritte des Programms nachverfolgt werden. Dies hat den Vorteil, dass die Studierenden Fragen stellen können und eine schrittweise Lösung des Problems bekommen [10].

- Kognitives Modell
  Bei einem kognitiven Modell wird versucht, das Wissen so nachzubauen, wie es im menschlichen Geist repräsentiert wird. Das Lernen soll erleichtert werden, in dem das kognitiv Denken besser unterstützt wird [10].

Studierenden-Modell

Das Studierenden-Modell wird als das Kernsystem von intelligenten tutoriellen Systemen angesehen, es repräsentiert das Wissen über Studierende, ihre Kompetenzen und Lernleistungen, welche als Grundlagen für die Entwicklung eines Lernpfads dienen [2]. Bei einigen intelligenten tutoriellen Systemen wird nicht nur versucht auf den aktuellen Wissensstand der Studierenden einzugehen, sondern auch auf ihren emotionalen Zustand, sowie den Lernstil [11]. Die Überprüfung des Wissensstands der Studierenden kann relativ einfach mit diversen Tests durchgeführt werden. Es ist jedoch schwieriger, den emotionalen Zustand und den individuellen Lernstil der Studierenden herauszufinden [2].

Das Studierenden-Modell wird als dynamisches Modell betrachtet und muss zu jederzeit den aktuellen Wissensstand, sowie eine Historie über alle bereits absolvierten Aufgaben der Studierenden enthalten [2]. Um das Lernen für die Studierenden positiv zu gestalten, soll es eine visuelle Darstellung des erreichten Wissens, sowie des Lernprozess der Studierenden geben [2]. Das Studierenden-Modell enthält zwar alle relevanten Informationen über die Studierenden, um jedoch auch auf die individuellen Bedürfnisse eingehen zu können, muss das Studierenden-Modell diese Informationen dem Tutoren-Modell bereitstellen. Das Tutoren-Modell kann anschließend dieses Wissen nutzen und Entscheidungen treffen in Bezug auf die Interaktion mit den Studierenden [11].

Tutoren-Modell

Das Tutoren-Modells steht im ständigen Austausch mit den Studierenden und versucht auf die individuellen Bedürfnisse einzugehen [12]. Es enthält die Wissensbasis für die Lernstrategie und muss entscheiden, wann den Studierenden was präsentiert werden soll [12]. Das Tutoren-Modell wird sowohl vom Wissensmodell als auch vom Studierenden-Modell mit Informationen versorgt [2]. Es muss mithilfe der verfügbaren Informationen Strategien und Aktionen entwickeln und entscheiden, ob und in welcher Form mit den Studierenden interagiert werden soll [2]. Das Hauptproblem bei der Interaktion mit den Studierenden

besteht darin, zu entscheiden, wann und welcher Form eingegriffen werden soll und ob das Eingreifen für die Studierenden positive oder negative Effekte hat [12].

Die Interaktion zwischen den Studierenden und dem Tutoren-Modell erfolgt über die Kommunikationskomponenten, welche auch als Benutzer-Schnittstelle bezeichnet wird [2]. Die Kommunikation kann in Form von Feedback, Hinweismeldungen sowie Dialogen erfolgen [2]. Die Benutzer-Schnittstelle stellt das Wissen der Domain in verschiedenen Formen zur Verfügung und passt sich an die Bedürfnisse der Studierenden an [2].

2.3 Individualisierter Lernpfad

Als Lernpfad versteht man die Zusammenstellung von individuell, je nach Können zusammengestellten Lernaufgaben, mit welchen die Studierenden ihr persönliches Lernziel erreichen sollen [13]. Der Lernpfad kann als eine Sammlung von Aufgaben angesehen werden, bei welchen die Studierenden verschiedene Aktivitäten ausführen müssen, um einen spezifischen Wissensstand zu erreichen [14]. Bei traditionellen Lehrmitteln, wie z.B. Lehrbüchern, bekommen alle Studierenden den gleichen Lernpfad, wodurch nur sehr eingeschränkt auf die verschiedenen Bedürfnisse der Studierenden eingegangen wird [14]. Im Gegensatz dazu, wird mit intelligenten tutoriellen Systeme versucht, einen individuellen Lernpfad für die Studierenden zu gestalten [13].

Der Grundgedanke ist, dass Studierende über unterschiedliches Hintergrundwissen, andere Lerntypen und oft auch verschiedene Lernziele verfügen [14]. Daher soll mithilfe eines individualisierten Lernpfads auf die unterschiedlichen Bedürfnisse der Studierenden eingegangen werden, um so Frustration und Langeweile zu vermeiden [13]. Das Ziel ist außerdem, dass Studierende mithilfe eines individualisierten Lernpfads Wissen schneller und besser erarbeiten können, da auf ihren persönlichen Lerntyp eingegangen wird [13].

2.4 Privatsphäre-bewahrende Systeme

Unter Privatsphäre versteht man nach Warren und Brandeis „the right to be left alone“ [15]. Sie gingen damit im neunzehnten Jahrhundert gegen die Verletzung der Privatsphäre durch die Zeitungen vor [15]. Alan Westin beschrieb Privatsphäre als den „Wunsch der Menschen, frei zu wählen, unter welchen Umständen und in welchem Umfang sie sich selbst, ihre Haltung und ihr Verhalten anderen gegenüber bloßstellen wollen“ [16]. Privatsphäre ist nicht mehr nur eine Frage des öffentlichen Raums, sondern gewinnt in der heutigen Zeit, etwa durch das Internet und den damit verknüpften Geräten, zunehmend an Bedeutung [17].

Intelligente tutorielle Systeme benötigen, um auf die individuellen Bedürfnisse eingehen zu können, eine große Menge an Informationen über ihre Benutzer [2]. Dies birgt jedoch die Gefahr des Datenmissbrauchs [2]. Es sollte daher bereits bei der Entwicklung von Systemen speziell, wenn personenbezogene Daten verarbeitet werden, darauf geachtet werden, dass die Privatsphäre der Benutzer bewahrt wird [18]. Zudem besteht das Risiko, dass die Benutzer nicht mehr selbst bestimmen können, wann und welche Informationen über sie selbst an andere weitergegeben werden [18].

Privacy By Design

Eine Möglichkeit, um die Privatsphäre der Benutzer in einem System zu bewahren, ist der Ansatz von *Privacy by Design* [19]. Hiermit soll den besonderen Anforderungen nachgekommen werden, welche im Zusammenhang mit personenbezogenen Daten an Systeme gestellt werden [19]. Oftmals bringen neue technische Systeme versteckte Gefahren, welche nach der Entwicklung des Grunddesigns nur schwer entfernt werden können [19]. Es sollte daher bereits beim Entwurf neuer Systeme auf die Datenschutzprobleme geachtet werden [19]. Privacy by Design sollte in allen Systemen verwendet werden, welche personenbezogene Daten verarbeiten [19].

Die Grundidee von *Privacy By Design* ist, dass Systeme entwickelt werden, mit welchen die Verarbeitung von personenbezogenen Daten minimiert oder ganz vermieden wird [19]. Inhaltliche Daten und Personenkennzeichnungen sollen voneinander getrennt werden, es werden Pseudonyme verwendet oder Personendaten werden frühzeitig gelöscht [19]. Folgende allgemeine Ziele sollten nach Schaar (2010) bei dem Entwurf von Systemen beachtet werden:

- **Datenminimierung**
  Es soll darauf geachtet werden, dass das System so wenig personenbezogenen Daten wie möglich erhebt, verarbeitet, speichert oder nutzt [19].
- **Kontrollierbarkeit**
  Den Nutzern des Systems soll die Möglichkeit gegeben werden, zu kontrollieren,

welche ihrer Daten vom System verarbeitet werden. Zudem soll es die Möglichkeit der Einwilligung und des Widerspruchs zur Verarbeitung geben [19].

- **Transparenz**
  Alle Beteiligten sollen ausführlich über die Funktionen des Systems informiert werden [19].
- **Vertraulichkeit der Daten**
  Es muss sichergestellt werden, dass nur befugte Personen Zugang zu personenbezogenen Daten haben [19].
- **Datenqualität**
  Die Qualität der Daten soll von technisch ausgebildeten Personen überprüft werden. Zudem soll sichergestellt werden, dass alle Daten zugänglich sind, wenn es für rechtliche Angelegenheiten notwendig ist [19].
- **Möglichkeit zur Datentrennung**
  Systeme, die für unterschiedliche Zwecke genutzt werden, sollen auch gewährleisten, dass Daten und Prozesse, welche unterschiedlichen Aufgaben oder Zwecken dienen, auf sichere Weise voneinander getrennt werden [19].

Informationssicherheit

Ein weiterer wichtiger Aspekt bei der Wahrung der Privatsphäre in Systemen ist die Informationssicherheit [18]. Mit dem CIA-Dreieck (Abbildung 2.2) werden die drei wesentlichen Eigenschaften von Daten beschrieben, welche man sicherstellen möchte, diese bestehen aus *Confidentiality* (Vertraulichkeit), *Integrity* (Integrität) und *Availability* (Verfügbarkeit) [18]. Im nachfolgenden Abschnitt wird näher auf das CIA-Dreieck eingegangen.

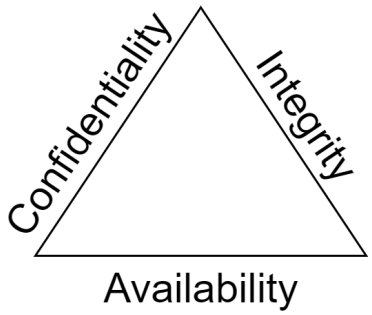


Abbildung 2.2: CIA-Dreieck [18]

Unter *Vertraulichkeit* versteht man, dass nur autorisierte Benutzer Zugang zu geschützten Daten erhalten dürfen [18]. Es soll damit verhindert werden, dass unbefugte Benutzer Zugang zu Daten erhalten, für welche sie nicht berechtigt sind [18]. Die Vertraulichkeit der Informationen muss auch bei der Übertragung der Daten berücksichtigt werden, es muss also sichergestellt werden, dass bei der Übertragung der Daten, keine Informationen an Dritte weitergegeben werden oder abgefangen werden können [9].

Mit *Integrität* wird sichergestellt, dass die verfügbaren Daten gültig sind und man davon ausgehen kann, dass unautorisierte Benutzer die Daten nicht modifiziert haben [18]. Es soll damit verhindert werden, dass Informationen oder Systeme verändert werden, ohne zu wissen, wer diese Änderungen durchgeführt hat [18]. Zudem muss sichergestellt werden, dass bei der Übertragung der Daten keine Modifikationen stattfinden können [9].

Mit *Verfügbarkeit* soll sichergestellt werden, dass autorisierte Benutzer Zugang zu Ressourcen haben und unautorisierte Benutzer nicht in der Lage sind, diesen Zugang zu verweigern [18]. Das System soll zu jederzeit für autorisierte Benutzer verfügbar sein [18]. Bei der Übertragung der Daten muss sichergestellt werden, dass unautorisierte Benutzer eine Übertragung nicht verhindern können [9].

Um Confidentiality, Integrity und Availability zu gewährleisten gibt es verschiedene Möglichkeiten [18]. Für die Wahrung der Vertraulichkeit können etwa Daten verschlüsselt werden [20]. Für die Integrität können Daten mit einer Signatur versehen werden [20]. Um sicherzustellen, dass Systeme verfügbar sind, können sie als verteilte Systeme entwickelt werden [21]. In den nachfolgenden Abschnitten wird näher auf Verschlüsselung, Signatur und verteilte Systeme eingegangen.

Verschlüsselung

Hierbei ist das Ziel, dass Nachrichten zwischen zwei Parteien ausgetauscht werden können, ohne dass diese für Unbefugte leserlich sind [22]. Für die Umsetzung einer Verschlüsselung benötigen die beteiligten Personen einen gemeinsamen Schlüssel, welchen nur sie kennen [22]. Zuerst verschlüsselt der Sender seine Nachricht, den Klartext, mit dem geheimen Schlüssel [20]. Die verschlüsselte Nachricht, welche man Geheim- oder Chiffrentext nennt, ist nicht mehr lesbar und wird an den Empfänger gesendet [20]. Der Empfänger verfügt über den Schlüssel und kann dadurch den Geheimtext wieder in den Klartext umwandeln [20]. Diese Verschlüsselung nennt man symmetrische Verschlüsselung, da sowohl die Verschlüsselung als auch die Entschlüsselung mit demselben Schlüssel durchgeführt wird [20]. Abbildung 2.3 verdeutlicht dieses Verfahren noch einmal.

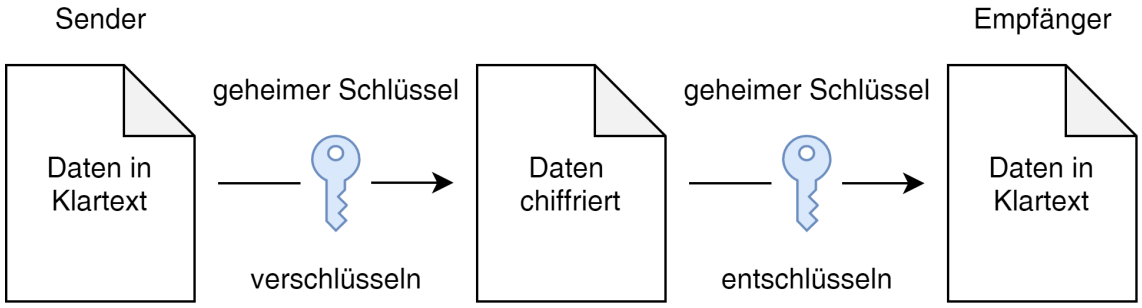


Abbildung 2.3: Prinzip der symmetrischen Verschlüsselung

Bei einer asymmetrischen Verschlüsselung, welche auch als Public-Key-Kryptografie bezeichnet wird, hat jeder Teilnehmer zwei Schlüssel, einen öffentlichen und einen geheimen Schlüssel [22]. Die öffentlichen Schlüssel können publiziert werden, während die geheimen Schlüssel privat aufbewahrt werden müssen [22]. Wenn nun der Sender an den Empfänger eine Nachricht senden will, so braucht der Sender den öffentlichen Schlüssel des Empfängers [20]. Der Sender verschlüsselt die Nachricht mit dem öffentlichen Schlüssel des Empfängers und übermittelt die Nachricht an den Empfänger [20]. Anschließend kann der Empfänger mit seinem geheimen privaten Schlüssel die Nachricht vom Sender entschlüsseln [20]. Abbildung 2.4 visualisiert den Prozess der asymmetrischen Verschlüsselung.

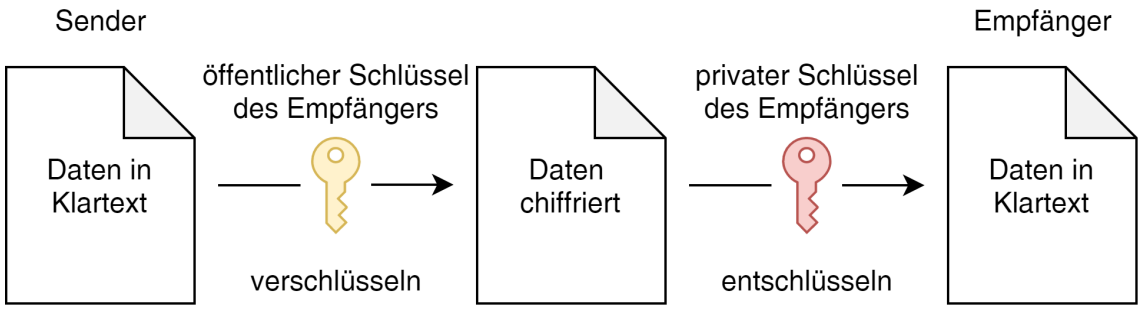


Abbildung 2.4: Prinzip der asymmetrischen Verschlüsselung

Auch wenn mithilfe der Verschlüsselung die Vertraulichkeit von Daten erhöht werden kann, so kann es trotzdem zu einem unbefugten Zugriff auf geheime Nachrichten kommen [20]. Generell muss man davon ausgehen, dass Angreifer die Ver- und Entschlüsselungsfunktion der Schlüssel wissen, daher muss bei der Erstellung eines geheimen Schlüssels darauf geachtet werden, dass der Schlüssel nicht durch das Wissen über den verwendeten Algorithmus herausgefunden werden kann [20]. Die Sicherheit der Verschlüsselung hängt nur von der Geheimhaltung des Schlüssels ab, jedoch nicht von der Geheimhaltung des Verschlüsselungsverfahrens [20]. Dies wird auch als das *Prinzip von Kerckhoff* bezeichnet [20].

Signatur

Mithilfe von Signaturen soll sichergestellt werden, dass die vorliegenden Daten nicht unentdeckt verändert wurden und somit die Integrität verletzt wurde [20]. Die Grundfunktion hinter Signaturen ist die *Hashfunktion*, hierbei wird eine beliebig lange Zeichenkette auf eine kleinere, charakteristische Zeichenkette komprimiert [20]. Eine Hashfunktion ist nur sehr schwer umkehrbar, auch wenn man den Hashwert einer Zeichenkette kennt, so ist es kaum möglich diesen wieder in den ursprünglichen Text umzuwandeln, da verschiedene Zeichenketten denselben Hashwert haben können [20]. Ein Hashwert dient hauptsächlich zur Überprüfung der Integrität [20].

Digitale Signaturen basieren zudem auf der *Public-Key-Kryptografie*, hierbei ist das Prinzip, wie bereits erwähnt, dass es einen öffentlichen und einen dazu passenden privaten Schlüssel gibt [22]. Der öffentliche Schlüssel zum Verifizieren der Nachricht wird zentral zur Verfügung gestellt und mithilfe des privaten Schlüssels kann eine Nachricht signiert werden [20]. In den meisten Fällen wird jedoch nicht eine ganze Nachricht signiert, sondern nur eine Art digitaler Fingerabdruck der Nachricht [20]. Eine digitale Signatur kann wie eine handschriftliche Unterschrift angesehen werden, welche von jedem verifiziert werden kann [20].

Im konkreten Fall wird der Hashwert einer Nachricht erstellt und dieser anschließend verschlüsselt [20]. Nun kann beim Empfangen der Nachricht der Hashwert entschlüsselt werden und mit dem tatsächlichen Hashwert der Nachricht überprüft werden [20]. Sollten diese zwei Werte nicht zusammenpassen, weiß man, dass die Nachricht von einer unbefugten Person modifiziert wurde [20].

Verteilte Systeme

Unter einem verteilten System versteht man mehrere, unabhängige Computer, welche miteinander über ein Netzwerk verbunden sind und so kommunizieren können [21]. Die Bestandteile des Systems sind meist räumlich voneinander getrennt [21]. Damit ein System als 100% verfügbar gilt, darf es im Jahr weniger als eine Stunde ausfallen. Da dies sehr schwer zu erreichen ist, versucht man die Verfügbarkeit mit 99% und aufwärts zu gewährleisten [21]. Die Verfügbarkeit eines Systems wird auch meist zwischen den Betreibern und dem Kunden in einem Vertrag (Service Level Agreement) festgehalten [21].

Mithilfe von verteilten Systemen können wichtige Bestandteile eines Systems redundant ausgelegt werden, wodurch eine höhere Verfügbarkeit sichergestellt werden kann [21]. Systeme sind meist so aufgebaut, dass sie aus verschiedenen, voneinander abgängigen Services bestehen [21]. Sollte es zu einem Problem in einem Bereich (Service) kommen, kann ein redundantes System die Aufträge des ausgefallenen Systems übernehmen und so meistens den Ausfall des ganzen Systems verhindern [21]. Redundanz hat besonders in der kritischen Infrastruktur, wie etwa dem Banken-Sektor, eine hohe Bedeutung [21].

2.5 Answer Set Programming

Answer Set Programming (ASP) ist ein deklaratives Problemlösungsparadigma, welches seine Wurzeln in der logischen Programmierung und dem nicht-monotonen Reasoning hat [23]. Mit ASP wird nicht ein klassisches Programm entwickelt, welches ein Problem löst, sondern es wird das Problem in Form von *Regeln* und *Fakten* modelliert und anschließend mithilfe eines *Reasoners* automatisch eine Lösung ermittelt [24]. Ein ASP-Programm wird mit einem Satz von Regeln und Fakten beschrieben, dabei ist die Semantik durch die *Antwortmengen* gegeben. Für das vorgegebene Problem kann es keine, eine oder mehrere Antwortmengen geben, welche die Lösung des Problems enthalten [23].

Viele Constraint-Satisfaction-Probleme lassen sich mit ASP lösen. Das Constraint-Satisfaction-Problem besteht darin, für eine beschriebenes Problem zu entscheiden, ob es möglich ist, jeder Variablen einen Wert aus einem speziellen Wertebereich zuzuweisen, sodass alle Bedingungen gleichzeitig erfüllt werden [25]. Dabei wird eine systematische Suche durchgeführt, wobei alle möglichen Kombinationen durchprobiert werden [25]. In dieser Arbeit ist die Zuteilung der Übungsaufgaben zu den Studierenden ein Constraint-Satisfaction-Problem.

DLV ist eine bewährtes ASP-System und im Weiteren wird sich diese Arbeit auf DLV beschränken. DLV ist ein deduktives Datenbanksystem, welches auf der deklarativen Programmiersprache *Datalog* basiert [26]. Zudem bietet es verschiedene Möglichkeiten für eine Kommunikation mit externen Systemen (Datenbank-Integration) an [26]. DLV besteht aus Regeln, Fakten, Aggregationen und Constraints [26]. Im nachfolgenden Abschnitt wird genauer auf die einzelnen Elemente eingegangen.

Regel und Fakt

Eine Regel hat folgenden Aufbau: *Regelkopf :- Regelrumpf* [26]. Der Regelkopf ist eine Disjunktion (Oder-Verknüpfung) von aussagenlogischen Variablen (Atomen) und der Regelrumpf ist eine Konjunktion (Und-Verknüpfung) von Literalen (negiertes oder nicht-negiertes Atom) [26]. Eine Regel kann in etwa so gelesen werden „wenn Regelrumpf wahr ist, dann ist Regelkopf wahr“ [26]. Wenn eine Regel ohne Regelrumpf steht, dann wird sie als Fakt bezeichnet, da sie eine unbedingte Wahrheit modelliert [26].

Aggregation

In DLV gibt es fünf Aggregationsfunktionen: #sum, #count, #times, #max und #min welche in etwa mit den Aggregationsfunktionen von SQL verglichen werden können [26]. Durch diese Funktionen können Operationen durchgeführt werden, welche mit Regeln und Fakten nicht abgedeckt werden können [26].

Starke Constraints

Eine Regel, welche keinen Regelkopf besitzt, wird als starkes Constraint (engl. Strong Constraint) bezeichnet [26]. Mit starken Constraints wird eine Bedingung modelliert, welche in keinem Fall zutreffen darf [24].

Schwache Constraints

Mithilfe von schwachen Constraints (engl. Weak Constraint) können in DLV Optimierungen eingebaut werden [24]. Im Gegensatz zu starken Constraints wird bei schwachen Constraints das Implikationssymbol von :- auf :$\sim$ geändert [26]. Zudem sollte die Bedingung zwar nicht erfüllt werden, es ist jedoch nicht zwingend notwendig [26]. Bei schwachen Constraints können auch Gewichtungen sowie Prioritätsstufen hinzugefügt werden, diese werden mit eckigen Klammern angegeben [26].

Datenbank-Integration

DLV kann auf mehrere Arten mit externen Systemen kommunizieren:

- Verwendung eines Java-Wrappers, um DLV direkt aus einem Java-Programm aufzurufen [26].
- Verwendung eines ODBC-Interfaces, um auf relationale Datenbanken zugreifen zu können [26].
- Verwendung von DLVEX, um mit C++ auf DLV zugreifen zu können [26].

Kapitel 3

Architektur und konzeptuelles Modell

In diesem Kapitel wird auf die Architektur und das konzeptuelle Modell der entwickelten Anwendung eingegangen. Dabei wird zuerst in Abschnitt 3.1 beschrieben, wie die einzelnen Komponenten aufgebaut sind und wie diese zusammenhängen. Anschließend wird in Abschnitt 3.2 auf das konzeptuelle Modell eingegangen.

3.1 Architektur und Plattform

Es wurde ein experimenteller Standalone-Prototyp zur Absolvierung individuell erstellter Aufgabenlisten entwickelt. Dabei wird bei der wissensbasierten Aufgabenzuteilung die Privatsphäre der Studierenden gewahrt. Die entwickelte Anwendung gliedert sich in drei verschiedene Backend-Services (Maintenance-Service, Task-Assignment-Generator-Service, Task-Service) mit jeweils einer dazugehörigen Datenbank. Wie in Abschnitt 2.4 Privacy by Design beschrieben soll bei Systemen, welche verschiedene Funktionen haben, darauf geachtet werden, dass jeweils nur die relevanten Daten für die einzelnen Bereiche verfügbar sind. Daher wurde bei der Entwicklung der Anwendung entschieden, dass jedes Service eine eigene Datenbank erhält und nur das jeweilige Service auf seine eigene Datenbank Zugriff hat. Sollte jedoch ein Service Daten aus einer anderen Datenbank benötigen, so können diese Daten beim zugehörigen Service über eine Schnittstelle angefordert werden.

Die Anwendung wurde mit einer Microservice-Architektur umgesetzt. Daher wurden die Aufgaben der einzelnen Services strikt getrennt. Jedes Service ist für einen Bereich zuständig: Für die verschiedenen Verwaltungstätigkeiten wurde das *Maintenance-Service* entwickelt, es übernimmt alle Aufgaben für die Registrierung und Anmeldung der Studierenden. Es erstellt Token, welche für die Anmeldung und Überprüfung der Session notwendig sind.

Zudem verwaltet das Maintenance-Service die Lernfortschrittsbestätigungen, welche die Studierenden für das erfolgreiche Absolvieren einer Aufgabe erhalten. Des Weiteren bietet es für die Studierenden eine Möglichkeit zur Sicherung der Daten an. Auf diesen Punkt wird im nachfolgenden Abschnitt noch näher eingegangen.

Das Task-Service übernimmt alle Funktionalitäten in Bezug auf die Erstellung, Verwaltung und Durchführung von Aufgaben. Eine Aufgabe besteht dabei aus einer zu lösenden Fragestellung, einer zugeordneten Aufgabenfamilie, welche einem Datenbank-Schema entspricht, und einer Menge von Aufgabenklassen, welche durch das Absolvieren der Aufgabe erlernt werden. Das letzte Service, das Task-Assignment-Generator-Service (TAG-Service), enthält alle Funktionen, welche sich direkt auf die Studierenden beziehen. Diese Funktionen sind etwa das Erstellen und Verwalten der verschiedenen User-Präferenzen im Zusammenhang mit der Aufgabenliste, sowie die Generierung einer individuellen Aufgabenliste. Im Gegensatz zu den anderen beiden Services läuft das TAG-Service nur lokal bei den Studierenden. In den nachfolgenden Abschnitten werden die Architektur und das konzeptuelle Modell der einzelnen Services beschrieben.

Das Maintenance-Service und das Task-Service verwenden eine PostgreSQL Datenbank, im Gegensatz dazu ist die Datenbank des Task-Assignment-Generator-Service eine H2 In-Memory-Datenbank und läuft nur lokal bei den Studierenden. Dieser Ansatz wurde gewählt, damit die vertraulichen Daten über die Studierenden nur von ihnen selbst abgerufen werden können. Damit es jedoch nicht aufgrund der In-Memory-Datenbank zu einem Datenverlust kommen kann, werden sie als verschlüsseltes Backup in der Datenbank des Maintenance-Services gespeichert. Wenn die Anwendung neu gestartet wird, wird das Backup aus dem Maintenance-Service geladen, entschlüsselt und wieder in die H2 Datenbank gespeichert.

Als Benutzeroberfläche wurde ein zentrales Frontend entwickelt, dass die Daten von den verschiedene Backend-Services erhält. Hierbei erfolgt eine Trennung in User-Sicht und Admin-Sicht. Diese Trennung ergibt sich durch die zugewiesenen Rollen der Benutzer. User können somit nur Aktionen durchführen, für welche sie die notwendigen Rechte haben. Ein DLV-Programm ist für die Generierung der Aufgabenliste zuständig und läuft wie das Task-Assignment-Generator-Service nur lokal bei den Studierenden. Dieses Service ist zudem das einzige Service, welches auf das DLV-Programm zugreifen kann. Da jedoch das DLV-Programm verschiedene Informationen für die Generierung der Aufgabenliste benötigt, werden diese Daten vom Task-Assignment-Generator-Service bei den anderen Services angefordert und an das DLV-Programm weitergegeben.

Nachfolgende Abbildung 3.1 zeigt die beschriebene Architektur. Das Task-Assignment-Generator-Service kann sowohl mit dem Maintenance-Service als auch mit dem Task-Service kommunizieren. Da das Task-Assignment-Generator-Service jedoch nur lokal bei den Studierenden läuft, können das Maintenance-Service und Task-Service zwar eine Antwort zurücksenden, wenn das Task-Assignment-Generator-Service etwas anfordert, jedoch können sie nicht selber eine Anfrage an das Task-Assignment-Generator-Service senden. Das Maintenance-Service und das Task-Service laufen zentral, daher können diese zwei Services sowohl eine Anfrage an das andere Service senden als auch auf eine Anfrage reagieren. Alle

Services haben ein gemeinsames Frontend, den Task-Assignment-Generator-Demonstrator, mit welchem die User mit den verschiedenen Services interagieren können.

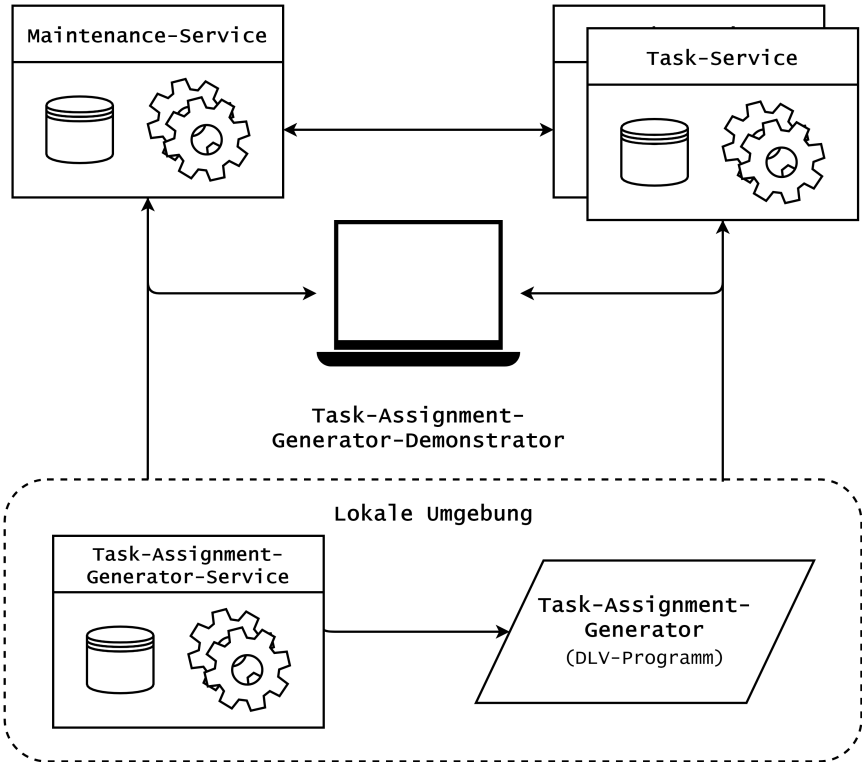


Abbildung 3.1: Gesamtarchitektur Prototyp

3.2 Konzeptuelles Modell

In diesem Abschnitt wird das konzeptuelle Modelle des entwickelten Standalone-Prototypen beschrieben. Da jedes Service eine eigenen Datenbank besitzt, wird in den einzelnen Abschnitten nur auf die relevanten Komponenten des jeweiligen Services eingegangen. Abbildung 3.2 zeigt das gesamte konzeptuelle Modell des Standalone-Prototypen. In Anhang B befinden sich für eine genauere Beschreibung der einzelnen Datenbanken das jeweilige logische Datenbankschema als SQL-DDL und den dazugehörigen Skripten. Im Folgenden werden die Elemente des konzeptuellen Modells pro Service genauer erläutert.

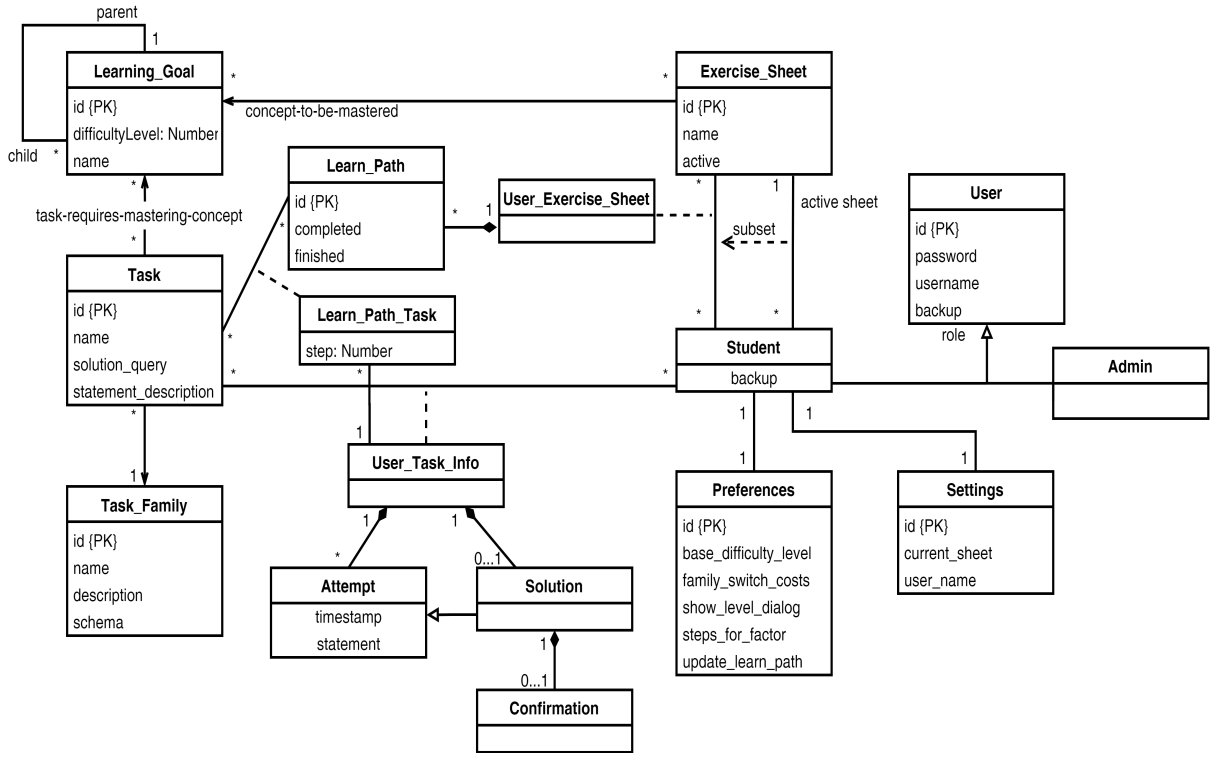


Abbildung 3.2: Konzeptuelles Modell

Maintenance-Service

Das Maintenance-Service ist für die allgemeine Verwaltung der Aufgaben und der User zuständig. Hier werden alle registrierten User mit der dazugehörigen Rolle gespeichert. Jedem User wird genau eine Rolle zugeordnet, jedoch können mehrere User dieselbe Rolle besitzen. Je nach zugeordneter Rolle unterscheiden sich auch die Aktionen die ausgeführt werden können. Es gibt eine Rolle für *Student* und eine für *Admin*. Im konzeptuellen Modell werden Rollen mittels Vererbung gekennzeichnet (role). Administratoren (Admin) können Aufgabensammlungen (Exercise_Sheet) anlegen, welche eine oder mehrere Aufgabenklassen (Learning_Goal) enthält und welche von den Studierenden erarbeitet werden sollen. Es kann jedoch immer nur eine Aufgabensammlung als *aktiv* markiert werden, diese ist die aktuell zu lösende Aufgabensammlung. Damit Studierende bestätigen können, dass sie die vorgegebenen Aufgabenklassen erfolgreich erlernt haben, können sie eine Lernfortschrittsbestätigung (Confirmation) abgeben. Zu dieser gehören noch weitere Komponenten, auf welche jedoch im Abschnitt des Task-Services und des Task-Assignment-Generator-Services näher eingegangen wird.

Task-Service

Wie bereits beschrieben, ist das Task-Service für die Verwaltung und Überprüfung von Aufgaben zuständig. Hier werden die eingegeben Queries zu einer Aufgabe entweder ausgeführt *Attempt* (Studierende sehen nur das Ergebnis) oder eine Lösung abgegeben *Solution*. Daher steht im Zentrum des konzeptuellen Modells die Aufgabe (Task). Jede Aufgabe gehört genau zu einer Aufgabenfamilie (Task_Familie). Es kann jedoch zu einer Aufgabenfamilie eine Vielzahl an Aufgaben gehören. Jede Aufgabe erfüllt eine oder mehrere Aufgabenklassen (Learning_Goal), das heißt mit der Absolvierung der Aufgabe erreichen Studierenden diese Aufgabenklassen (Lernziele). Eine Aufgabenklasse kann über verschiedene Aufgaben erreicht werden.

Task-Assignment-Generator-Service

Die Hauptaufgabe des Task-Assignment-Generator-Services ist die Erstellung einer Aufgabenliste (Learn_Path) für die aktive Aufgabensammlung (Exercise_Sheet). Eine Aufgabenliste besteht dabei aus einzelnen Schritten, den User-Task-Info-Objekten, in welchen gespeichert wird, welche Aufgabe zu absolvieren ist, welcher Aufgabenfamilie die Aufgabe zugeordnet ist, ob diese bereits geschafft wurde, und an welcher Stelle sie in der vorgegebenen Aufgabenliste steht. Um den Studierenden genauere Informationen zu ihrer vergangenen Leistung geben zu können, wird bei jeder versuchten Lösung einer Aufgabe ein dazugehöriges Log-Eintrag (Attempt) gespeichert. Hier wird mitgespeichert, um welches User-Task-Info-Objekt es sich handelt und welcher Query, wann eingegeben wurde. So kann den Studierenden ihre Lern-Entwicklung gezeigt werden. Diese Daten werden als vertrauliche Informationen angesehen, daher werden sie nur lokal bei den Studierenden gespeichert und sind für keine anderen Services zugänglich.

Das Task-Assignment-Generator-Service hat des Weiteren noch die Klassen *Preferences, Settings* und *Confirmation*. In *Preferences* werden die verschiedene Einstellungen der Studierenden gespeichert. So können Sie etwa den bevorzugten Schwierigkeitsgrad angeben, ob sich die Aufgabenfamilien unterscheiden dürfen, oder ob nach dem erfolgreichen Absolvieren einer Aufgabe ein Dialog für die Anpassung der Aufgabenliste geöffnet werden soll. In der Klasse *Settings* wird gespeichert, was die zuletzt aktive Aufgabensammlung war, um bei einer Änderung den User informieren zu können. Dies geschieht etwa, wenn die ÜbungsleiterInnen einen neue Aufgabensammlung als aktiv markieren und neue Aufgabenklassen von den Studierenden erarbeitet werden müssen. In diesem Fall bekommen die Studierenden eine Nachricht, dass die von ihnen generierte Aufgabenliste für eine veraltete Aufgabensammlung ist und sie eine neue individuelle Aufgabenliste generieren müssen. Die letzte Klasse ist *Confirmation*. Für jede erfolgreich erlernte Aufgabenklasse, erhalten die Studierenden eine *Confirmation*. Diese Bestätigungen können Studierende an das Maintenance-Service senden, um die Aufgabenklassen einer Aufgabensammlung erfolgreich abzuschließen.

3.3 Diskussion Zielerreichung

Im nachfolgendem Abschnitt wird auf die in Abschnitt 1.4 beschriebenen Ziele der Arbeit eingegangen. Dabei wird jedes Ziel auf seine konkrete Umsetzung in Bezug auf die Architektur und das konzeptuelle Modell beschrieben.

A1 Individualisierung Aufgabenliste

Für das Ziel *Individualisierung Aufgabenliste* ist vor allem die Klasse *Preferences* im konzeptuellen Modell (Abbildung 3.2) verantwortlich. Nachfolgend wird auf die einzelnen Punkte näher eingegangen.

A1.1 Einstellungsmöglichkeit Aufgabenfamilie
Für die Erreichung des Zieles *Einstellungsmöglichkeit Aufgabenfamilie* wird zum einen im Task-Service jede Aufgabe einer Aufgabenfamilie zugeordnet. Zum anderen hat das Task-Assignment-Generator-Service in der Klasse *Preferences* die Variable *family_switch_costs*. Hier können die Studierenden einen Wert festlegen, mit welchem gesteuert wird, ob bei der Generierung einer Aufgabenliste immer dieselbe oder unterschiedliche Aufgabenfamilien verwendet werden sollen.

A1.2 Einstellungsmöglichkeit Schwierigkeitsgrad
Die Erfüllung der Anforderung *Einstellungsmöglichkeit Schwierigkeitsgrad* wurde ebenso mit der Klassen *Preferences* umgesetzt. Diese hat die Variable *base_difficulty_level* hat, mit welchem die Studierenden ihren bevorzugten Schwierigkeitsgrad für eine Aufgabe festlegen können. Bei der Generierung der Aufgabenliste wird dieser Wert dem Task-Assignment-Generator übermittelt.

A1.3 Anpassung Aufgabenliste
Des Weiteren kann eine Einstellungsmöglichkeit für die Anpassung der Aufgabenliste in der Klasse *Preferences* (Abbildung 3.2) gefunden werden. Mit den Variablen *show_level_dialog* und *update_learn_path* können die Studierenden einstellen, ob sich nach der erfolgreichen Absolvierung einer Aufgabe einen Dialog öffnen soll, bei welchem sie angeben können, ob die Aufgabenliste *zu leicht*, *zu schwer* oder *genau richtig* war. Zum anderen können sie entscheiden, ob die Aufgabenliste bei einer Auswahl von *zu leicht* oder *zu schwer* neu generiert werden soll.

A2 Wissensstand der Studierenden

Wie im konzeptuellen Modell ersichtlich ist, werden viele verschiedene Informationen über die Studierenden gespeichert. Mit der Anforderung *Wissensstand der Studierenden* sollen die Studierende Zugang zu ihren persönlichen Informationen haben.

A2.1 Wissensstand Aufgabenklasse

Um die Anforderung *Wissensstand Aufgabenklasse* erfüllen zu können, erhalten die Studierenden eine Lernfortschrittsbestätigung, sobald eine Aufgabe erfolgreich absolviert wurde. Es wird für jede Aufgabenklasse eine eigene Bestätigung erstellt, daher können Studierende mit der erfolgreichen Absolvierung einer Aufgabe auch mehrere Bestätigungen erhalten. Diese Bestätigung wird mit der Klasse *Confirmation* abgebildet. In Kapitel 5 Frontend wird die dazugehörige Hierarchie beschrieben, welche den Studierenden grafisch dargestellt wird.

A2.2 Wissensstand und Aufgabenlistengenerierung

Bereits erlerntes Wissen beeinflusst neue Aufgabenlistengenerierungen. Für jede bereits absolvierte Aufgabe erhalten Studierende eine Lernfortschrittsbestätigung (Klasse *Confirmation*). Damit Studierende eine Aufgabenklasse nicht zweimal absolvieren müssen, werden alle bereits absolvierten Aufgabenklasse dem *Task-Assignment-Generator* übermittel und bei der Generierung neuer Aufgabenlisten berücksichtigt.

A2.3 Vergangene Abfragen

Damit Studierende aus ihren Fehlern lernen und ihren Lernerfolg nachverfolgen können, werden ihre *Lösungsverusche* gespeichert. Bei jeder Abfrage, welche für eine Aufgabe eingegeben wurde, wird ein Lösungsversuch in der Klasse *Attempt* gespeichert. Dabei wird auch mitgespeichert, ob die Abfrage nur *ausgeführt* oder *übermittelt* wurde. Bei Lösungsversuchen, welche *übermittelt* wurden, ist auch noch die Klasse *Solution* relevant, da bei einer erfolgreichen Absolvierung eine Bestätigung (*Confirmation*) über die erreichten Lernziele ausgestellt wird.

A3 Wahrung der Privatsphäre

Für die Erreichung des Zieles *Wahrung der Privatsphäre* wurde bereits beim Entwickeln der Architektur darauf geachtet, dass die Daten über die Studierenden von den allgemeinen Daten getrennt werden. In Abbildung 3.1 sieht man, dass das Task-Assignment-Generator-Service sowie die dazugehörige Datenbank und der Task-Assignment-Generator nur lokal bei den Studierenden verfügbar sind.

A3.1 Privates Wissen über die Studierenden

Wie in Abbildung 3.1 ersichtlich ist, hat jedes Service seine eigene Datenbank und die Funktionsbereiche wurden voneinander getrennt. Das Task-Service ist nur für die Verwaltung von Aufgaben zuständig und speichert keine Informationen zu Studierenden. Das Maintenance-Service speichert nur Informationen, welche für das Session-Handling und das Ausstellen von Lernfortschrittsbestätigungen notwendig sind. Alle User-bezogenen Daten werden nur im lokalen Task-Assignment-Generator-Service gespeichert und sind nur für die Studierenden selbst zugänglich.

A3.2 Lernfortschrittsbestätigung

Die Lernfortschrittsbestätigung *Confirmation* wird in der Datenbank des Task-Assignment-

Generator-Services gespeichert. Um eine Aufgabenklasse erfolgreich zu absolvieren, wird zudem dem Maintenance-Service eine Bestätigung übermittelt. Diese Lernfortschrittsbestätigung enthält jedoch nur Informationen, welche wirklich notwendig sind, um bestätigen zu können, dass eine Aufgabe abgeschlossen wurde. Diese besteht aus folgenden Daten: der User, welcher die Aufgabenklasse erlernt hat, die Aufgabensammlung, die Aufgabe, das eingegebene Lösungsstatement, die Uhrzeit sowie die erreichten Aufgabenklassen. Es wird jedoch nicht mitgespeichert wie viele Versuche notwendig waren oder welche Vorschritte absolviert wurden.

Zusammenfassung

In diesem Kapitel wurde die Architektur des entwickelten Standalone-Prototypen beschrieben. Dazu wurde auf die einzelnen Teilbereiche (Services) näher eingegangen und die Zusammenhänge erläutert. Des Weiteren wurde das konzeptuelle Modell beschrieben und näher auf die einzelnen Klassen eingegangen und beschrieben in welchem Service sie verwendet werden. Zum Schluss wurden die in Abschnitt 1.4 beschriebenen Anforderungen in Bezug auf die Architektur und das konzeptuelle Modell diskutiert.

Kapitel 4

Backend

In diesem Kapitel wird das entwickelte Backend näher beschrieben. Dabei werden die konkreten Implementierungsdetails des Maintenance-Service, Task-Assignment-Generator-Service und Task-Service erläutert. Zum Teil beginnen die Endpunkte mit *v1*, diese Architekturentscheidung wurde aus Versionierungsgründen entschieden. Der Standalone-Prototyp besteht aus drei verschiedenen Services, welche sich unterschiedlich voneinander verändern können. Um dennoch sicherzustellen, dass die Funktionen weiterhin zur Verfügung gestellt werden können, beginnen die meisten Endpunkt mit *v1*. Sollten große Änderungen in einem Service durchgeführt werden, mit welchen sich auch die Schnittstellen ändern, so kann man diese Endpunkte z.B. mit *v2* beginnen und Schrittweise die anderen Services auf die neuen Schnittstellen umstellen. Zudem benötigt man für alle Endpunkte, welche mit v1 beginnen, eine Authentifizierung um darauf zugreifen zu können.

In den nachfolgenden Abschnitten werden nun die verschiedenen Funktionsbereiche der Services mithilfe einer tabellarischen Darstellung näher beschrieben. Für ein besseres Verständnis wird teilweise auch auf konkrete Code-Ausschnitte eingegangen. In Kapitel 5 werden die dazugehörigen Visualisierungen näher beschrieben.

4.1 Maintenance-Service

Das Maintenance-Service kümmert sich um die Erstellung und Verwaltung von Usern sowie ihren dazugehörigen Rollen. Sobald ein User sich mithilfe des Maintenance-Services registriert hat, wird ein anonymer Session-Token erstellt, mit welchem sich der User nun beim Task-Assignment-Generator identifizieren kann und, falls vorhanden, werden bereits gespeicherte Daten aus der Maintenance-Service Datenbank geladen. Das Maintenance-Service ist für die allgemeine Verwaltung der Session zuständig, es überprüft etwa mit welcher Rolle welche Aktionen ausgeführt werden dürfen. Des Weiteren ist dieses Service für die Erstellung und Verwaltung der Aufgabensammlungen zuständig. Mithilfe einer Aufgaben-

sammlung können die Lehrenden vorgeben, welche Aufgabenklassen von den Studierenden erarbeitet werden sollen. Für eine einfachere Verwaltung können mehrere Aufgabensammlungen gleichzeitig existieren. Damit jedoch die Studierenden wissen, welche Aufgabenklassen aktuell zu üben sind, kann eine Aufgabensammlung auf *aktiv* gesetzt werden. Sobald Studierende eine Aufgabenklasse der Aufgabensammlung erreicht haben, erhalten sie eine Lernfortschrittsbestätigung. Diese kann nun mithilfe des Maintenance-Service den Lehrenden übermittelt werden. Um Lernfortschrittsbestätigungen nicht fälschen zu können, werden sie vom Maintenance-Service signiert, um die Echtheit der Bestätigung zu gewährleisten.

Schnittstellen

Für folgende Funktionsbereiche stellt das Maintenance-Service Endpunkte zur Verfügung:

- Authentifizierung
- Aufgabensammlung
- Lernfortschrittsbestätigung
- User-Verwaltung

In den nachfolgenden Abschnitten wird auf diese Endpunkte näher eingegangen.

Authentifizierung

Die Endpunkte für die Authentifizierung beschäftigen sich mit der Registrierung und dem Login von Benutzern. Dabei wird unterschieden, ob dieser Benutzer bereits einen aktiven Account hat oder nicht:

Endpunkt	**Aktion**
/auth/register	Noch nicht registrierte Benutzer können sich über diesen Endpunkt einen User erstellen.
/auth/login	Registrierte Benutzer können sich mit ihren Zugangsdaten einloggen und erhalten einen Session-Token.

Tabelle 4.1: Schnittstelle Authentifizierung

Damit ein User überhaupt auf die Anwendung zugreifen kann, benötigt er einen *JWT-Token* für die Authentifizierung. Daher wird beim Login-Prozess für den aktiven User ein *JWT-Token* mithilfe des HS256 Algorithmus erstellt. Dieser Token enthält die Gültigkeitsdauer und die Rolle des Users, mit welcher die Berechtigung verwaltet wird. Der Code-Ausschnitt 4.1 zeigt diesen Prozess:

```
public String createToken (String role) throws LoginException {
    Algorithm algorithm = Algorithm.HMAC256(secretKey);
    Date now = new Date();
    Date validity = new Date(now.getTime() + validityInMilliseconds);
    try {
        String token = JWT.create()
            .withExpiresAt(validity)
            .withClaim("role", role)
            .sign(algorithm);
        return token;
    } catch (JWTCreationException exce) {
        throw new LoginException(exce.getMessage());
    }
}
```

Listing 4.1: JWT-Token erstellen

Aufgabensammlung

Die Endpunkte für Aufgabensammlung sind für die Erstellung und Verwaltung der verschiedenen Aufgaben zuständig. Zudem beschäftigen sie sich auch mit der Abfrage aller Aufgabensammlungen bzw. der aktuell aktiven Aufgabensammlung, welche von den Studierenden absolviert werden muss. Eine Aufgabensammlung besteht aus einer Sammlung von Aufgabenklassen, welche durch die Absolvierung von verschiedenen Aufgaben erreicht werden können. Für die Verwaltung der Aufgabensammlungen gibt es folgende Endpunkte:

Endpunkt	**Aktion**
/v1/exercisesheet/{id}	Gibt entweder alle Aufgabensammlungen oder mithilfe der übermittelten ID die angeforderte Aufgabensammlung zurück. Zudem kann mit *POST* oder *PUT* eine Aufgabensammlung erstellt oder aktualisiert werden. *DELETE* löscht die Aufgabensammlung der übermittelten ID.
/v1/exercisesheet/active	Gibt die zurzeit aktive Aufgabensammlung zurück.

Tabelle 4.2: Schnittstelle Aufgabensammlung

Es kann immer nur eine Aufgabensammlung als *aktiv* markiert sein. Daher muss beim Speichern einer Aufgabensammlung überprüft werden, ob diese als *aktiv* markiert wurde, wenn dies der Fall ist, muss die vorherige aktive Aufgabensammlung als inaktiv markiert werden. Folgender Code-Ausschnitt zeigt diesen Ablauf:

```
public void saveExerciseSheet(ExerciseSheetDTO exerciseSheetDTO) {
    ExerciseSheet result = mapExerciseSheetDTO(exerciseSheetDTO);
    if(exerciseSheetDTO.isActive()){
        ExerciseSheet oldActiveSheet = exerciseSheetRepository.
        findByActiveTrue();
        oldActiveSheet.setActive(false);
        exerciseSheetRepository.save(oldActiveSheet);
    }
    exerciseSheetRepository.save(mapExerciseSheetDTO(exerciseSheetDTO));
}
```

Listing 4.2: Aufgabensammlung speichern

Lernfortschrittsbestätigung

Die Endpunkte für die Lernfortschrittsbestätigung beschäftigten sich mit der Abgabe einer Bestätigung an die Lehrenden. Es wird dabei übermittelt, welche Aufgabenklassen der Aufgabensammlung mit welcher Aufgabe erreicht wurde. Für jede erreichte Aufgabenklasse erhalten Studierende eine Lernfortschrittsbestätigung. Sollten nicht alle Aufgabenklassen einer Aufgabensammlung erreicht werden können, so können Studierende auch nur für einen Teil der Aufgabenklassen eine Bestätigung abgeben. Dies ermöglicht es eine Teilbewertung zu erhalten.

Endpunkt	**Aktion**
/v1/confirm/{id}	Übermittelt alle gesendeten Lernfortschrittsbestätigungen. Wenn eine ID mitgegeben wird, so wird die angeforderte Lernfortschrittsbestätigung zurückgegeben. Mithilfe eines *POST* Request kann eine neue Lernfortschrittsbestätigung übermittelt werden.
/v1/confirm/overview/{id}	Gibt eine Übersicht aller gespeicherten Lernfortschrittsbestätigungen zu einer gewissen Aufgabensammlung zurück.

Tabelle 4.3: Schnittstelle Lernfortschrittsbestätigung

Damit sichergestellt wird, dass die abgegebene Lernfortschrittsbestätigung nicht gefälscht wurde, wird diese auf ihre Gültigkeit überprüft. Dazu wurde vom Task-Service das State-

ment des Users mit dem *Private Key* verschlüsselt. Das Task-Service bietet nun dem Maintenance-Service den *Public Key* an, um das Statement zu entschlüsseln. Sollte dies nicht möglich sein, da das Statement nicht vom Task-Service signiert wurde, wird die Bestätigung als gefälscht angesehen und nicht anerkennt. Der Code-Ausschnitt 4.3 zeigt das Vorgehen.

```
public String verify(String query) throws InvalidSignatureException {
    try {
        PublicKey publicKey = asymmetricCryptography
        .createPublicKey(securityClient.getPublicKey();
        String result = asymmetricCryptography
        .decryptText(query, publicKey);
        return result;
    } catch (NoSuchAlgorithmException | InvalidKeySpecException |
             NoSuchPaddingException | UnsupportedEncodingException |
             IllegalBlockSizeException | BadPaddingException |
             InvalidKeyException e){
    throw new InvalidSignatureException("Signature is not valid");
    }
}
```

Listing 4.3: Integrität der Bestätigung prüfen

User-Verwaltung

Der letzte Aufgabenbereich des Maintenance-Service ist die User-Verwaltung. Hier können berechtigte User alle registrierten Benutzer einsehen und etwa die zugeteilte Rolle ändern. Zudem werden mithilfe des *user* Endpunkts die verschlüsselten Backups der Studierenden an die zentrale Datenbank des Maintenance-Services übermittelt. Die nachfolgende Tabelle zeigt die dazugehörigen Schnittstellen.

Endpunkt	**Aktion**
/v1/user/{id}	Gibt alle registrierte Benutzer zurück. Zusätzlich kann ein spezieller User angefordert werden, indem die ID mitgegeben wird. *DELETE* löscht den übermittelten User und mit *PUT* kann ein User aktualisiert werden. Es gibt keinen *POST* Endpunkt da die Erstellung eines Users über /auth/register erfolgt.
/v1/user/backup	Speichert das Backup der Lokalen Datenbank eines Users. Die Information um welchen User es sich handelt, befindet sich im übermittelten Objekt, welches auch das verschlüsselte Backup enthält.

Tabelle 4.4: Schnittstelle User-Verwaltung

Auch wenn das Maintenance-Service mit dem Backup der Studierenden in Kontakt kommt, so kann es dennoch keine Informationen vom Backend auslesen. Jedes Backup von den Studierenden wird individuell verschlüsselt und kann auch nur wieder mit dem individuellen privaten Schlüssel entschlüsselt werden.

4.2 Task-Service

Der Grundgedanke beim Task-Service ist, dass dieses Service für verschiedene Wissensgebiete speziell entwickelt wird. In dieser Arbeit wurde das Task-Service für die Datenbankabfragesprache SQL umgesetzt. Sollte jedoch ein anderer Aufgabenbereich erlernt werden, so kann das Task-Service ausgetauscht werden. Das aktuelle Task-Service beschäftigt sich mit der Erstellung und Verwaltung von Aufgaben, den dazugehörigen Aufgabenfamilien und den möglichen Aufgabenklassen. Zudem führt das Task-Service die konkrete Überprüfung der eingegebenen Abfragen durch und gibt den Studierenden Feedback zu ihren Lösungen. Sobald eine Aufgabe erfolgreich abgeschlossen wurde, wird dafür eine Lernfortschrittsbestätigung erstellt. Um diese vor Fälschungen zu schützen, wird sie vom Task-Service signiert. Dadurch kann die Integrität gewährleistet werden.

Schnittstellen

Für folgende Funktionsbereiche besitzt das Task-Service eigene Endpunkte:

- Verwaltung der Aufgaben
- Verwaltung der Aufgabenfamilien
- Verwaltung der Aufgabenklassen
- Lösungsabfrage
- Security: Signierung von Lernbestätigungen

In den nachfolgenden Abschnitten werden diese Endpunkte näher beschrieben.

Verwaltung der Aufgaben

Die Grundaufgabe des Task-Services ist die Verwaltung der Tasks (Aufgaben). Eine Aufgabe beinhaltet eine Fragestellung, welche von den Studierenden gelöst werden muss. Für die Überprüfung wird zudem eine Beispiellösung angegeben, mit welcher die Lösung der

Studierenden verglichen wird. Mit jeder Aufgabe werden verschiedene Aufgabenklassen erreicht. Zudem gehört jede Aufgabe zu einer Aufgabenfamilie, welche ein Datenbankschema beschreibt. Der Endpunkt für die Aufgaben ermöglicht es den Lehrenden neue Aufgaben anzulegen, zu bearbeiten und zu löschen. Tabelle 4.5 beschreibt diesen Endpunkt.

Endpunkt	**Aktion**
/v1/task/{id}	Liefert alle gespeicherten Aufgaben zurück. Es kann zudem eine einzelne Aufgabe angefordert werden, indem die ID mitgegeben wird. Mit *POST* kann eine neue Aufgabe und allen dazugehörigen Daten (wie etwa die Aufgabenklassen) angelegt werden. Mit *DELETE* ohne ID werden alle Aufgaben gelöscht und mit einer ID wird nur die speziell ausgewählte Aufgabe gelöscht.

Tabelle 4.5: Schnittstelle Aufgaben

Verwaltung der Aufgabenfamilien

Für ein besseres Verständnis wird jede Aufgabe einer Aufgabenfamilie zugeordnet. Alle Aufgaben der gleichen Aufgabenfamilie verwenden dasselbe Datenbankschema. Studierende können in der Anwendung einstellen, dass sie immer dieselbe Aufgabenfamilie verwenden möchten. Dies hat den Vorteil, dass nicht mehrere Datenbank-Schematas erlernt werden müssen. In Tabelle 4.6 wird der Endpunkt für die Aufgabenfamilien beschrieben.

Endpunkt	**Aktion**
/v1/family/{id}	Hiermit können alle verfügbaren Aufgabenfamilien abgefragt werden. Des Weiteren kann eine Familie angefordert werden, in dem die ID übermittelt wird. Analog zum *Task-Endpunkt* kann mit *POST* und *DELETE* eine Aufgabenfamilie angelegt oder gelöscht werden.

Tabelle 4.6: Schnittstelle Aufgabenfamilien

Verwaltung der Aufgabenklassen

Wie in Abschnitt 1.3 beschrieben, wurde eine Generalisierungshierarchie erstellt, welche den Sprachkonstrukten von SQL entspricht. Jedes Element dieser Hierarchie wird nun als eine *Aufgabenklasse* angesehen. Mit jeder Aufgabe, welche die Studierenden absolvieren können, werden verschiedene Aufgabenklassen erreicht. Die Endpunkte für die Aufgabenklassen ermöglichen es den Lehrenden neue Aufgabenklassen zu erstellen und zu verwalten. Zudem können die Aufgabenklassen abgefragt werden, welche bei einer speziellen Aufgabe erlernt werden. In Tabelle 4.7 werden die verschiedenen Schnittstellen beschrieben.

Endpunkt	**Aktion**
/v1/learningGoals/{id}	Übermittelt alle gespeicherten Aufgabenklassen. Wenn eine ID angegeben wird, wird die angeforderte Aufgabenklasse zurückgeliefert. Mit *POST* kann eine neue Aufgabenklasse angelegt und mit *DELETE* kann diese gelöscht werden.
/v1/learningGoals/task/{taskId}	Liefert alle Aufgabenklasse einer Aufgabe zurück.

Tabelle 4.7: Schnittstelle Aufgabenklassen

Lösungsabfrage

Nachdem Studierende eine Lösung für eine Aufgabe erarbeitet haben, können sie diese mithilfe der Schnittstellen für die Lösungsabfrage entweder *ausführen* oder *überprüfen* lassen. Nachfolgende Tabelle 4.8 zeigt diese Funktionen.

Endpunkt	**Aktion**
/v1/statement/check	Mit diesem Endpunkt wird überprüft, ob die eingegebene Lösung korrekt ist. Dabei wird dem Endpunkt im *body* die Lösung sowie die absolvierte Aufgabe übermittelt und überprüft, ob sie korrekt ist. Sollte dies der Fall sein, wird für die erreichten Aufgabenklassen eine Bestätigung ausgestellt und signiert.
/v1/statement/execute	Auch bei diesem Endpunkt werden im *body* die Lösung sowie die versuchte Aufgabe geliefert. Die übermittelte Lösung wird ausgeführt und das Ergebnis im Frontend angezeigt. Dies soll als Hilfestellung dienen.

Tabelle 4.8: Schnittstelle Lösungsabfrage

Bei der Überprüfung des Statements werden mehrere Schritte durchgeführt. Da sichergestellt werden muss, dass mit der eingegebenen Lösung keine Änderungen auf der Datenbank durchgeführt werden können, muss die Session als *Read only* markiert werden. Dies passiert im nachfolgenden Code Ausschnitt (4.4) in Zeile 2 und 3. Anschließend muss überprüft werden, ob es die angegebene Aufgabe überhaupt gibt, wenn dies der Fall ist, wird diese aus der Datenbank ausgelesen. Nun wird die Lösung des eingegebenen Statements mit der Lösung des korrekten Statements, welches zu der Aufgabe gespeichert wurde, verglichen. Stimmt die Lösung dieser Statements überein, wird markiert, dass die Aufgabe richtig gelöst wurde.

```
1  public ConfirmedObjectDTO checkResult(StatementDTO statement) throws
       SQLException {
2      Session session = entityManager.unwrap(Session.class);
3      session.setDefaultReadOnly(true);
4      Optional<Task> optionalTask = taskRepository
5      .findById(statement.getTaskID());
6      if (optionalTask.get() != null) {
7          try {
8              Task task = optionalTask.get();
9              setSearchPath(task.getTaskFamily().getdBSchema());
10             List<Map<String, Object>> statementResult = jdbcTemplate
11             .queryForList(statement.getQuery());
12             List<Map<String, Object>> result = jdbcTemplate
13             .queryForList(task.getResultQuery());
14             ConfirmedObjectDTO confirmedObjectDTO = new
15             ConfirmedObjectDTO();
16             if (statementResult.equals(result)) {
17                 String signedResult = createConfirmation(statement);
18                 confirmedObjectDTO.setGoalConfirmations(
19                 getConfirmations(signedResult, task));
20                 confirmedObjectDTO.setCorrectAnswer(true);
21             } else {
22                 confirmedObjectDTO.setCorrectAnswer(false);
23             }
24             session.setDefaultReadOnly(false);
25             setSearchPath("public");
26             return confirmedObjectDTO;
27         } catch (Exception e) {
28             setSearchPath("public");
29             e.getCause().getMessage());
30             throw new SQLException(
31             "Eingegebenes Skript ist nicht valide: " + e.getCause()
32             .getMessage());
33         }
34     }
35     return null;
36 }
```

Listing 4.4: Überprüfung eines Queries

Security

Sobald eine Aufgabe korrekt gelöst wurde, erstellt das Task-Service für jede erreichte Aufgabenklasse eine Bestätigung und signiert diese. Hierfür ist der Security Endpunkt zuständig. Um sicherzustellen, dass die Bestätigung tatsächlich vom Task-Service erstellt wurde und nicht gefälscht ist, wird der Lernfortschrittsbestätigung eine Signatur hinzugefügt. Die Signatur wird durch das Verschlüsseln mit einem *Private Key* des eingegebene Statements erstellt. Der nachfolgende Code-Ausschnitt zeigt den Verschlüsselungsprozess.

```
1 private String createConfirmation(StatementDTO statementDTO) throws
    DecryptException {
2       PrivateKey privateKey = generateKeys.getPrivateKey();
3       String encryptedResult = generateKeys.encryptText(statementDTO.
4       getQuery(), privateKey);
5       return encryptedResult;
6    }
```

Listing 4.5: Verschlüsselung der Lösung

Da der *Private Key* nicht nach außen weitergegeben werden darf, enthält nachfolgende Tabelle 4.9 nur einen Endpunkt mit dem dazugehörigen *Public Key*. Dieser ist notwendig, um die Signatur zu entschlüsseln und somit die Integrität der Lernfortschrittsbestätigung zu überprüfen.

Endpunkt	**Aktion**
/v1/security/key	Um sicherzustellen, dass die Lernfortschrittsbestätigung nicht gefälscht wurde, wird diese vom Task-Service mit einer Signatur ausgestattet. Damit das Maintenance-Service überprüfen kann, ob die Signatur gültig ist, benötigt es den zur asymmetrischen Verschlüsselung dazugehörigen *Public Key*. Dieser kann mit diesem Endpunkt abgerufen werden.

Tabelle 4.9: Schnittstelle Security

4.3 Task-Assignment-Generator-Service

Anders als die beiden anderen Services wird das Task-Assignment-Generator-Service (TAG-Service) nicht zentral, sondern lokal bei den Studierenden betrieben. Dafür wird der Service bei den Studierenden als Java-Programm ausgeführt. Damit soll sichergestellt werden, dass Daten von Studierende nicht von außen erreichbar sind. Da jedoch die Daten, welche in der In-Memory Datenbank verwaltet werden, mit dem Stoppen der lokalen Anwendung nicht mehr verfügbar sind, werden die Daten verschlüsselt und zentral in der Datenbank des Maintenance-Services gespeichert. Damit das Task-Assignment-Generator-Service funktioniert, muss beim Starten des Java-Programms ein individueller *Secret Key* mitgegeben werden. Mit diesem wird das Backup ent- bzw. verschlüsselt, um sicherzustellen, dass nur die Studierenden selbst Zugriff auf ihre Daten haben.

Das Task-Assignment-Generator-Service verwaltet die individuellen User-Präferenzen, welche für das Generieren der Aufgabenliste notwendig sind. Diese Aufgabenliste, welche mithilfe des DLV-Programms generiert wird, wird anschließend vom Frontend, dem Task-Assignment-Generator-Demonstrator, den Studierenden präsentiert. Zudem kann das Task-Assignment-Generator-Service den Studierenden übermitteln, welche Aufgabenklassen sie bereits in der Gesamthierarchie erreicht haben. Da versucht wird, bei der Generierung der Aufgabenliste auf die individuellen Bedürfnisse der Studierenden einzugehen, speichert das Task-Assignment-Generator-Service zudem, wie viele Versuche notwendig waren, um eine Aufgabe zu lösen. Daher werden alle Statements, welche die Studierenden absetzen, zuerst an das Task-Assignment-Generator-Service gesendet, welches diese anschließend zur Überprüfung an das Task-Service weiterleitet.

Schnittstellen

Für folgende Funktionsbereiche besitzt das Task-Service eigene Endpunkte:

- Aufgabenklasse
- User-Lösungsabfrage
- Aufgabenliste
- Verbindungsprüfung

In den nachfolgenden Abschnitten werden die dazugehörigen Endpunkte näher beschrieben.

Aufgabenklasse

Wie bereits in Abschnitt 3.2 erwähnt, können Studierende mit verschiedene Aufgaben Aufgabenklassen erreichen. Dabei entsprechen die Aufgabenklassen den Sprachkonstrukten von SQL. Dieser Endpunkt kümmert sich um die Abfrage aller bereits erlernten Aufgabenklassen. Dies soll den Studierenden einen Einblick in ihren aktuellen Wissensstand geben. Die nachfolgende Tabelle 4.10 beschreibt den dazugehörigen Endpunkt.

Endpunkt	**Aktion**
/v1/confirmedGoals	Übermittelt alle gespeicherten und bestätigten Aufgabenklassen des Users. Hiermit kann dem User angezeigt werden, welche Aufgabenklassen bereits absolviert wurden und welche noch ausstehend sind.

Tabelle 4.10: Schnittstelle Aufgabenklassen

User-Lösungsabfrage

Die Endpunkte für die User-Lösungsabfrage (Tabelle 4.11) dienen nur als zwischen Endpunkte von Task-Service und Task-Assignment-Generator-Service. Die Endpunkte *check* und *execute* leiten die Anfrage zur Prüfung bzw. Ausführung einer Aufgabenlösung an das Task-Service weiter. Es wird jedoch, zur Unterstützung des Users und für die Anpassung der Aufgabenliste, beim Ausführen ein Log-Eintrag erstellt. Diese Liste mit Lösungsversuchen ist nur für die Studierenden einsehbar.

Endpunkt	**Aktion**
/v1/statement/check	Überprüft ob die eingegebene Lösung korrekt ist.
/v1/statement/execute	Führt die Lösung aus und liefert das Ergebnisse zurück.
/v1/statement/submit	Übermittelt die gespeicherten Lernfortschrittskontrollen für eine Aufgabensammlung an das Maintenance-Service. Hiermit kann ein User eine Aufgabe abgeben.

Tabelle 4.11: Schnittstelle User-Lösungsabfrage

Der Endpunkt *submit* führt den Code-Ausschnitt 4.6 aus. Da bei den Bestätigungen der Aufgabenklassen nicht mitgespeichert wird, für welchen User die Bestätigung ausgestellt

wurde, muss diese Information das Task-Assignment-Generator-Service selber noch anhängen. Anschließend sendet es die Bestätigung an das Maintenance-Service. Sollte dieses Service jedoch feststellen, dass die Signatur nicht gültig ist, so erhält das Task-Assignment-Generator-Service eine Rückmeldung und leitet dieses an das Frontend weiter.

```
public boolean submitConfirmation() throws SignatureInvalidException,
    SettingsNotFoundException {
    ExerciseSheetDTO exerciseSheetDTO = exerciseSheetClient
    .getActiveExerciseSheet();
    SubmissionDTO submissionDTO = new SubmissionDTO();
    submissionDTO.setSheetID(exerciseSheetDTO.getId());
    String userName = settingsService.getSettings().getUserName();
    submissionDTO.setUsername(userName);
    boolean complete = true;
    for (LearningGoalDTO learningGoalDTO : exerciseSheetDTO.getGoals()) {
        GoalConfirmationDTO goalConfirmation = learningGoalService
        .findByName(learningGoalDTO.getName());
        if (goalConfirmation != null) {
            submissionDTO.getGoalConfirmations().add(goalConfirmation);
        } else {
            complete = false;
        }
    }
    if (!submissionDTO.getGoalConfirmations().isEmpty()) {
        Boolean validSignature = learningConfirmationClient
        .submit(submissionDTO);
        if (validSignature == null || !validSignature) {
            throw new SignatureInvalidException
            ("Signatur für die Bestätigung ist nicht valide");
        }
    }
    return complete;
}
```

Listing 4.6: Bestätigung absenden

Aufgabenliste

Um das Lernen an die individuellen Bedürfnisse der Studierenden anpassen zu können gibt es den Funktionsbereich *Aufgabenliste*. Hier können die Studierenden verschieden Präferenzen in Bezug auf die Aufgabenliste angeben. Diese betreffen etwa den Schwierigkeitsgrad einer Aufgabe, die zugewiesene Aufgabenfamilie oder auch das Anpassen des Schwierigkeitsgrades nach jeder absolvierten Aufgabe. Zudem wird auch direkt die Generierung der Aufgabenliste angestoßen. Dabei werden die angegebenen User-Präferenzen an den Task-Assignment-Generator-Service weitergegeben und anschließend ein individualisierte Aufgabenliste generiert. Im Kapitel 6 wird näher auf die verschiedenen Aspekte der Erstellung der Aufgabenliste eingegangen.

Eine weitere Funktion des Funktionsbereichs für die Aufgabenlisten ist die Abfrage aller bereits generierten Aufgabenlisten, sowie der dazugehörigen Lösungsversuche. Als Unterstützung für die Studierenden wird bei jeder Aufgabe, welche ausgeführt oder überprüft wird, ein Log-Eintrag erzeugt. Zu einem Lösungsversuch gehört die eingegebene Lösung sowie die Aufgabenstellung. Dies soll einen Einblick in den Lernfortschritt geben. Da nur das Task-Assignment-Generator-Service weiß, welche Aufgaben einer Aufgabenliste bereits absolviert wurden, gibt es auch einen Endpunkt, welcher die nächste offene Aufgabe in der Aufgabenliste zurückliefert. Nachfolgende Tabelle 4.12 zeigt diesen Funktionsbereich.

Endpunkt	**Aktion**
/v1/learnpath/{id}	Liefert alle verfügbaren Aufgabenlisten. Wenn eine ID angegeben wird, so wird nur die angeforderte Aufgabenliste übermittelt.
/v1/learnpath/create	Die Aufgabenliste wird mithilfe des DLV-Programms auf Basis der User-Präferenzen generiert. Mit diesem Endpunkt wird die Generierung angestoßen.
/v1/learnpath/next/{id}	Liefert die nächste Aufgabe in der übermittelten Aufgabenliste.
/v1/learnpath/preferences	Hiermit können die gespeicherten User-Präferenzen abgerufen werden. Mit *POST* können die User-Präferenzen gesichert werden.
/v1/learnpath/log	Alle gespeicherten Logs zu den absolvierten Aufgaben der Aufgabenlisten werden übermittelt.

Tabelle 4.12: Schnitstelle Aufgabenliste

Der Endpunkt *create* stoßt die Generierung der Aufgabenliste an. Dazu wird dem Task-Assignment-Generator die ID der Aufgabensammlung übermittelt. Anschließend werden alle notwendigen Informationen (User-Präferenzen, Vorwissen, Informationen zu den Aufgaben) vom Task-Assignment-Generator-Service dem DLV-Programm übermittelt. Manche Informationen sind nur im Task-Service gespeichert, daher muss das Task-Assignment-Generator-Service diese Informationen anfordern. Sobald das notwendige Wissen vorhanden ist, kann eine individuelle Aufgabenliste generiert werden. Dieser wird für den User in der lokalen In-Memory-Datenbank gespeichert. Um einen Datenverlust zu vermeiden, wird direkt nach der Generierung ein verschlüsseltes Backup an das Maintenance-Service gesendet.

```
1 int sheetID = exerciseSheetClient.getActiveExerciseSheet().getId();
2 ModelBufferedHandler result = aspEngine
3 .run("t, reached_g, not_g_reached", sheetID);
4 LearnPathResultDTO learnPathResultDTO = userTaskInfoService
5 .saveLearnPath(result, sheetID);
6 dataBaseService.createBackup();
```

Listing 4.7: Generierung einer Aufgabenliste

Verbindungsprüfung

Damit das Frontend weiß, dass das bei den Studierenden individuell gestartete Task-Assignment-Generator-Service richtig konfiguriert wurde und bereit für Anfragen ist, gibt es die nachfolgenden Schnittstellen (Tabelle 4.13). Zudem wird mit diesen Endpunkten einerseits sichergestellt, dass die Daten nicht an Unbefugte weitergegeben werden und andererseits die Daten der Studierenden nicht verloren gehen.

Endpunkt	**Aktion**
/v1/connection/check	Sobald das Frontend über diesen Endpunkt eine Rückmeldung bekommt, weiß es, dass das Task-Assignment-Generator-Service bereit für Anfragen ist.
/v1/connection/clear	Sobald ein User sich abgemeldet hat, wird mit diesem Endpunkt von den gespeicherten Daten ein verschlüsseltes Backup erstellt und im Maintenance-Service gespeichert. Zudem werden die lokal gespeicherten Daten des Users gelöscht.
/v1/connection/user	Da beim Starten des lokalen Services, das Task-Assignment-Generator-Service noch nicht weiß, welcher User angemeldet ist, wird der User, welcher sich im Frontend angemeldet hat, an das Task-Assignment-Generator-Service gesendet. Anschließend wird überprüft, ob es bereits ein Backup zu diesem User gibt. Wenn dies so ist, dann wird dieses aus dem Maintenance-Service abgerufen und mit dem individuellen Schlüssel entschlüsselt. Sollte der übermittelte Schlüssel jedoch fehlerhaft sein, wird der angemeldete User wieder abgemeldet.

Tabelle 4.13: Schnittstelle Verbindungsprüfung

Wie bereits in der oben stehenden Tabelle beschrieben, werden mithilfe des Endpunkts *clear* die Benutzerdaten gelöscht, nachdem ein Backup erstellt wurde. Der nachfolgende Code-Ausschnitt (4.8) zeigt die Erstellung des Backups. Dabei werden als Erstes die gespeicherten Settings geladen. Diese sind notwendig um herauszufinden, welcher User aktuell angemeldet ist. Anschließend wird eine Verbindung zur Datenbank aufgebaut und ein Skript zur Erstellung des Backups ausgeführt. Nun wird ein Objekt erstellt, welches den aktuellen User und das Backup als verschlüsselten String enthält. Sobald diese Schritte abgeschossen sind, kann dem Maintenance-Service mithilfe der *User-Schnittstelle* das Backup übermittelt werden.

```
public void createBackup() throws SQLException, IOException,
    DecryptException, SettingsNotFoundException {
    Settings settings = settingsService.getSettings();
    if (settings != null) {
        conn = DriverManager.getConnection(url, username, password);
        Statement stmt = conn.createStatement();
        stmt.executeQuery(String.format("SCRIPT TO  '%s'", filePath));
        File file = new File(filePath);
        byte[] backup = new byte[(int) file.length()];
        FileInputStream fileInputStream = new FileInputStream(file);
        fileInputStream.read(backup);
        BackUpDTO backUpDTO = new BackUpDTO();
        backUpDTO.setUserName(settings.getUserName());
        backUpDTO.setBackup(encryptionService.encrypt(new String(backup))
    );
        userClient.createBackUp(backUpDTO);
        conn.close();
    } else {
        logger.info("createBackup, user not found! cannot load backup");
    }
}
```

Listing 4.8: Erstellung eines Backups

Nachdem der Endpunkt *user* den aktuell angemeldeten User erhalten hat, wird versucht das gespeicherte Backup wiederherzustellen. Sollte dies nicht möglich sein, da ein falscher Schlüssel vom User eingegeben wurde, wird eine Fehlermeldung ausgegeben. Der nachfolgende Code-Ausschnitt (4.9) zeigt diesen Ablauf:

```
public void restoreBackUp(String backup) throws SQLException,
    IOException, DecryptException {
    FileOutputStream fileOutputStream = new FileOutputStream(filePath);
    try {
        fileOutputStream.write(encryptionService.decrypt(backup)
        .getBytes());
        this.resetDataBase();
        conn = DriverManager.getConnection(url, username, password);
        Statement stmt = conn.createStatement();
        stmt.executeUpdate(String.format("RUNSCRIPT FROM  '%s'", filePath
    ));
        conn.close();
    } catch (DecryptException e) {
        throw new DecryptException("Wrong secret key!");
    }
}
```

Listing 4.9: Backup wiederherstellen

4.4 Diskussion Zielerreichung

Im nachfolgenden Abschnitt werden die in Abschnitt 1.4 beschrieben Anforderungen auf ihre Zielerreichung in Bezug auf das Backend diskutiert. Dabei werden die Anforderungen in *Individualisierung Aufgabenliste, Wissenenstand der Studierenden* und *Wahrung der Privatsphäre* unterteilt.

A1 Individualisierung Aufgabenliste

Die Aufgabenliste soll von den Studierenden an ihre individuellen Bedürfnisse angepasst werden können. Dabei sollen die Aufgabenfamilien, der Schwierigkeitsgrad und die Aufgabenliste eingestellt werden können.

A1.1 Einstellungsmöglichkeit Aufgabenfamilie
Für diese Einstellungen gibt es den Funktionsbereich *Aufgabenliste* im Abschnitt 4.3 Task-Assignment-Generator-Service. Hier können die Studierenden beim Endpunkt *preferences* ihre individuellen Einstellungen festlegen. Somit kann angegeben werden, ob ein Wechsel zwischen Aufgabenfamilien stattfinden darf oder nicht.

A1.2 Einstellungsmöglichkeit Schwierigkeitsgrad
Ebenso wie bei *Einstellungsmöglichkeit Aufgabenfamilie* ermöglicht der Endpunkt *preferences* die Einstellung des Schwierigkeitsgrads. Studierenden wird hiermit die Möglichkeit gegeben, ihren gewünschten Schwierigkeitsgrad einzustellen.

A1.3 Anpassung Aufgabenliste
Für die *Anpassung der Aufgabenliste* gibt es den Endpunkt *create* im Abschnitt 4.3 Task-Assignment-Generator-Service. Studierende haben dadurch die Möglichkeit die Aufgabenliste neu generieren zu lassen.

A2 Wissensstand der Studierenden

Mit dieser Anforderung sollen Studierende Einblick in ihren aktuellen bzw. vergangenen Wissensstand erhalten. Zudem soll bereits vergangenes Wissen bei der Aufgabenlistengenerierung berücksichtigt werden.

A2.1 Wissensstand Aufgabenklassen
Die Anforderung *Wissensstand Aufgabenklassen* wird mit dem Funktionsbereich *Aufgabenklassen* im Abschnitt 4.3 umgesetzt. Damit können alle bereits erlernten Aufgabenklassen abgefragt werden.

A2.2 Wissensstand und Aufgabenlistengenerierung
Bei der Generierung der Aufgabenliste soll bereits erlerntes Wissen berücksichtigt werden. Für die Umsetzung dieser Anforderung ist ebenso der Funktionsbereich *Aufgabenklasse* relevant. Hier wird das bereits erlernte Wissen gespeichert. Bei der Generierung einer neuen Aufgabenklasse wird dieses Wissen an das DLV-Programm weitergegeben und berücksichtigt.

A2.3 Vergangene Abfragen
Studierende können sich ihre vergangenen Abfragen mithilfe des Funktionsbereichs *Aufgabenliste* und dem Endpunkt *log* anzeigen lassen. Dieser Endpunkt übermittelt alle zu den Abfragen gespeicherten Lösungsversuche. Somit können sich die Studierenden ihren Lernfortschritt ansehen.

A3 Wahrung der Privatsphäre

Damit Lernplattformen auf die individuellen Bedürfnisse der Studierenden eingehen können, werden eine Vielzahl an Daten gesammelt. Es soll sichergestellt werden, dass diese Daten nur für die Studierenden verfügbar sind.

A3.1 Privates Wissen über die Studierenden
Der private Wissensstand der Studierenden ist nur im *Task-Assignment-Generator-Service* verfügbar. Dieses Service ist jedoch nur lokal verfügbar und nach dem Stoppen der Anwendung würden die Daten der Studierenden verloren gehen. Daher gibt es beim Funktionsbereich *Verbindungsprüfung* zum einen den Endpunkt *user*, welcher nach dem erfolgreichen Einloggen eines Users ein bereits bestehendes Backup in das lokale Service lädt. Zum anderen gibt es den Endpunkt *clear*, welcher nach dem ausloggen eines Users die Daten mit dem individuellen Schlüssel verschlüsselt und ein Backup an das Maintenance-Service sendet. Die genau Beschreibung dieses Services wird bei der Schnittstelle *Verbindungsprüfung* beschrieben.

A3.2 Lernfortschrittsbestätigung
Für die Überprüfung der Lernfortschrittsbestätigung gibt es auch im Task-Assignment-Generator-Service den Funktionsbereich *User-Lösungsabfrage*. Hier gibt es den Endpunkt *submit*, mit welchem Studierende die Möglichkeit haben ihre erfolgreich erlernten Aufgabenklassen zu einer Aufgabensammlung als Bestätigung abzugeben. Im Code-Ausschnitt 4.6 sieht man den Prozess, welcher beim Abgeben einer Lernfortschrittsbestätigung durchgeführt wird.

Zusammenfassung

In diesem Kapitel wurden die einzelnen Backend-Services (Maintenance-, Task- und Task-Assignment-Generator-Service) beschrieben. Es wurde auf die einzelnen Funktionsbereiche sowie Endpunkte der Services eingegangen und teilweise für ein besseres Verständnis auf einen konkreten Code-Ausschnitt eingegangen. Zudem wurden die Anforderungen aus Abschnitt 1.4 in Bezug auf das Backend diskutiert.

Kapitel 5

Frontend

In diesem Kapitel werden die verschiedenen Visualisierungen der dazugehörigen Backend-Funktionen, welche im Kapitel 4 beschrieben wurden, vorgestellt. Dabei wird zwischen den Funktionalitäten, welche Studierende durchführen können und jenen, welche nur User mit Administratorrechten durchführen können, unterschieden. Zudem zeigen die verschiedenen Farben (*Grün*, *Blau*) des Frontends an, welche Rolle die angemeldete Person hat. *Grün* steht für Studierende und *Blau* für AdministratorInnen.

5.1 Studierenden-Sicht

Im nachfolgendem Abschnitt werden die verschiedenen Screens der Studierenden-Sicht vorgestellt. Um Zugriff auf die Anwendung zu erhalten, benötigen die Studierenden einen Account. Diesen können sie sich mit dem Registrierungsscreen (Abbildung 5.1) erstellen. Sobald Studierende einen Account besitzen, können sie sich mit Username und Passwort einloggen (Abbildung 5.2).

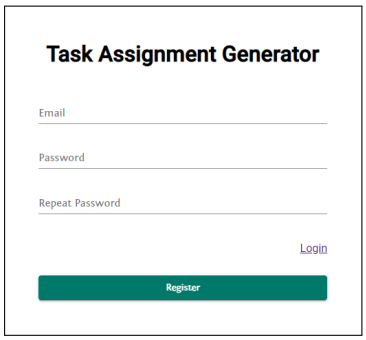


Abbildung 5.1: Registrierung

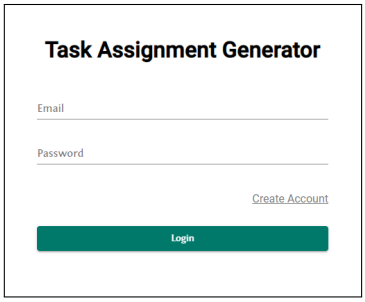


Abbildung 5.2: Login

Nach dem Login gelangt man auf das Dashboard, siehe Abbildung 5.3. Das Dashboard zeigt den Studierenden (sofern bereits vorhanden) ihre aktuelle Aufgabenliste an. Zudem ist sofort erkennbar, wie viele Aufgaben bereits erledigt wurden und wie viele noch offen sind.

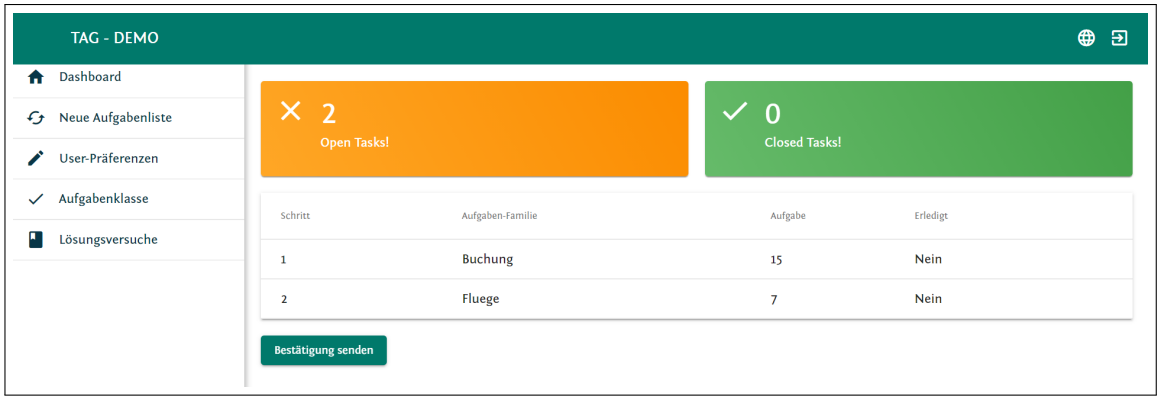


Abbildung 5.3: Dashboard

Sollten die Studierenden noch keine Aufgabenliste haben, so können sie sich unter *Neue Aufgabenliste* eine generieren lassen. Dabei sehen sie, wie in Abbildung 5.4 dargestellt, für welche Aufgabenklassen die Aufgabenliste generiert wird. Sollten Studierende Änderungen an ihren Präferenzen vorgenommen haben, können sie sich zudem mit dieser Funktion eine neue Aufgabenliste mit angepassten Präferenzen generieren lassen.

Abbildung 5.4: Aufgabenliste

Studierende können sich ihren aktuellen Wissensstand anzeigen lassen (Abbildung 5.5). Dabei wird in *grün* gezeigt, welche Aufgabenklassen (Sprachkonstrukte) bereits geübt wurden und in *orange*, welche Aufgabenklassen noch nicht geübt wurden. Wenn ein Child-Klasse geübt wurde, so wird auch die *Parent-Klasse* als geübt markiert. Nachfolgende Grafik zeigt, dass die *Child-Klasse count* erfolgreich geübt wurde und daher auch die allgemeine Aufgabenklasse *aggregation* geübt wurde.

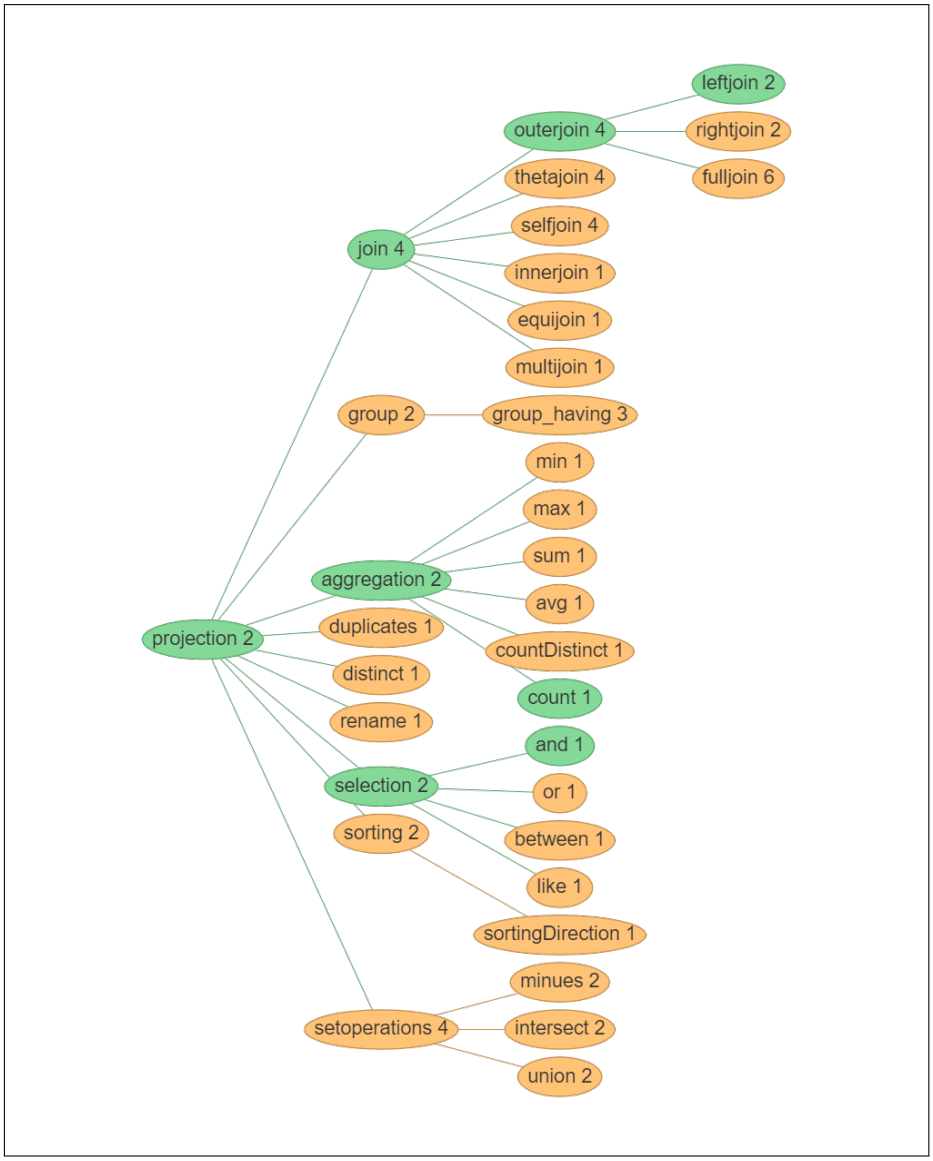


Abbildung 5.5: Lernfortschrittsübersicht

Sobald für Studierende eine individuelle Aufgabenliste generiert wurde, können sie beginnen, die verschiedenen Aufgaben zu lösen. Abbildung 5.6 zeigt diese Ansicht für Studierende. Dabei wird unter *Aufgabe* die zu lösende Aufgabe näher beschrieben. *Schema Beschreibung* zeigt wie die verschiedenen Datenbanktabellen zusammen hängen. Dabei zeigt ein unterstrichenes Wort den *Primary Key* und ein kursives Wort einen *Fremdschlüssel*. In nachfolgender Grafik hat die Tabelle *inhaber* einen zusammengesetzten *Primary Key* aus name und gebdat. Die Tabelle *konto* hat als Fremdschlüssel *inhname*. Um eine Aufgabe lösen zu können, hab die Studierenden die Möglichkeit unter *Fügen Sie hier ihren Query ein* ihre Lösung einzugeben.

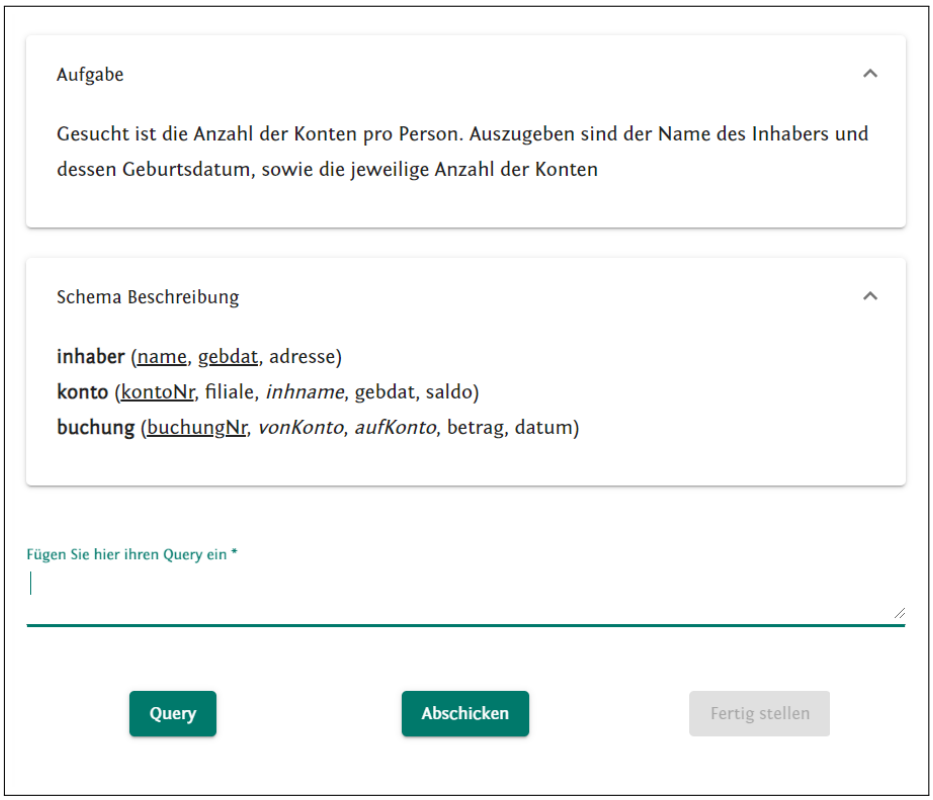


Abbildung 5.6: Lösung Statement

Abbildung 5.7 zeigt das Ergebnis des eingegebenen Statements. Sobald man auf *Query* klickt, wird das eingegebene Statement ausgeführt und unterhalb wird das Ergebnis angezeigt. Durch den Button *Abschicken* können die Studierenden ihr Ergebnis abschicken und überprüfen lassen. Die Studierenden erhalten anschließend eine Rückmeldung, ob ihre Lösung korrekt war. Sollte dies der Fall sein, wird der Button *Weiter/ Fertig stellen* aktiv geschaltet und es kann entweder mit der nächsten Aufgabe fortgesetzt werden oder, sollte es bereits die letzte Aufgabe gewesen sein, kann die Aufgabenliste abgeschlossen werden.

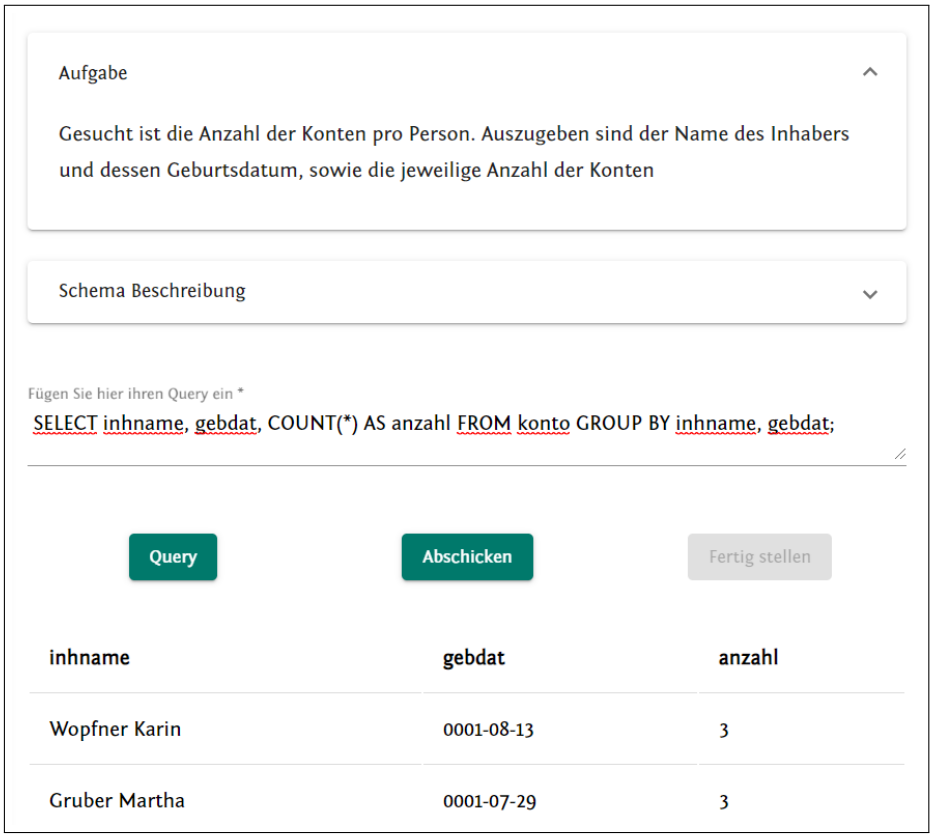


Abbildung 5.7: Statement Ausführen

Studierende können sich eine Übersicht alle ihrer eingegeben und abgeschickten Statements ansehen. So ist es etwa möglich, dass sich Studierende ansehen, wie viele Versuche sie bei einer Aufgabe benötigt haben und wie ihre Lernentwicklung war. Abbildung 5.8 zeigt diese Ansicht:

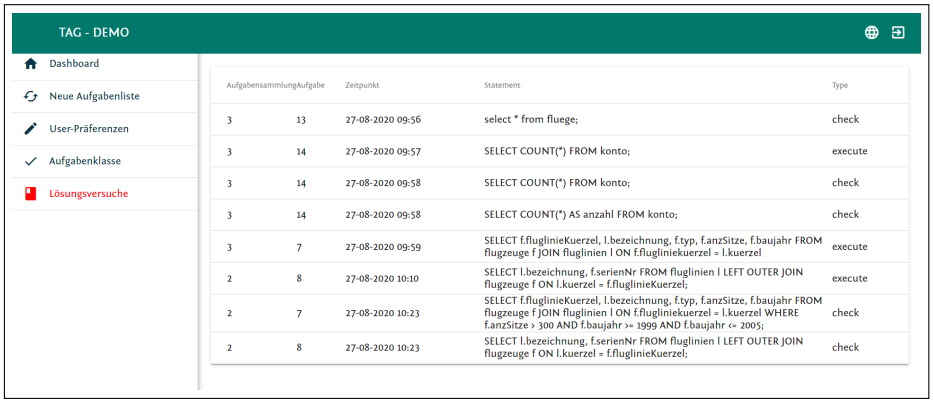


Abbildung 5.8: Log aller eingegeben Lösungsversuche

Um die generierte Aufgabenliste an die individuellen Bedürfnisse anpassen zu können, haben Studierende die Möglichkeit, verschiedene Präferenzen anzugeben. Die erste Einstellungsmöglichkeit beschäftigt sich mit den Aufgabenfamilien. Hier können Studierende entscheiden, wie wichtig es ihnen ist, dass es keinen Wechsel zwischen den verschiedenen Aufgabenfamilien bei den Aufgaben in einer Aufgabenliste gibt.

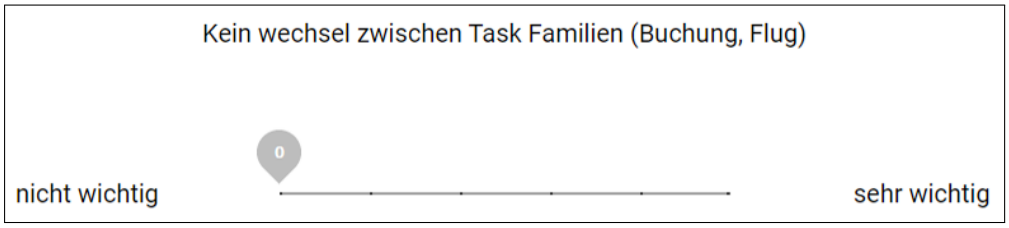


Abbildung 5.9: Einstellungsmöglichkeiten Aufgabenfamilien

In Abbildung 5.9 wurde eingestellt, dass es nicht wichtig ist, dass kein Wechsel zwischen Aufgabenfamilien stattfinden soll. In Abbildung 5.10 sieht man das Ergebnis dieser Einstellung. Es müssen daher sowohl Aufgaben mit der Aufgabenfamilie *Buchung* als auch Aufgaben mit der Aufgabenfamilie *Fluege* absolviert werden.

Schritt	Aufgaben-Familie	Aufgabe	Erledigt
1	Fluege	3	Nein
2	Fluege	7	Nein
3	Buchung	24	Nein

Abbildung 5.10: Wechseln zwischen Aufgabenfamilien

Im Gegensatz dazu wurde für Abbildung 5.11 angegeben, dass kein Wechsel zwischen Aufgabenfamilien durchgeführt werden soll. Daher gehören alle zu lösenden Aufgaben zur Aufgabenfamilie *Fluege*.

Schritt	Aufgaben-Familie	Aufgabe	Erledigt
1	Fluege	3	Nein
2	Fluege	7	Nein
3	Fluege	8	Nein

Abbildung 5.11: Kein Wechsel zwischen Aufgabenfamilien

Die nächste Einstellungsmöglichkeit für Studierende ist der Schwierigkeitsgrad. Studierende können auf einer Skala von 1 bis 15 angeben, welchen relativen Schwierigkeitsgrad (relativ in Bezug zu bereits gelösten Aufgaben) sie pro Aufgabe bevorzugen. Dabei steht 1 für sehr einfach und 15 für sehr schwierig. Abbildung 5.12 zeigt diese Einstellungsmöglichkeiten.

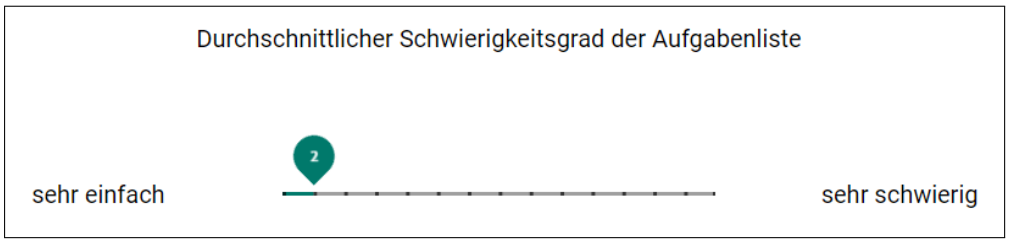


Abbildung 5.12: Schwierigkeitsgrad leicht

In Abbildung (5.13) sieht man eine Aufgabenliste die mit Einstellungen aus Abbildung 5.12 generiert wurde. Da eingestellt wurde, dass der durchschnittliche Schwierigkeitsgrad 2 sein soll, müssen insgesamt 5 Aufgaben mit einem durchschnittlichen Schwierigkeitsgrad von 2 erledigt werden.

Schritt	Aufgaben-Familie	Aufgabe	Erledigt
1	Fluege	13	Nein
2	Fluege	3	Nein
3	Fluege	7	Nein
4	Fluege	8	Nein
5	Buchung	24	Nein

Abbildung 5.13: Schwierigkeitsgrad leicht

Bei Abbildung 5.14 wurde ein durchschnittlicher Schwierigkeitsgrad von 5 eingestellt. Dadurch wurde die Anzahl der zu lösenden Aufgaben auf 3 reduziert, jedoch auch der Schwierigkeitsgrad der einzelnen Aufgaben erhöht.

Schritt	Aufgaben-Familie	Aufgabe	Erledigt
1	Fluege	3	Nein
2	Fluege	7	Nein
3	Buchung	24	Nein

Abbildung 5.14: Schwierigkeitsgrad mittel

In den Abbildungen 5.15 und 5.16 sieht man das Ergebnis der Aufgabenliste, wenn der durchschnittliche Schwierigkeitsgrad auf 10 bzw. 15 erhöht wird. Dadurch reduziert sich die Anzahl der Aufgaben auf zwei bzw. sogar auf eine, jedoch mit einem viel höheren Schwierigkeitsgrads.

Schritt	Aufgaben-Familie	Aufgabe	Erledigt
1	Fluege	3	Nein
2	Fluege	8	Nein

Abbildung 5.15: Schwierigkeitsgrad schwer

Schritt	Aufgaben-Familie	Aufgabe	Erledigt
1	Fluege	8	Nein

Abbildung 5.16: Schwierigkeitsgrad sehr schwer

Die Einstellungsmöglichkeit des Schwierigkeitsgrads zeigt den Grundgedanken des intelligenten tutoriellen Systems: Studierenden können durch die Wahl des relativen Schwierigkeitsgrads Steuern, wie schnell neue Aufgabenklassen in den zu lösenden Aufgaben hinzukommen. Dabei wird bei einem niedrigeren relativen Schwierigkeitsgrad die Aufgabenliste

länger, es müssen mehr Aufgaben, jedoch mit einem niedrigeren relativen Schwierigkeitsgrad absolviert werden, und mit einem höheren relativem Schwierigkeitsgrad wird die Aufgabenliste kürzer. Der Grundgedanke ist, dass alle Studierenden am Ende zum gleichen Ziel kommen, der Weg dort hin wird jedoch an die individuellen Bedürfnisse angepasst.

Eine weitere Einstellungsmöglichkeit für Studierende ist, den Schwierigkeitsgrad der Aufgaben nach jeder erfolgreich gelösten Aufgabe anpassen zu lassen. Dafür gibt es zwei Einstellungsmöglichkeiten, welche in Abbildung 5.17 dargestellt werden. Wenn der *Dialog für angepassten Schwierigkeitsgrad* aktiv ist, öffnet sich nach dem erfolgreichen Absolvieren einer Aufgabe ein Dialog, in welchen man entscheiden kann, ob der Schwierigkeitsgrad zu hoch, zu niedrig oder genau richtig war. Je nachdem wird der eingestellte Schwierigkeitsgrad um 1 erhöht oder verringert. Wenn der Dialog *Aufgabenliste nach Änderung neu generieren* aktiv ist, wird direkt nach einer Änderung des eingestellten Schwierigkeitsgrads eine neue Aufgabenliste generiert.

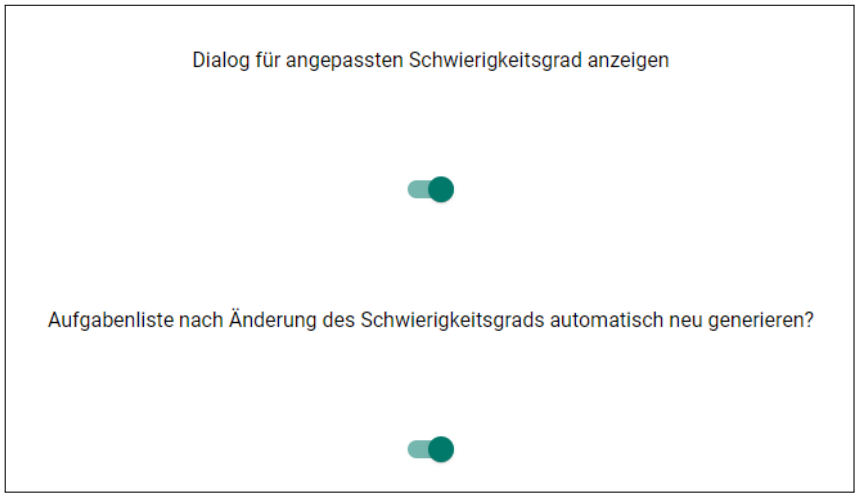


Abbildung 5.17: Schwierigkeitsgrad automatisch anpassen

Sollten für eine Aufgabensammlung bereits alle Aufgabenklassen erreicht worden sein und es wird dennoch eine neue Aufgabenliste generiert, so wird dies den Studierenden auf der Startseite angezeigt. Abbildung 5.18 zeigt diese Visualisierung.

Sobald ein User eine Aufgabe abgeschlossen hat, gibt es die Möglichkeit eine Lernfortschrittsbestätigung zu senden. Dazu erscheint der Button *Bestätigung senden* direkt am Dashboard. Für das Abgeben der Aufgabe ist es nicht notwendig, dass bereits alle Aufgaben gelöst wurde, sondern es können auch Teillösungen abgegeben werden. Jedoch wird dann den Studierenden eine Hinweismeldung angezeigt (Abbildung 5.19). Dadurch soll sichergestellt werden, dass die Leistung der Studierenden beurteilt werden kann, auch wenn eine Aufgabe nicht gelöst werden konnte.

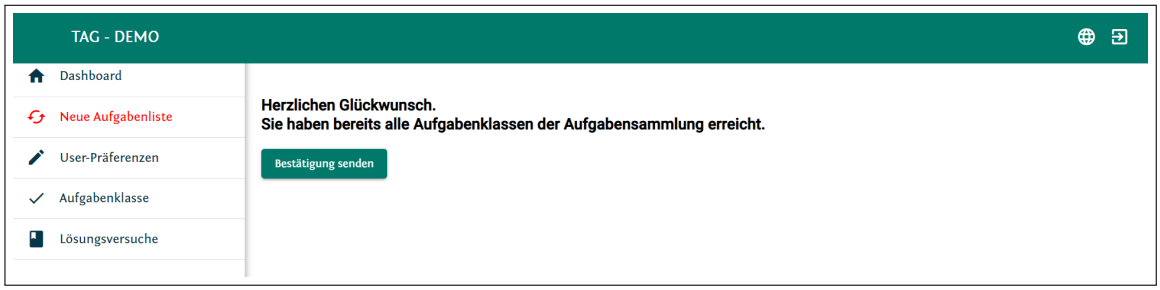


Abbildung 5.18: Alle Aufgaben erledigt

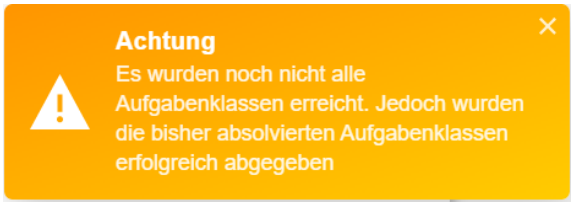


Abbildung 5.19: Hinweismeldung Lernfortschrittsbestätigung

Für eine einfachere Verwaltung können die ÜbungsleiterInnen mehrere Aufgabensammlungen anlegen. Die Studierenden können sich jeweils für die aktuelle *aktive* Aufgabensammlung eine Aufgabenliste generieren lassen. Sollte jedoch eine neue Aufgabensammlung als *aktiv* geschaltet werden, so bekommen die Studierenden bereits beim Einloggen eine Information. Anschließend können sie sich für die neue Aufgabensammlung eine neue Aufgabenliste generieren lassen.

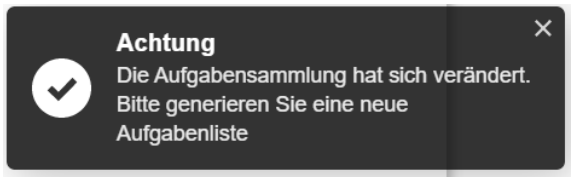


Abbildung 5.20: Hinweismeldung neue Aufgabensammlung

5.2 Administrator-Sicht

In diesem Abschnitt werden jene Funktionalitäten beschrieben, welche nur für User mit speziellen Rechten zugänglich sind. Damit ein User diese Rechte erhält, muss er von einem anderen Administrator berechtigt werden. Dies geschieht auf der Seite *User-Verwaltung*. Dafür gibt es zwei Ansichten, die erste (Abbildung 5.21) zeigt alle insgesamt angemeldeten User. Die zweite Abbildung 5.22 zeigt die Detailansicht der User-Verwaltung. Hier kann ein User gelöscht oder die Rolle und die damit verbundenen Rechte geändert werden.

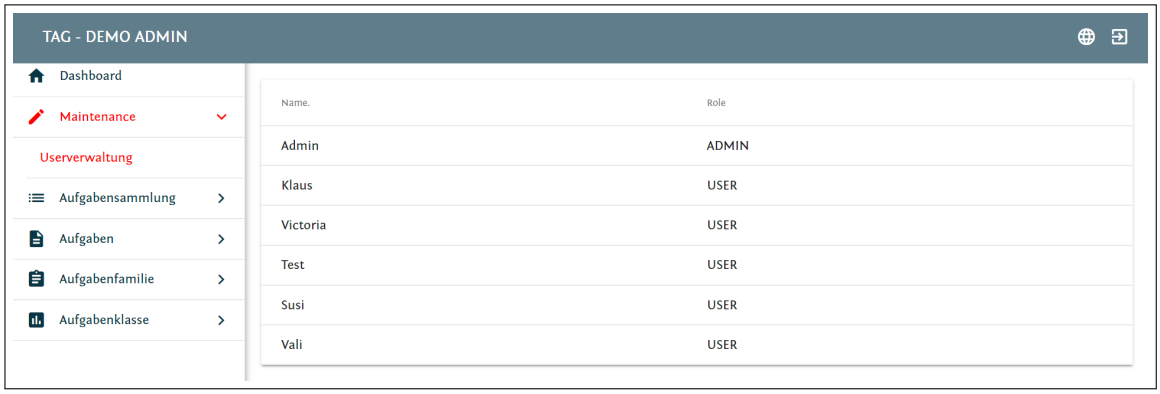


Abbildung 5.21: User-Verwaltung

Abbildung 5.22: User-Verwaltung - Details

AdministratorInnen können die Aufgabenklassen, welche von den Studierenden geübt werden, verwalten. Dabei können sie einstellen, wie die Aufgabenklassen zusammen hängen und welchen Schwierigkeitsgrad sie haben. Abbildung 5.23 zeigt einen Ausschnitt der Übersicht aller Aufgabenklassen. Jene Aufgabenklassen, welche in dieser Übersicht abgebildet sind, werden den Studierenden (Abbildung 5.5 in *Studierenden-Sicht*) in der Generalisierungshierarchie angezeigt.

No.	Name	Parent	Schwierigkeit
1	projection		2
3	join	projection	4
4	group	projection	2
5	aggregation	projection	2
6	duplicates	projection	1
7	distinct	projection	1
8	rename	projection	1
9	selection	projection	2
10	sorting	projection	2
11	setoperations	projection	4
12	outerjoin	join	4
13	thetajoin	join	4
14	selfjoin	join	4

Abbildung 5.23: Übersicht Aufgabenklassen

In einer Aufgabensammlung können mehrere Aufgabenklassen angegeben werden, welche anschließend von den Studierenden geübt werden sollen. Um die Verwaltung zu vereinfachen, können mehrere Aufgabensammlungen angelegt werden. Die AdministratorInnen können anschließend entscheiden, wann welche Aufgabensammlung von den Studierenden absolviert werden muss. Dazu wird sie als *aktiv* markiert. Anschließend können sich die Studierenden auf Basis der Aufgabensammlung ihre individuelle Aufgabenliste generieren lassen. In Abbildung 5.24 sieht man eine Übersicht aller Aufgabensammlungen. Aktuell ist Sammlung 3 aktiv geschaltet.

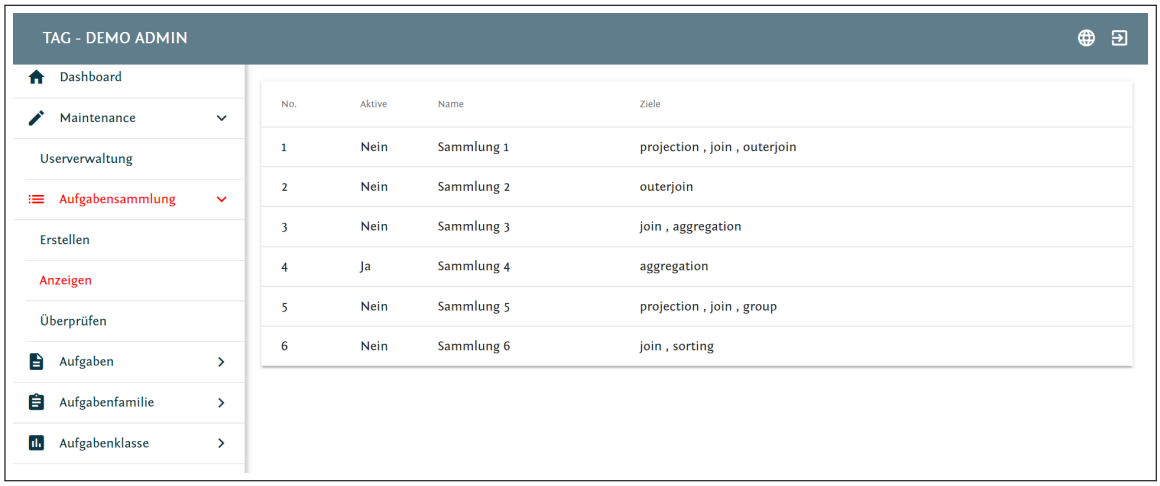


Abbildung 5.24: Übersicht Aufgabensammlungen

Zudem können unter *Aufgabensammlung* - *Überprüfen* alle abgegebenen Lernfortschrittsbestätigungen der Studierenden angesehen werden. Dabei wird auf der Übersicht (Abbildung 5.25) angezeigt, welche Studierende welche Aufgabenklassen erfolgreich absolviert haben. Aufgabenklassen, welche *rot* markiert sind, wurden dabei noch nicht von der Person erledigt. Im konkreten Fall hat *Susi* - *Join* noch nicht abgeschlossen. Alle anderen Aufgabenklassen wurden erfolgreich absolviert.

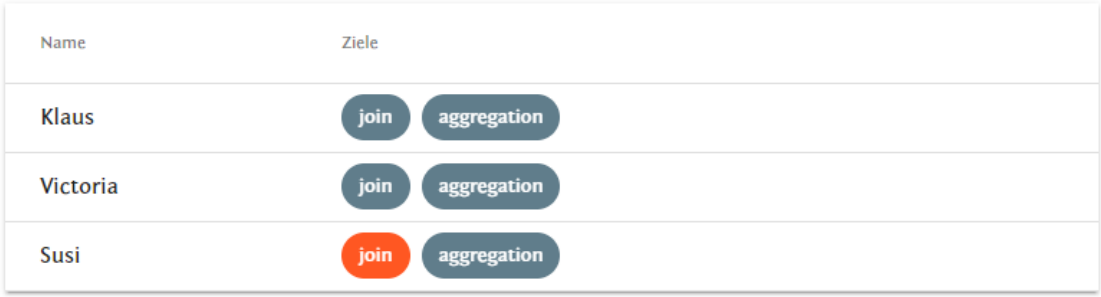
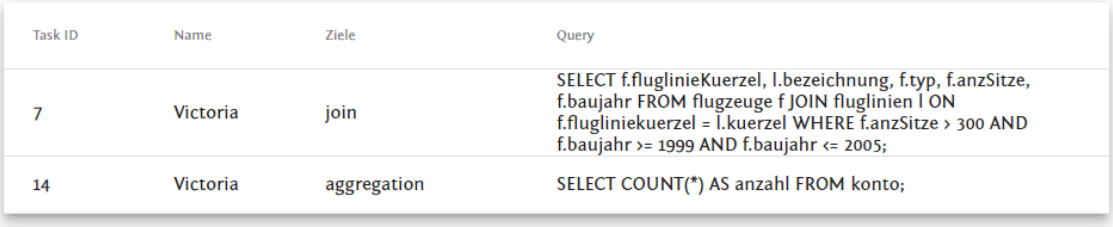


Abbildung 5.25: Lernfortschrittsbestätigungen

Um überprüfen zu können, ob die Aufgabe selbst gelöst wurde, also die Lösung nicht ident mit Lösungen anderer Studierenden ist, muss bei der Lernfortschrittsbestätigung auch die jeweilige Lösung mitgespeichert werden. In Abbildung 5.26 sieht man die Detailansicht einer Lernfortschrittsbestätigung.



Task ID	Name	Ziele	Query
7	Victoria	join	SELECT f.fluglinieKuerzel, l.bezeichnung, f.typ, f.anzSitze, f.baujahr FROM flugzeuge f JOIN fluglinien l ON f.fluglinienkuerzel = l.kuerzel WHERE f.anzSitze > 300 AND f.baujahr >= 1999 AND f.baujahr <= 2005;
14	Victoria	aggregation	SELECT COUNT(*) AS anzahl FROM konto;

Abbildung 5.26: Lernfortschrittsbestätigungen - Details

Damit die Aufgabenklassen auch erreicht werden können, müssen dazugehörige Aufgaben existieren. Diese können die AdministratorInnen anlegen, verwalten und löschen. Bei einer Aufgabe wird eine textuelle Aufgabenbeschreibung angegeben, zu welcher Aufgabenfamilie sie gehört, eine Musterlösung und welche Aufgabenklassen mit dieser Aufgabe erreicht werden. Abbildung 5.27 zeigt das Formular zur Verwaltung von Aufgaben.

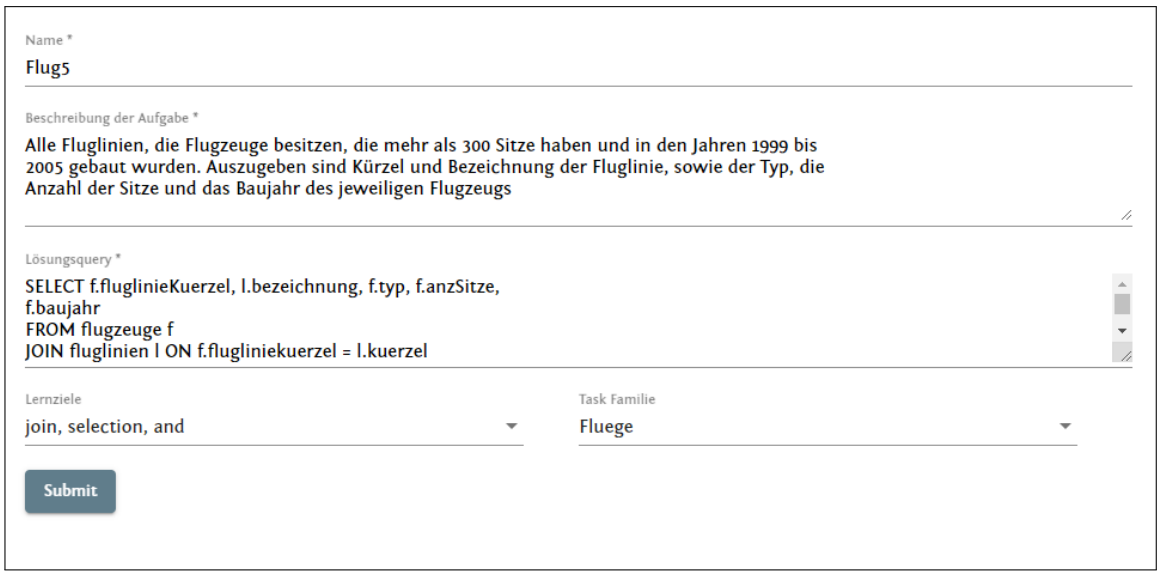


Abbildung 5.27: Verwaltungsseite Aufgaben

5.3 Diskussion Zielerreichung

In diesem Abschnitt werden die Anforderungen, welche in Abschnitt 1.4 beschrieben wurden, in Bezug auf das *Frontend* näher beschrieben. Dazu wird auf die einzelnen Unterpunkte näher eingegangen.

A1 Individualisierung Aufgabenliste

Studierende sollen die Möglichkeit haben, die Aufgabenlistengenerierung an ihre individuellen Bedürfnisse anzupassen. Für die Umsetzung dieser Anforderung ist im Frontend nur die *Studierenden-Sicht* relevant.

A1.1 Einstellungsmöglichkeit Aufgabenfamilie
Jede Aufgabe wird genau einem Aufgabenbereich (Aufgabenfamilie) zugeordnet. Studierende haben die Möglichkeit einzustellen, wie wichtig ihnen kein Wechsel zwischen Aufgabenfamilien ist. Sie können dabei einen Wert zwischen 0 und 5 einstellen, mit welchem entschieden wird, ob ein Wechsel zwischen Aufgabenfamilien stattfinden soll oder nicht. Dabei steht ein Wert von 0 für nicht wichtig und ein Wert von 5 für sehr wichtig. In Abbildung 5.9 wird die visuelle Umsetzung für die Einstellungsmöglichkeit der Aufgabenfamilie dargestellt.

A1.2 Einstellungsmöglichkeit Schwierigkeitsgrad
Die Einstellungsmöglichkeit für den Schwierigkeitsgrads der Aufgabenliste wird in Abbildung 5.14 dargestellt. Dabei können die Studierenden einen Wert zwischen 1 und 15 wählen, 1 steht für sehr einfach und 15 für sehr schwer. Mit dem eingestellten Schwierigkeitsgrad ändert sich vor allem die Anzahl der zu absolvierenden Aufgaben. So kann es etwa sein, dass mit Schwierigkeitsgrad 2 fünf Aufgaben absolviert werden müssen und für dieselbe Aufgabensammlung mit einem Schwierigkeitsgrad von 10 nur zwei Aufgaben erledigt werden müssen.

A1.3 Anpassung Aufgabenliste
Damit eine Aufgabenliste etwa nach dem Ändern der Präferenzen angepasst werden kann, gibt es den Menüpunkt *Neue Aufgabenliste*. Hier können sich die Studierenden jederzeit eine neue Aufgabenliste für die aktuelle Aufgabensammlung generieren lassen. Oft ist es schwierig, zu entscheiden, was der ideale Schwierigkeitsgrad für einen selbst ist. Daher können Studierende einstellen, dass sie nach erfolgreicher Absolvierung einer Übungsaufgabe nach dem Schwierigkeitsgrad zur Aufgabe befragt werden. Dieser Dialog kann in den Einstellungen ein- bzw. ausgeschaltet werden.

A2 Wissenenstand der Studierenden

Für die Studierenden ist es von Vorteil zu wissen, was ihr aktueller Wissensstand ist und wie sie sich entwickelt haben. Daher gibt es bezüglich des Wissensstand der Studierenden drei Anforderungen, deren Erfüllung durch das Frontend im folgenden Abschnitt beschrieben werden.

A2.1 Wissensstand Aufgabenklassen
In der Generalisierungshierarchie aus 1.1 sind alle möglichen Aufgabenklassen aufgegliedert, welche von den Studierenden erreicht werden können. Damit die Studierenden auch ihren Wissensstand einsehen können, wird ihnen eine Übersicht mit allen Aufgabenklassen, welche sie bereits erreicht haben und jenen, welche noch ausstehend sind, angezeigt (Abbildung 5.5). In dieser Hierarchie wird jedoch auch berücksichtigt, dass bei der Absolvierung einer untergeordneten Aufgabenklasse auch die übergeordnete Aufgabenklasse geübt wurde. Daher werden in der Übersicht sowohl die *Parent-* als auch die *Child-Klasse* Grün markiert.

A2.2 Wissensstand und Aufgabenlistengenerierung
Diese Anforderung ist in Bezug auf das Frontend irrelevant. Bei dieser Anforderung soll sichergestellt werden, dass vergangene Abfragen zukünftige Abfragen beeinflussen und eine Aufgabenklasse nicht zweimal erlernt werden muss. Diese Anforderung wird mit dem konzeptuellen Entwurf und dem Backend umgesetzt, jedoch wird sie nicht speziell visuell dargestellt.

A2.3 Vergangene Abfragen
Damit Studierende einsehen können, welche Aufgaben sie bereits mit welchem Statement versucht haben, wird ihnen eine Liste aller Lösungsversuche angezeigt. Dabei wird unterschieden, ob sie auf Query Ausführen (execute) oder Abschicken (check) geklickt haben. Diese Übersicht dient auch als Lernhinweis, da Studierenden sehen können, bei welchen Bereichen sie noch Probleme haben und daher mehr Versuche benötigen und welche sie mit wenigen Versuchen lösen konnten. In Abbildung 5.8 wird die visuelle Umsetzung dieser Anforderung gezeigt.

A3 Wahrung der Privatsphäre

Die entwickelte Anwendung soll die Privatsphäre der Studierenden bewahren und es soll darauf geachtet werden, dass nur relevante Daten an die ÜbungsleiterInnen weitergegeben werden. In diesem Abschnitt wird die Wahrung der Privatsphäre in Bezug auf das Frontend näher betrachtet.

A3.1 Privates Wissen über die Studierenden
Diese Anforderung ist in Bezug auf das Frontend irrelevant. Die Trennung der Daten

und das Speichern der privaten Daten der Studierenden am eigenen Computer wird bereits mit der entwickelten Architektur, dem konzeptuellen Modell und dem Backend umgesetzt und wird nicht visuell dargestellt.

A3.2 Lernfortschrittsbestätigung

Die Lernfortschrittsbestätigungen sollen nur ein Minimum der erforderlichen Daten enthalten. Damit die ÜbungsleiterInnen beurteilen können, ob eine Aufgabensammlung erfolgreich absolviert wurde, benötigen Sie Informationen über die durchgeführten Aufgaben. Wie Abbildung 5.26 zeigt, wird den ÜbungsleiterInnen nur übermittelt, welche Aufgabe mit welchen Statement gelöst wurde, jedoch nicht, welche Aufgaben von den Studierenden zuvor absolviert wurden, um etwa den Schwierigkeitsgrad zu reduzieren. Zudem wird nicht mitgesendet, wie oft eine Aufgabe probiert wurde, bis sie erfolgreich absolviert werden konnte.

Zusammenfassung

In diesem Kapitel wurden die verschiedenen Darstellungen der Backend-Funktionalitäten im Frontend beschrieben. Dabei wurde zwischen der Sicht für die *Studierenden* und jener für die *Administratoren* unterschieden. Die verschiedenen Darstellungen sind nur für Benutzer zugänglich, welche die entsprechende Rolle haben. Zudem wurden die in Abschnitt 1.4 beschrieben Anforderungen in Bezug auf das Frontend diskutiert.

Kapitel 6

Wissensbasierte Aufgabenzuteilung

In diesem Kapitel wird der Task-Assignment-Generator (TAG) näher beschrieben. Er ist für die wissensbasierte Aufgabenzuteilung zuständig. Wie bereits in Kapitel 2.5 beschrieben, wurde der Task-Assignment-Generator als DLV-Programm umgesetzt. Das DLV-Programm besteht aus Regeln, Fakten und Constraints. Es wird versucht, jene Aufgabenliste zu finden, welche mit den Präferenzen der Studierenden am meisten übereinstimmt. Dabei werden alle möglichen Lösungen berechnet und jene Aufgabenliste mit den geringsten *Kosten* ausgewählt. Diese *Kosten* setzen sich aus unterschiedlichen Einstellungen, welche mithilfe von Constraints, Regeln und Fakten modelliert werden, zusammen.

6.1 Fakten

Der Großteil der Fakten wird direkt aus der Datenbank geladen und dem DLV-Programm über einen Java-Wrapper hinzugefügt. Für ein besseres Verständnis werden Ausschnitte des DLV-Programms beschrieben.

Mithilfe der nachfolgenden Fakten werden Aufgabenklassen samt jeweils *übergeordneter Aufgabenklasse* und den *angenommenen Lernkosten* beschrieben. Dabei ist der erste Parameter die *Parent-Klasse* und der zweite Parameter die *Aufgabenklasse* selber. Da auch die *Aufgabenklasse* wieder von einer anderen *Aufgabenklasse* die *Parent-Klasse* sein kann, wird so die Generalisierungshierarchie abgebildet.

Nachfolgendes Beispiel zeigt, dass die Aufgabenklasse *join* sowohl die *Child-Klasse* von *projection*, als auch die *Parent-Klasse* von *outerjoin* ist.

```
%c(<parentconstruct>, <construct>, <learningcosts>)
c(projection, join, 4).
c(join, outerjoin, 4).
```

Es werden nun alle Aufgabenklassen gezeigt, welche bei der Generierung der Aufgabenliste direkt über die Datenbank eingefügt werden. Durch die verschiedenen Verknüpfungen untereinander (*Parent-Child-Klasse*) entsteht so die Generalisierungshierarchie.

```
c(projection, join, 4).
 c(join, outerjoin, 4).
 c(projection, group, 2).
 c(group, group_having, 3).
 c(projection, sorting, 2).
 c(projection, aggregation, 2).
 c(join, thetajoin, 4).
 c(join, selfjoin, 4).
 c(projection, duplicates, 1).
 c(projection, distinct, 1).
 c(projection, rename, 1).
 c(projection, setoperations, 4).
 c(setoperations, minus, 2).
 c(setoperations, intersect, 2).
 c(setoperations, union, 2).
 c(aggregation, min, 1).
 c(aggregation, max, 1).
 c(aggregation, sum, 1).
 c(aggregation, avg, 1).
 c(aggregation, count, 1).
 c(projection, selection, 2).
 c(selection, and, 1).
 c(selection, or, 1).
 c(selection, between, 1).
 c(selection, like, 1).
 c(projection, projection, 2).
 c(join, innerjoin, 1).
 c(join, equijoin, 1).
 c(join, multijoin, 1).
 c(outerjoin, leftjoin, 2).
 c(outerjoin, rightjoin, 2).
 c(outerjoin, fulljoin, 6).
 c(aggregation, countDistinct, 1).
 c(sorting, soringDirection, 1).
```

Mit *g(<construct>)* wird angegeben, welche Aufgabenklassen von den Studierenden erreicht werden sollen. Diese Ziele werden von der aktuell zu lösenden Aufgabensammlung übermittelt. Nachfolgende Fakten geben an, dass *outerjoin* und *aggregation* geübt werden

sollen.

```
35 g(outerjoin).
36 g(aggregation).
```

Um angeben zu können, welche Aufgaben (mit welcher Aufgabenfamilie) existieren und welche Aufgabenklassen damit gelöst werden, gibt es die Fakten *x(<example>,<family>)* und *x_c(<example>, <construct>)*. Dabei zeigt *x(<example>,<family>)* zu welcher Familie die Aufgabe gehört und *x_c(<example>,<construct>)* welche Aufgabenklasse mit welcher Aufgabe erreicht werden können. So zeigen etwa die nachfolgenden Ausschnitte, dass Aufgabe *flug6* zu der Aufgabenfamilie *fluege* gehört und dass die Aufgabenklassen *selection* und *projection* mit dem erfolgreichen Absolvieren erreicht werden.

```
37 x(flug6, fluege).
38 x(flug7, fluege).
39 x(flug8, fluege).
40 x(buchung3, buchung).
41 x(buchung4, buchung).
42 x(buchung1, buchung).
43 x(buchung2, buchung).
```

```
44 x_c(flug6, selection).
45 x_c(flug6, projection).
46 x_c(flug7, distinct).
47 x_c(flug7, selection).
48 x_c(flug7, projection).
49 x_c(flug8, minus).
50 x_c(flug8, selection).
51 x_c(buchung3, and).
52 x_c(buchung3, like).
53 x_c(buchung4, multijoin).
54 x_c(buchung4, soringDirection).
55 x_c(buchung1, count).
56 x_c(buchung2, group).
57 x_c(buchung2, count).
```

Studierende können Aufgabenklassen bereits durch vorherige Aufgaben erreicht haben. Diese werden in *reached_pre* gespeichert. Nachfolgendes Beispiel zeigt, dass bereits *count* mit einer anderen Aufgabe erlernt wurde.

```
58 reached_pre(count).
```

Um eine Aufgabenliste auf maximal 5 Schritte zu beschränken, werden 5 Fakten mit Prädikat *s(<step>)* angelegt. Jeder Fakt steht für einen möglichen Schritt in der Aufgabenliste.

```
59 s(1).
60 s(2).
```

```
61 s(3).
62 s(4).
63 s(5).
```

6.2 Regeln

Die Logik des DLV-Programms wird mit verschiedenen Regeln beschrieben. Folgende disjunktive Regel drückt aus, dass bei jedem Schritt in der Aufgabenliste entweder eine der möglichen Aufgaben erledigt wird oder nichts durchgeführt wird (idle). Nachfolgendes Beispiel zeigt, dass in jedem Schritt entweder die Aufgaben *flug6, flug,7, flug8, buchung3, buchung4, buchung1* oder *buchung2* durchgeführt werden können.

```
64 idle(S)
65  v t(S,flug6)
66  v t(S,flug7)
67  v t(S,flug8)
68  v t(S,buchung3)
69  v t(S,buchung4)
70  v t(S,buchung1)
71  v t(S,buchung2)
72 :- s(S).
```

Die Aufgabenklassen sind in einer Hierarchie eingeordnet und sie können bereits als *absolviert* gelten, wenn die *Child-Klasse* gelöst wurde. Daher müssen alle Aufgabenklassen gesucht werden, welche auf diesem Weg abgeschlossen wurden. Dafür gibt es nachfolgende Regeln: Es wird zuerst die *reached_c_clsr* Regel erstellt, welche aus allen zu erlernenden Aufgabenklassen und den übergeordneten *Parent-Klassen* besteht. Anschließend wird die Regel *reached_pre_clsr* erstellt, welche überprüft, ob eine bereits erlernte Aufgabenklasse (*reached_pre*) in der Liste der zu erlernenden Aufgabenklassen *reached_c_clsr* enthalten ist.

```
73 reached_c_clsr(C,C) :- c(_,C,_).
74 reached_c_clsr(C,C3) :- c(C2,C,_), reached_c_clsr(C2,C3).
75
76 reached_pre_clsr(C2) :- reached_pre(C),  reached_c_clsr(C,C2).
```

Da nun alle Aufgabenklassen gefiltert wurden, welche bereits direkt oder indirekt absolviert wurden, können die noch zu lösenden Aufgabenklassen ermittelt werden und in *c2* gespeichert werden. Dazu werden alle Aufgabenklassen von c verwendet und überprüft, ob sie nicht in *reached_pre_clsr* enthalten sind.

```
78 c2(P,C,D) :-  c(P,C,D), not reached_pre_clsr(C).
```

Nun können alle möglichen erlernbaren Aufgabenklassen ermittelt werden. Dazu wird überprüft, ob sich die Aufgabenklasse oder die übergeordnete *Parent-Klasse* in der Liste aller zu erlernenden Aufgabenklassen (*c2*) befindet.

```
79 c_clsr(C,C) :- c2(_,C,_).
80 c_clsr(C,C3) :- c2(C2,C,_), c_clsr(C2,C3).
```

Auf Basis aller möglichen Aufgabenklassen werden anschließend alle Aufgabenklassen (über die Hierarchie) ermittelt, welche mit einer Aufgabe erlernt werden können.

```
81 x_c_clsr(X,C2) :- x_c(X,C), c_clsr(C,C2).
```

Jetzt weiß man, welche Aufgabenklassen mit welchen Aufgaben erreicht werden können, daher kann ermittelt werden, welche Aufgabenklassen bei welchem Schritt erreicht werden können.

```
82 g_reachedby(S,C) :-
83     s(S),
84     t(S2,X),
85     S2 <= S,
86     x_c_clsr(X,C).
```

Damit bei der Generierung der Aufgabenliste auf die Präferenzen der Studierenden eingegangen werden kann, wird mithilfe der nachfolgenden Regel überprüft, ob ein Wechsel zwischen Aufgabenfamilien stattfindet. Mit dem später gezeigten schwachen Constraint, werden die Kosten für einen Wechsel zwischen Aufgabenfamilien (Familie-Switch) definiert.

```
87 familyswitch(S2) :-
88   t(S1,X1), t(S2,X2),
89   #succ(S1,S2),
90   x(X1,F1), x(X2,F2),
91   F1 <> F2.
```

Zudem können Studierende den gewünschten Schwierigkeitsgrad einstellen, dazu muss zuerst herausgefunden werden, welche Aufgabenklasse im vorherigen Schritt erlernt wurde.

```
92 g_reachedbefore(S,C) :-  g_reachedby(S2,C), #succ(S2,S).
```

Anschließend können die zusätzlichen Aufgabenklassen (aus der Hierarchie) einer Aufgabe, welche noch nicht erreicht wurden, herausgefunden werden.

```
93 g_additional(S,C) :-
```

```
94      t(S,X),
95      x_c_clsr(X,C),
96      not g_reachedbefore(S,C).
```

Mit diesen Informationen kann nun der relative Schwierigkeitsgrad eines Schritts in der Aufgabenliste berechnet werden.

```
97  relativeDifficulty(S,RD) :-
98    t(S,X),
99    #sum{D, C: g_additional(S,C), c(P,C,D) } = RD,  #int(RD)
```

Damit man weiß, welche Ziele bereits erreicht bzw. nicht erreicht worden sind, muss eine *goal_closure* erstellt werden, anschließend können alle nicht erreichten Aufgabenklassen (*not_g_reached*) herausgefiltert werden.

```
100 g_clsr(C2) :- g(C),  c_clsr(C,C2).
101
102 not_g_reached(C) :- g_clsr(C), not g_reached(C).
```

Sollten Elemente in *not_g_reached* enthalten sein, wird den Studierenden nach der Generierung der Aufgabenliste angezeigt, dass die Aufgabenliste unvollständig ist. Sie können nach der Absolvierung der ersten Aufgabenliste die Generierung erneut starten, um für die restlichen Aufgabenklassen eine Aufgabenliste zu erhalten.

6.3 Constraints

In diesem Abschnitt werden die einzelnen Constraints des DLV-Programms beschrieben.

Das Ziel ist es, dass Studierende mit einer Aufgabenliste alle Aufgabenklassen absolvieren können. Es ist jedoch nicht zwingen notwendig, dass alle Aufgabenklassen in einer Aufgabenliste erfüllt werden. Sollte eine Aufgabenklasse nicht erfüllt werden (es befindet sich in der Sammlung aller nicht erreichten Ziele *not_g_reached* ein Eintrag), so wird dieses mit extra Kosten gewichtet. Bei der Generierung der Aufgabenliste wird dann eher versucht alle Aufgabenklassen zu erreichen, es ist aber auch möglich, dass eine unvollständige Aufgabenliste generiert wird. Die Kosten für nicht erreichte Aufgabenklassen werden extra berechnet und sind das doppelte von ihren Lernkosten (LC):

```
103 :~ not_g_reached(C), c(_,C,LC), Costs = LC*2. [Costs:1]
```

Wichtig ist, dass eine Aufgabe nur einmal in der Aufgabenliste vorkommt. Um dies sicherzustellen, gibt es folgendes starke Constraint, welches festlegt, dass eine Aufgabe nicht

zweimal durchgeführt werden darf. t ist in diesem Fall der Task (Aufgabe) und S sind die Schritte. Daher darf z.B. die gleiche Aufgabe nicht bei Schritt 1 und Schritt 2 durchgeführt werden.

```
104 :- t(S1,X), t(S2,X), S1<>S2.
```

Auf Basis der Benutzer-Präferenzen wird angegeben, wie viel welcher Schwierigkeitsgrad kostet. Somit kann gesteuert werden, welche Aufgaben ausgewählt werden. Nachfolgendes Beispiel zeigt nur einen Ausschnitt bei welchen Aufgaben mit einem Schwierigkeitsgrad von 6 bevorzugt werden. Daher haben Aufgaben mit einem Schwierigkeitsgrad von 6 Kosten von 6 und damit einen Faktor von 1. Aufgaben mit einer Schwierigkeitsgrad von 2 haben hingegen die Kosten von 4 und damit einen Faktor von 2. Auch Aufgaben mit einem Schwierigkeitsgrad von 12 haben Kosten von 24 und ebenfalls damit einen Faktor von 2. Es wird daher versucht, Aufgaben auszuwählen, welche ungefähr einen Faktor von 1 haben.

```
105 :~ relativeDifficulty(T, 0).   [2:1]
106 :~ relativeDifficulty(T, 1).   [3:1] % Faktor: 3
107 :~ relativeDifficulty(T, 2).   [4:1] % Faktor: 2
108 :~ relativeDifficulty(T, 3).   [5:1]
109 :~ relativeDifficulty(T, 4).   [5:1]
110 :~ relativeDifficulty(T, 5).   [6:1]
111 :~ relativeDifficulty(T, 6).   [6:1] % Faktor: 1
112 :~ relativeDifficulty(T, 7).   [7:1]
113 ...
114 :~ relativeDifficulty(T,10).  [15:1]
115 :~ relativeDifficulty(T,11).  [20:1]
116 :~ relativeDifficulty(T,12).  [24:1] % Faktor: 2
117
118 ...
119 :~ relativeDifficulty(T,16).  [51:1]
120 :~ relativeDifficulty(T,17).  [56:1]
121 :~ relativeDifficulty(T,18).  [62:1] % Faktor: 3
```

Sobald in der Aufgabenliste einmal *idle* vorkommt werden keine Aufgaben mehr erledigt.

```
123 :- idle(S), t(S2,X), S2>S.
```

Studierende können mit nachfolgendem Prädikat angeben, wie viel ein Wechsel zwischen Aufgabenfamilien kostet. Dabei wird *user_family_swtich_costs* zur Laufzeit durch das Java-Programm durch den eingestellten Wert der Studierenden ersetzt.

```
124 :~ familyswitch(S). [user_family_switch_costs:1]
```

6.4 Diskussion Zielerreichung

Im nachfolgendem Abschnitt werden auf die in Abschnitt 1.4 beschrieben Ziele des entwickelten Prototyps eingegangen. Dabei wird jedes Ziel auf seine konkrete Umsetzung in Bezug auf die wissensbasierte Aufgabenzuteilung (Task-Assignment-Generator) betrachtet.

A1 Individualisierung Aufgabenliste

Für Studierende soll es verschiedene Einstellungsmöglichkeiten geben, mit welchen sie die Aufgabenliste an ihre individuellen Bedürfnisse anpassen können. In diesem Abschnitt wird auf der Erfüllung der Anforderungen in Bezug auf den Task-Assignment-Generator näher eingegangen.

A1.1 Einstellungsmöglichkeit Aufgabenfamilie
Studierende können entscheiden, ob es ihnen wichtig ist, dass alle Aufgaben dieselbe Aufgabenfamilie besitzen oder ob auch mehrere Aufgabenfamilien verwendet werden dürfen. Dies wird mit dem Prädikat *familyswitch* umgesetzt. Dabei werden die Kosten für einen Wechsel zwischen Aufgabenfamilien auf einen Wert, welcher von den Studierenden zwischen 0 und 5 gewählt werden kann, gesetzt.

A1.2 Einstellungsmöglichkeit Schwierigkeitsgrad
Die Studierenden können in ihren Benutzer-Präferenzen den gewünschten durchschnittlichen Schwierigkeitsgrad einer Aufgabe festlegen. Dieser wird mithilfe der schwachen Constraints *relativeDifficulty* umgesetzt. Aufgaben, welche dem gewünschten Schwierigkeitsgrad entsprechen, werden dabei niedrigere Kosten hinzugerechnet, als Aufgaben, welche einen anderen Schwierigkeitsgrad besitzen.

A1.3 Anpassung Aufgabenliste
Studierende sollen die Möglichkeit haben, die Aufgabenliste mit angepassten Präferenzen neu generieren zu lassen. Diese Anforderung ist in Bezug auf die wissensbasierte Aufgabenzuteilung nicht relevant, da eine erneute Generierung für Studierende immer möglich ist, jedoch dafür eine Schnittstelle im Backend und Frontend bereitgestellt werden muss.

A2 Wissensstand der Studierenden

Lernplattformen sammeln eine Vielzahl an Informationen zum Wissensstand der Studierenden. Diese sollen den Studierenden zur Verfügung gestellt werden. In diesem Abschnitt wird auf die Erfüllung der Anforderungen in Bezug auf den Task-Assignment-Generator näher eingegangen.

A2.1 Wissensstand Aufgabenklasse
Studierende sollen sich ihren aktuellen Wissensstand in Bezug auf die Aufgabenklassen ansehen können. Diese Anforderung ist für die wissensbasierte Aufgabenzuteilung nicht relevant, da hier nur die Generierung der Aufgabenliste erfolgt, jedoch keine Visualisierung. Diese Anforderung wird mit dem Frontend umgesetzt.

A2.2 Wissensstand und Aufgabenlistengenerierung
Eine Aufgabenklasse, welches bereits erfolgreich erlernt wurde, soll nicht noch einmal erlernt werden müssen. Wie man mit dem Fakt *reached_pre* sieht, werden bereits erlernte Aufgabenklassen bei der Generierung der Aufgabenliste berücksichtigt und müssen nicht noch einmal absolviert werden.

A2.3 Vergangene Abfragen
Bei dieser Anforderung sollen Studierenden sich ihre vergangenen Abfragen erneut ansehen können, um so aus ihren Fehlern lernen zu können. Auch diese Anforderung ist in Bezug auf die Aufgabenzuteilung nicht relevant, da es sich hierbei wieder um eine grafische Darstellung handelt. Diese Anforderung wird mit dem Backend ermöglicht und im Frontend visualisiert.

A3 Wahrung der Privatsphäre

Die entwickelte Anwendung soll die Privatsphäre der Studierenden bewahren. Es soll das private Wissen der Studierenden nur lokal gespeichert werden und nur das notwendige Minimum der Daten an die ÜbungsleiterInnen weitergegeben werden. Wie bereits in Kapitel *3 Architektur* beschrieben, läuft die wissensbasierte Aufgabenzuteilung ebenso wie das Task-Assignment-Generator-Service nur bei den Studierenden lokal. Dadurch sind die Informationen, welche für die Aufgabenzuteilung notwendig sind, nur für Studierende verfügbar.

A3.1 Privates Wissen über die Studierenden
Die konkrete Anforderung für das private Wissen über die Studierenden ist in Bezug auf die wissensbasierte Aufgabenzuteilung nicht relevant, da hier nur die Aufgabenliste generiert wird, jedoch kein Wissen gespeichert wird. Diese Anforderung wurde mithilfe der Architektur und dem Backend umgesetzt.

A3.2 Lernfortschrittsbestätigung
Die Entscheidung, dass der Task-Assignment-Generator lokal bei den Studierenden läuft, wurde aufgrund der Privatsphäre der Studierenden gewählt. So kann sichergestellt werden, dass bei der Aufgabenzuteilung keine Daten an Unbefugte weitergegeben werden. Die weiteren Anforderungen für die Lernfortschrittsbestätigung sind jedoch nicht relevant in Bezug auf die wissensbasierte Aufgabenzuteilung. Da hier keine Bestätigungen abgegeben werden, sondern nur die notwendige Aufgabenliste generiert wird.

Zusammenfassung

In diesem Kapitel wurde die wissensbasierte Aufgabenzuteilung genauer beschrieben. Dabei wurde auf die verschiedenen Regeln, Fakten und Constraints eingegangen, mit welchen unter anderem die Benutzer-Präferenzen abgebildet werden. Zudem wurden mithilfe von Beispiel-Ausschnitten gezeigt, wie die Aufgabenzuteilung im Detail umgesetzt wurde. Zum Schluss wurden die in Abschnitt 1.4 beschriebenen Anforderungen in Bezug auf die wissensbasierte Aufgabenzuteilung diskutiert.

Kapitel 7

Überprüfung der Zielerreichung

In diesem Kapitel werden die Anforderungen, welche in Abschnitt 1.4 beschrieben wurden, auf ihre konkrete Umsetzung überprüft. In den einzelnen Entwicklungskapiteln wurden die Anforderungen bereits auf die jeweilige Umsetzung diskutiert, nun soll allgemeine überprüft werden, ob die Ziele erreicht wurden. In nachfolgender Tabelle 7.1 wird gezeigt, welche Anforderung in welchem Kapitel umgesetzt wurde. Ist ein Feld *grau* hinterlegt, so ist diese Anforderung für dieses Kapitel nicht relevant.

Anforderung	**Architektur**	**Backend**	**Frontend**	**Zuteilung**
A1.1 Aufgabenfamilie	✓	✓	✓	✓
A1.2 Schwierigkeitsgrad	✓	✓	✓	✓
A1.3 Aufgabenliste Anpassung	✓	✓	✓	
A2.1 Aufgabenklasse	✓	✓	✓	
A2.2 Aufgabenlistengenerierung	✓	✓		✓
A2.3 Vergangene Abfragen	✓	✓	✓	
A3.1 Privates Wissen	✓	✓		
A3.2 Lernfortschrittsbestätigung	✓	✓	✓	

Tabelle 7.1: Überblick Anforderungen

7.1 Individualisierung Aufgabenliste

In diesem Abschnitt wird auf die Erfüllung der Anforderungen in Bezug auf die Individualisierung der Aufgabenliste eingegangen. Dabei sollen Studierende verschiedene Möglichkeiten haben, die Aufgabenliste an ihre individuellen Bedürfnisse anzupassen. Diese sind: die Entscheidung in Bezug auf den Wechsel zwischen Aufgabenfamilien, den Schwierigkeitsgrad einer Aufgabe und die Anpassung der Aufgabenliste.

A1.1 Einstellungsmöglichkeit Aufgabenfamilie
Diese Anforderung wird im konzeptuellen Modell berücksichtigt. Hier wird zum einen jeder Aufgabe eine Aufgabenfamilie zugeteilt und zum anderen haben Studierende mit der Klasse *Preferences* die Möglichkeit festzulegen, wie wichtig es ihnen ist, dass kein Wechsel zwischen Aufgabenfamilien stattfindet. Diese Architektur zieht sich dann weiter in das Backend und im Frontend haben Studierende eine Seite, auf welcher Sie ihre Präferenzen festlegen können. Bei der Generierung mithilfe des Task-Assignment-Generators wird noch der von den Studierenden festgelegte Wert für die Kosten eines Wechsels an die wissensbasierte Aufgabenzuteilung übermittelt. Durch die Einstellung der Studierenden wird dann entschieden, ob ein Wechsel zwischen Aufgabenfamilien stattfinden darf oder nicht.

A1.2 Einstellungsmöglichkeit Schwierigkeitsgrad
Studierende sollen die Möglichkeit haben, selbst zu entscheiden, welchen Schwierigkeitsgrad eine Aufgabe haben soll. Für diese Anforderung wurde die Klasse *Preferences* erstellt. Hier wird, beginnend beim konzeptuellen Modell, der eingestellte Wert der Studierenden gespeichert. Diese Klasse wird dann ebenfalls im Backend umgesetzt und mit einem entsprechenden Endpunkt dem Frontend zur Verfügung gestellt. Die Studierenden können auf einer visuellen Oberfläche den gewünschten Wert einstellen. Dieser Wert kann von 1 bis 15 definiert werden. Dabei steht 1 für sehr einfach und 15 für sehr schwierig. Der eingestellte Wert wird anschließend bei der wissensbasierten Aufgabenzuteilung berücksichtigt und bevorzugte Aufgaben ausgewählt, welche dem Wert entsprechen. Sollten die Studierenden einen Wert von 5 eingestellt haben, so werden sie, um zum Ziel zu gelangen, mehrere leichtere Aufgaben in ihrem Lernpfad. Bei einer Einstellung von 10 hingegen, werden sie weniger, jedoch schwerere Aufgaben erhalten. Diese Einstellungsmöglichkeit bedeutet, dass Studierende unterschiedlich viele Aufgaben erledigen müssen, um die vorgegeben Aufgabenklassen zu erreichen, jedoch am Ende alle Studierende zum gleichen Ziel gelangen.

A1.3 Anpassung Aufgabenliste
Oft ist es schwierig zu entscheiden, was der ideale Schwierigkeitsgrad für einen selbst ist. Daher können Studierende einstellen, dass sie nach erfolgreicher Absolvierung einer Übungsaufgabe nach dem Schwierigkeitsgrad zur Aufgabe befragt werden. Diese Einstellungsmöglichkeit ist ebenfalls in der Klasse *Preferences* im konzeptuellen

Modell umgesetzt worden. Zudem gibt es auch hier eine Schnittstelle im Backend, mit welcher das Frontend den Studierenden eine visuelle Darstellung anbietet. Studierende haben aber auch die Möglichkeit sich unter dem Menüpunkt *neue Aufgabenliste* jederzeit eine neue Aufgabenliste generieren zu lassen. Bereits erreichte Aufgabenklassen werden bei einer neuen Generierung berücksichtigt. In der wissensbasierte Aufgabenzuteilung werden bereits erreichte Aufgabenklassen mit dem Fakt *reached_pre* abgedeckt. Bei der Generierung einer neuen Aufgabenliste werden nur mehr Aufgaben für Aufgabenklassen ausgewählt, welche noch nicht erreicht wurden.

7.2 Wissensstand der Studierenden

In diesem Abschnitt wird auf die Anforderung *Wissensstand der Studierenden* näher eingegangen. Für die Studierenden ist es von Vorteil zu Wissen, was ihr aktueller Wissensstand ist und wie sie sich entwickelt haben. Daher gibt es bezüglich des Wissens der Studierenden drei Anforderungen, welche nun näher beschrieben werden.

A2.1 Wissensstand Aufgabenklasse
Studierende sollen sich ihre Leistung in Bezug zu den Aufgabenklassen ansehen können. Sie sollen erkennen können, welche Aufgabenklassen sie bereits erreicht haben und welche noch ausstehend sind. Diese Anforderung wird bis auf die wissensbasierte Aufgabenzuteilung in allen Bereichen umgesetzt. Es wird mit dem konzeptuellen Modell in der Klasse *Confirmation* angezeigt, welche Aufgabenklassen bereits erreicht wurden. Diese Schnittstelle wird auch im Backend umgesetzt, sobald eine Aufgabe von den Studierenden erfolgreich absolviert wurde, wird eine Bestätigung für die enthaltenen Aufgabenklassen erstellt. Alle bestätigten Aufgabenklassen werden abschließend dem Frontend übermittelt und dieses bietet eine visuelle Darstellung an (Abbildung 5.5), in welcher man alle Aufgabenklassen sieht und farblich markiert ist, welche bereits erlernt wurden. Für die wissensbasierte Aufgabenzuteilung ist diese Anforderung nicht relevant.

A2.2 Wissensstand und Aufgabenlistengenerierung
Mit der Anforderung *Wissensstand und Aufgabenlistengenerierung* soll sichergestellt werden, dass bereits erlernte Aufgabenklassen zukünftige Aufgabenlistengenerierung beeinflussen und eine Aufgabenklasse nicht öfter erlernt werden muss. Auch dieses Ziel wurde im konzeptuellen Modell mit der Klasse *Confirmation* umgesetzt. Das Backend verfügt über den Funktionsbereich *Aufgabenklasse* und übermittelt der wissensbasierten Aufgabenzuteilung alle bestätigten Aufgabenklassen. Diese werden mit dem Fakt *reached_pre* abgebildet und so sichergestellt, dass bei der Generierung der Aufgabenliste keine Aufgabenklassen doppelt erlernt werden muss. Für das Frontend ist diese Anforderung nicht relevant, da es sich hier um keine visuelle Anforderung handelt.

A2.3 Vergangene Abfragen

Damit Studierende einsehen können, welche Aufgaben sie bereits mit welchen Statements versucht haben, wird ihnen eine Liste mit allen Lösungsversuchen aller vergangenen Abfragen angezeigt. Dabei wird unterschieden, ob sie auf Query Ausführen (*execute*) oder Abschicken (*check*) geklickt haben. Im konzeptuellen Modell wurde diese Anforderung mit der Klasse *Attempt* berücksichtigt. Auch im Backend wird eine entsprechende Schnittstelle *Aufgabenliste/log* bereitgestellt. Das Frontend bietet den Studierenden (Abbildung 5.8) eine visuellen Darstellung ihres Lernverlaufs an.

7.3 Wahrung der Privatsphäre

In diesem Abschnitt wird auf die Anforderungen in Bezug auf die Wahrung der Privatsphäre der Studierenden eingegangen. Studierende haben verschiedene Möglichkeiten um die Aufgabenliste an ihre individuellen Bedürfnisse anzupassen, dabei ist es jedoch wichtig, dass keine privaten Daten an unbefugte weitergegeben werden. Folgende Anforderung werden in Bezug auf die Privatsphäre der Studierenden an den Standalone-Prototypen gestellt:

A3.1 Privates Wissen über die Studierenden

Bei der Entwicklung der Anwendungen wurde die Microservice-Architektur eingesetzt. Dies bedeutet, dass jedes Service einen eigenen abgesteckten Aufgabenbereich hat. Dadurch wurde auch ermöglicht, ein eigenes Service für Studierende zu entwickeln, welches die privaten Daten der Studierenden enthält. Um zu gewährleisten, dass außer den Studierenden niemand anders Zugriff auf diese Daten hat, wurde entschieden, dieses Service nur als Java-Programm am Rechner der Studierenden laufen zu lassen. Da jedoch das Java-Programm eine nicht persistente In-Memory-Datenbank verwendet, sind die Daten nach dem Beenden des Programms verloren. Um die Daten zu sichern wird, nun nach gewissen Aktionen der Studierenden, von den lokalen Daten ein Backup erstellt und in der Maintenance-Service Datenbank gespeichert. Damit diese Daten wirklich nur für Studierende einsehbar sind, wird das Backup mit einem individuellen Schlüssel verschlüsselt. Diesen müssen die Studierenden bereits beim Starten des Task-Assignment-Generator-Services mitgeben und ist nur ihnen bekannt. Nach dem erneuten Starten des lokalen Service wird anschließend beim Maintenance-Server für den angemeldeten User abgefragt, ob bereits ein Backup vorhanden ist. Wenn eines vorhanden ist, wird versucht dieses mit dem eingegeben individuellen Schlüssel zu entschlüsseln. Sollte dies nicht möglich sein, wird der User wieder abgemeldet und ein Fehler ausgegeben. Somit wird sichergestellt, dass niemand die Daten von jemand anderes einsehen kann.

A3.2 Lernfortschrittsbestätigung

Die ÜbungsleiterInnen geben vor, welche Aufgabenklassen von den Studierenden erlernt werden sollen. Um die erfolgreiche Absolvierung auch bestätigen zu können, benötigen die Studierenden einen Nachweis, die *Lernfortschrittsbestätigung*. Studierende erhalten für jede erreichte Aufgabenklasse eine Bestätigung, welche mit einer

Signatur ausgestattet ist, um die Integrität zu gewährleisten. Bei dieser Bestätigung wird jedoch nur das Minimum der notwendigen Daten mitgesendet. So ist etwa bei einer Lernfortschrittsbestätigung nur gespeichert, welche Ziele mit welchem Query gelöst wurden. Es wird jedoch nicht mitgespeichert, wie oft eine Aufgabe versucht wurde und welche Vorschritte notwendig waren, um eine Aufgabe zu lösen.

7.4 Limitationen

Der aktuell entwickelte Standalone-Prototype wurde für die Datenbankabfragesprache SQL entwickelt. Damit auch andere Wissensbereiche abgefragt werden können, muss das System mit einem Task-Service für den neuen Wissensbereichs erweitert werden. Dazu müssen die Endpunkte die Angabe des Wissensbereichs unterstützen. Mit dieser Anpassung können anschließend neue Wissensbereiche auf dieselbe Art integriert werden und mehrere Wissensbereiche mithilfe der Anwendung überprüft werden. Zusätzlich muss für den jeweiligen Wissensbereich eine entsprechende Benutzeroberfläche entwickelt werden.

Kapitel 8

Fazit

Das Ziel dieser Arbeit war es, einen experimentellen Standalone-Prototyp für die wissensbasierte und vertraulichkeitsbewahrende Zuteilung von Übungsaufgaben in intelligenten tutoriellen Systemen zu entwickel. Der entwickelte Standalone-Prototyp geht auf die Bedürfnisse der Studierenden ein und bewahrt zudem die Privatsphäre.

Für den Prototypen wurden drei Backend-Services und ein zentrales Frontend auf Basis der Microservice-Architektur entwickelt. Dabei wurden für die unterschiedlichen Funktionalitäten jeweils ein eigenes Backend-Service entwickelt. Jedes Service hat nur die für ihn relevanten Daten zur Verfügung. Das Service, welches mit den Daten der Studierenden arbeitet, wurde so umgesetzt, dass es nur lokal bei den Studierenden zur Verfügung steht. Damit soll die Privatsphäre der Studierenden gewahrt werden. Die anderen beiden Services, welche für die allgemeine Verwaltung und für die Durchführung der Übungsaufgaben verantwortlich sind, sollen in Zukunft an einer zentralen Stelle verfügbar sein. Dies könnte etwa auf einem oder mehreren zentralen Servern sein. Als Benutzeroberfläche wurde ein zentrales Frontend entwickelt, welches die verfügbaren Daten für die User visualisiert.

Die Zuteilung der Übungsaufgaben zu den Studierenden wurde mithilfe von Answer Set Programming umgesetzt. Dabei wurden verschiedene Regeln und Fakten entwickelt, welche die Generierung der individuellen Aufgabenliste ermöglichen. Studierende haben verschiedene Einstellungsmöglichkeiten, welche anschließend bei der Generierung einer individuellen Aufgabenliste berücksichtigt werden.

Aktuell wird an der JKU Business School mit dem Projekt *eTutur++* ein ähnlicher Ansatz verfolgt. Bei diesem Projekt wurde jedoch nicht darauf geachtet, dass die Anwendung vertraulichkeitsbewahrend ist. Des Weiteren unterscheidet sich die Lernzieldarstellung des *eTutor++* und des Task-Assignment-Generators in dieser Arbeit. Ein weiterer Unterschied ist, dass der *eTutor++* mit RDF und SPARQL umgesetzt wurde und der Task-Assignment-Generator mit Answer Set Programming.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[1] W. Ma, O. O. Adesope, J. C. Nesbit, and Q. Liu, "Intelligent tutoring systems and learning outcomes: A meta-analysis.," *Journal of Educational Psychology*, vol. 106, no. 4, pp. 901–918, 2014.

[2] R. Nkambou, J. Bourdeau, and R. Mizoguchi, "Introduction: What Are Intelligent Tutoring Systems, and Why This Book?," in *Advances in Intelligent Tutoring Systems* (R. Nkambou, J. Bourdeau, and R. Mizoguchi, eds.), pp. 1–12, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[3] J. R. Carbonell, "AI in CAI: An Artificial-Intelligence Approach to Computer-Assisted Instruction," *IEEE Transactions on Man-Machine Systems*, vol. 11, no. 4, pp. 190–202, 1970.

[4] V. Caputi and A. Garrido, "Student-oriented planning of e-learning contents for Moodle," *Journal of Network and Computer Applications*, vol. 53, pp. 115 – 127, 2015.

[5] Y. Kats, ed., *Learning Management System Technologies and Software Solutions for Online Teaching: Tools and Applications*. IGI Global, 2010.

[6] Moodle, "About Moodle - MoodleDocs." `https://docs.moodle.org/37/en/About_Moodle`, July 2019. Online; accessed 25.07.2019.

[7] Blackboard, "Was ist blackboard learn? | blackboard-hilfe." `https://help.blackboard.com/de-de/Learn/Instructor/Getting_Started/What_Is_Blackboard_Learn`, 2021. Online; accessed 12.02.2021.

[8] Canvas, "Cloudbasiertes LMS I Canvas Lern-Software I Instructure." `https://www.instructure.com/canvas/de/hochschule`, June 2020. Online; accessed 26.06.2020.

[9] E. Aïmeur and H. Hage, "Preserving Learners' Privacy," in *Advances in Intelligent Tutoring Systems* (R. Nkambou, J. Bourdeau, and R. Mizoguchi, eds.), pp. 465–483, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[10] R. Nkambou, "Modeling the Domain: An Introduction to the Expert Module," in *Advances in Intelligent Tutoring Systems* (R. Nkambou, J. Bourdeau, and R. Mizoguchi, eds.), pp. 15–32, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[11] B. P. Woolf, "Student Modeling," in *Advances in Intelligent Tutoring Systems* (R. Nkambou, J. Bourdeau, and R. Mizoguchi, eds.), pp. 267–279, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[12] J. Bourdeau and M. Grandbastien, "Modeling tutoring knowledge," in *Advances in Intelligent Tutoring Systems* (R. Nkambou, J. Bourdeau, and R. Mizoguchi, eds.), pp. 123–143, Springer Berlin Heidelberg, 2010.

[13] A. Garrido and E. Onaindia, "Assembling Learning Objects for Personalized Learning: An AI Planning Perspective," *IEEE Intelligent Systems*, vol. 28, no. 2, pp. 64–73, 2013.

[14] C.-M. Chen, "Intelligent web-based learning system with personalized learning path guidance," *Computers & Education*, vol. 51, pp. 787–814, Sept. 2008. Publisher: Elsevier Ltd.

[15] S. D. Warren and L. D. Brandeis, "The right to privacy," vol. 4, no. 5, pp. 193–220, 1890.

[16] A. F. Westin, *Privacy and Freedom*. Atheneum, 1967.

[17] Y. Imine, A. Lounis, and A. Bouabdallah, "An accountable privacy-preserving scheme for public information sharing systems," *Computers & Security*, vol. 93, p. 101786, 2020.

[18] J. Andress, *The Basics of Information Security, Second Edition: Understanding the Fundamentals of InfoSec in Theory and Practice*. Syngress Publishing, 2nd ed., 2014.

[19] P. Schaar, "Privacy by Design," *Identity in the Information Society*, vol. 3, pp. 267–274, Aug. 2010.

[20] A. Beutelspacher, H. B. Neumann, and T. Schwarzpaul, *Kryptografie in Theorie und Praxis: mathematische Grundlagen für Internetsicherheit, Mobilfunk und elektronisches Geld*. Aus dem Programm Mathematik/Kryptografie, Wiesbaden: Vieweg + Teubner, 2., überarb. aufl ed., 2010. OCLC: 553762625.

[21] A. Schill and T. Springer, *Verteilte Systeme*. eXamen.press, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[22] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. USA: John Wiley & Sons, Inc., 2nd ed., 1996.

[23] T. Eiter, G. Ianni, and T. Krennwallner, "Answer Set Programming: A Primer," in *Reasoning Web. Semantic Technologies for Information Systems: 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures* (S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M.-C. Rousset, and R. A. Schmidt, eds.), pp. 40–110, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[24] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, *Answer Set Solving in Practice*. Morgan & Claypool Publishers, 2012.

[25] H. H. Hoos and E. Tsang, "Local Search Methods," in *Handbook of Constraint Programming* (F. Rossi, P. v. Beek, and T. Walsh, eds.), vol. 2 of *Foundations of Artificial Intelligence*, pp. 135 – 167, Elsevier, 2006. ISSN: 1574-6526.

[26] M. Alviano, W. Faber, N. Leone, S. Perri, G. Pfeifer, and G. Terracina, "The Disjunctive Datalog System DLV," in *Datalog Reloaded* (O. de Moor, G. Gottlob, T. Furche, and A. Sellers, eds.), (Berlin, Heidelberg), pp. 282–301, Springer Berlin Heidelberg, 2011.

Anhang A

Installationsleitung

Maintenance-Service

Um das Maintenance-Service lokal laufen zu lassen, muss man sich von nachfolgendem Verzeichnis das maintenanceService-1.0.0.jar downloaden.
`https://github.com/v1c7ory/MaintenanceService/tree/master/src/lib`

Damit das Maintenance-Service funktioniert benötigt man eine PostgreSQL Datenbank. Diese kann hier heruntergeladen werden:
`https://www.postgresql.org/download/`
Nun kann eine neue Datenbank über *pgAdmin* mit dem Namen *maintenanceService* angelegt werden. Das Maintenance-Service verwendet folgende Einstellungen der PostgreSQL Datenbank:

```
1 spring.datasource.url=jdbc:postgresql://localhost:5432/maintenanceService
2 spring.datasource.username=postgres
3 spring.datasource.password=admin
```

Befehl: Die Anwendung wird mit folgendem Befehl ausgeführt:
java -jar maintenanceService-1.0.0.jar

Anpassungen: Sollte man andere Einstellungen für die PostgreSQL Datenbank verwenden, so kann man die durch das Ändern mit -D wie folgt, anpassen:
java -jar -Dspring.datasource.url=jdbc:postgresql://localhost:5432/{andereDatenbank}
maintenanceService-1.0.0.jar
Diese Anpassung kann mit allen oben genannten spring.datasource Attributen gemacht werden. Das Maintenance-Service ist am Port *8081* verfügbar. Dieser muss frei sein, damit das Maintenance-Service lauffähig ist. Gegebenenfalls kann dieser ebenfalls mit -Dserver.port=8081 beim Starten der Anwendung geändert werden. Die Anwendung wurde erfolgreich gestartet, wenn man den Output *Started MaintenanceServiceApplication* erhält.

Task-Service

Das Task-Service wird genau gleich wie das Maintenance-Service gestartet, jedoch ist es in einem anderen Verzeichnis verfügbar:
`https://github.com/v1c7ory/TaskService/tree/master/src/lib` das taskService-1.0.0.jar downloaden.

Damit das Task-Service funktioniert, benötigt man PostgreSQL. Diese kann hier heruntergeladen werden: `https://www.postgresql.org/download/`
Nun kann eine neue Datenbank über *pgAdmin* mit dem Namen *taskService* angelegt werden. Das Task-Service verwendet folgende Einstellungen der PostgreSQL Datenbank:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/taskService
spring.datasource.username=postgres
spring.datasource.password=admin
```

Das Task-Service benötigt für das Ver- und Entschlüsseln der Lernfortschrittsbestätigungen ein Public-Private Key-Paar. Sollte man bereits ein Schlüsselpaar besitzen, so kann der Pfad mit folgendem Befehl *-Dsecurity.key.path=C:/Privat/key/* angegeben werden. Besitzt man noch keine Schlüssel, werden beim Starten der Anwendung welche generiert, dafür muss man jedoch auch den gewünschten Pfad zum Ordner angeben.

Befehl: Die Anwendung wird mit folgendem Befehl ausgeführt:
java -jar -Dsecurity.key.path={Path} taskService-1.0.0.jar

Anpassungen: Sollte man andere Einstellungen für die PostgreSQL Datenbank verwenden, so kann man die durch das Ändern mit -D wie folgt, anpassen:
java -jar -Dspring.datasource.url=jdbc:postgresql://localhost:5432/{andereDatenbank}
-Dsecurity.key.path={Path} taskService-1.0.0.jar
Diese Anpassung kann mit allen oben genannten spring.datasource Attributen gemacht werden. Das Task-Service ist am Port *8083* verfügbar. Dieser muss frei sein, damit das Task-Service lauffähig ist. Gegebenenfalls kann dieser ebenfalls mit -Dserver.port=8083 beim Starten der Anwendung geändert werden. Die Anwendung wurde erfolgreich gestartet, wenn man den Output *Started TaskServiceApplication* erhält.

Task-Assignment-Generator-Service

Um das TAG-Service lokal laufen zu lassen, muss man sich von folgendem Verzeichnis
`https://github.com/v1c7ory/TaskAssignmentGenerator/blob/master/src/lib/`
das TaskAssignmentGenerator-1.0.0.jar downloaden.

Zudem benötigt man die aktuellste Version des DLV-Programms. Dieses kann man sich ebenfalls vom oben angegeben Verzeichnis downloaden. Dazu auf *tag_v06.dlv* klicken,

anschließend auf *Raw* nun kann man das File mit einem Rechtsklick - Speichern unter - downloaden. Beim Speichern muss man noch darauf achten, dass bei der Endung .dlv steht.

Nun benötigt man noch ein DLV-System, mit dem das DLV-Programm ausgeführt wird. Diese findet man ebenso im oben genannten Verzeichnis. Dazu auf *dlv.mingw-odbc.exe* klicken und downloaden.

Anschließend öffnet man die Eingabeaufforderung und wechselt in den Ordner, in welchem sich das heruntergeladene Jar befindet. Nun gibt man den nachfolgenden Befehl ein, wobei darauf geachtet werden muss, dass sich die mingw-odbc.exe und das DLV im selben Ordner wie das Jar befinden, ansonsten muss der Pfad zu den Dateien noch angegeben werden.
Wichtig nach dem Pfad zum DLV muss der individuelle geheime Private Key angegeben werden.

Befehl: java -jar TaskAssignmentGenerator-1.0.0.jar dlv.mingw-odbc.exe tag_v06.dlv {PrivateKey}

Die Anwendung wurde erfolgreich gestartet, wenn man den Output *Started TaskAssignmentGeneratorApplication* erhält.

Task-Assignment-Generator-Demonstrator

Der Source-Code für den Task-Assignment-Generator-Demonstrator ist unter folgendem Link verfügbar `https://github.com/v1c7ory/TaskDemonstrator`. Für diese Anwendung muss der gesamte Source-Code heruntergeladen werden, dazu auf *code* und anschließend *download zip*. Um das Task-Assignment-Generator-Service lokal laufen zu lassen, benötigt man eine Angular CLI. Dafür benötigt man wiederum Node.js. Dieses kann hier heruntergeladen werden `https://nodejs.org/de/`. Nun kann mithilfe des Befehls *npm install -g @angular/cli* die Angular CLI installiert werden. Eine genaue Beschreibung dazu findet man unter `https://angular.io/guide/setup-local`.

Um das Task-Assignment-Generator-Demonstrator-Service lokal laufen zu lassen, muss man im heruntergeladenen Code in den Unterordner *tag-demo* wechseln. Hier öffnet man die Eingabeaufforderung und führt zuerst den Befehl *npm install* aus. Sollte hier ein Fehler auftreten, kann das *package-lock.json* gelöscht werden und der Befehl erneut ausgeführt werden. Die Anwendung wird abschließend mit dem Befehl *ng serve* gestartet.

Die Anwendung ist auf *localhost:4200* verfügbar. Hier kann man sich nun einen neuen User erstellen. Um jedoch z.B. neue Aufgabensammlungen erstellen zu können, benötigt man einen User als Administrator. Da beim ersten Start der neuen Anwendung noch kein User mit solchen Rechten existiert, müssen diese Rechte direkt über die Datenbank bereitgestellt werden. Dazu in der Tabelle *user_roles* bei *roles* den Wert 0 eintragen.

Anhang B

Logisches Datenbankschema

Die Tabellen der Anwendungen werden mithilfe von *Spring Data JPA* beim ersten Start der jeweiligen Anwendung vollautomatisch erstellt. Im Folgenden werden die so generierten logischen Datenbankschemata je Service angegeben. Das automatisch generierte logische Datenbankschema des Maintenance-Services und Task-Services wurden mittels pgAdmin GUI exportiert.

Die Datenbank des Task-Assignment-Generator-Services ist eine H2 In-Memory-Datenbank. Da nach dem Beenden des Services die Daten der Studierenden nicht mehr verfügbar sind, wird ein Backup der lokalen Datenbank verschlüsselt in der Maintenance-Service Datenbank gespeichert. Zur Laufzeit des Task-Assignment-Generator-Services ist daher das logische Datenbankschema im File *file.sql* im Ordner *src/lib* verfügbar. Nachfolgend werden die logischen Datenbankschemata angegeben.

Maintenance-Service

```
CREATE TABLE public.exercise_goals
(
    id_sheet integer NOT NULL,
    id_goal integer,
    CONSTRAINT fkjmyoj67mr3ftwhdwte99wnbga FOREIGN KEY (id_sheet)
        REFERENCES public.exercise_sheet (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE public.exercise_goals
    OWNER to postgres;
```

```
CREATE TABLE public.exercise_sheet
(
    id integer NOT NULL,
    name character varying(255) COLLATE pg_catalog."default",
    active boolean,
    CONSTRAINT exercise_sheet_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE public.exercise_sheet
    OWNER to postgres;
```

```
CREATE TABLE public.learning_confirmation
(
    id integer NOT NULL,
    query character varying(1024) COLLATE pg_catalog."default",
    taskid integer NOT NULL,
    tag_user_id integer,
    "timestamp" timestamp without time zone,
    sheetid integer NOT NULL,
    learning_goalid integer NOT NULL,
    CONSTRAINT learning_confirmation_pkey PRIMARY KEY (id),
    CONSTRAINT fk3629yt9qoh1gceix2g8g9vbby FOREIGN KEY (tag_user_id)
        REFERENCES public.users (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE public.learning_confirmation
    OWNER to postgres;
```

```
CREATE TABLE public.user_roles
(
    user_id integer NOT NULL,
    roles integer,
    CONSTRAINT fkhfh9dx7w3ubf1co1vdev94g3f FOREIGN KEY (user_id)
        REFERENCES public.users (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE public.user_roles
    OWNER to postgres;
```

```
CREATE TABLE public.users
(
    id integer NOT NULL DEFAULT nextval('users_id_seq'::regclass),
```

```
    password character varying(255) COLLATE pg_catalog."default",
    username character varying(255) COLLATE pg_catalog."default",
    backup oid,
    CONSTRAINT users_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE public.users
    OWNER to postgres;
```

Task-Service

```
CREATE TABLE public.hibernate_sequences
(
    sequence_name character varying(255) COLLATE pg_catalog."default" NOT
    NULL,
    next_val bigint,
    CONSTRAINT hibernate_sequences_pkey PRIMARY KEY (sequence_name)
)

TABLESPACE pg_default;

ALTER TABLE public.hibernate_sequences
    OWNER to postgres;
```

```
CREATE TABLE public.learning_goal
(
    id integer NOT NULL,
    difficulty integer,
    name character varying(255) COLLATE pg_catalog."default",
    parent_construct_id integer,
    CONSTRAINT learning_goal_pkey PRIMARY KEY (id),
    CONSTRAINT fkk86jehlaeuwp5mvd6kr79a5nj FOREIGN KEY (
   parent_construct_id)
        REFERENCES public.learning_goal (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE public.learning_goal
    OWNER to postgres;
```

```
CREATE TABLE public.task
(
    id integer NOT NULL,
    difficulty_level integer,
    name character varying(255) COLLATE pg_catalog."default",
```

```
    result_query character varying(1024) COLLATE pg_catalog."default",
    statement_description character varying(1024) COLLATE pg_catalog."
    default",
    task_family_id bigint,
    CONSTRAINT task_pkey PRIMARY KEY (id),
    CONSTRAINT fksws3xd2jgmneg1p5ghyrhb51x FOREIGN KEY (task_family_id)
        REFERENCES public.task_family (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE public.task
    OWNER to postgres;
```

```
CREATE TABLE public.task_family
(
    id bigint NOT NULL,
    dbschema character varying(255) COLLATE pg_catalog."default",
    description character varying(500) COLLATE pg_catalog."default",
    name character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT task_family_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE public.task_family
    OWNER to postgres;
```

```
CREATE TABLE public.task_goals
(
    task_id integer NOT NULL,
    goal_id integer NOT NULL,
    CONSTRAINT fk88jhlv2ximhxr9xrfv8onsu8x FOREIGN KEY (goal_id)
        REFERENCES public.learning_goal (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fkjcyw3eqfp0yroh6lljxbdknk FOREIGN KEY (task_id)
        REFERENCES public.task (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE public.task_goals
    OWNER to postgres;
```

Task-Assignment-Generator-Service

```
CREATE TABLE GOAL_CONFIRMATION (
    ID INTEGER NOT NULL,
    DIFFICULTY INTEGER NOT NULL,
    LEARNING_GOAL_ID INTEGER NOT NULL,
    NAME VARCHAR(255),
    PARENT VARCHAR(255),
    QUERY CLOB,
    TASK_ID INTEGER NOT NULL,
    TIMESTAMP TIMESTAMP, PRIMARY KEY (ID));
```

```
CREATE MEMORY TABLE "PUBLIC"."HIBERNATE_SEQUENCES"(
    "SEQUENCE_NAME" VARCHAR(255) NOT NULL,
    "NEXT_VAL" BIGINT
);
ALTER TABLE "PUBLIC"."HIBERNATE_SEQUENCES" ADD CONSTRAINT "PUBLIC"."
    CONSTRAINT_5" PRIMARY KEY("SEQUENCE_NAME");
```

```
CREATE MEMORY TABLE "PUBLIC"."LEARN_PATH"(
    "ID" INTEGER DEFAULT (NEXT VALUE FOR "PUBLIC"."
    SYSTEM_SEQUENCE_4E27F73F_C241_4F0F_9212_805406C4C343") NOT NULL
    NULL_TO_DEFAULT SEQUENCE "PUBLIC"."
    SYSTEM_SEQUENCE_4E27F73F_C241_4F0F_9212_805406C4C343",
    "ACTIVE" BOOLEAN NOT NULL,
    "COMPLETE" BOOLEAN NOT NULL,
    "FINISHED" BOOLEAN NOT NULL,
    "SHEET_ID" INTEGER NOT NULL
);
ALTER TABLE "PUBLIC"."LEARN_PATH" ADD CONSTRAINT "PUBLIC"."CONSTRAINT_58"
     PRIMARY KEY("ID");
```

```
CREATE MEMORY TABLE "PUBLIC"."SETTINGS"(
    "ID" INTEGER DEFAULT (NEXT VALUE FOR "PUBLIC"."
    SYSTEM_SEQUENCE_8FA00267_665F_4E32_A238_4E13B3982A55") NOT NULL
    NULL_TO_DEFAULT SEQUENCE "PUBLIC"."
    SYSTEM_SEQUENCE_8FA00267_665F_4E32_A238_4E13B3982A55",
    "CURRENT_SHEET" INTEGER NOT NULL,
    "USER_NAME" VARCHAR(255)
);
ALTER TABLE "PUBLIC"."SETTINGS" ADD CONSTRAINT "PUBLIC"."CONSTRAINT_8"
    PRIMARY KEY("ID");
```

```
CREATE MEMORY TABLE "PUBLIC"."USER_PREFERENCES"(
    "ID" INTEGER DEFAULT (NEXT VALUE FOR "PUBLIC"."
    SYSTEM_SEQUENCE_9DCC8EEC_2537_4301_B901_5EC760A8C874") NOT NULL
    NULL_TO_DEFAULT SEQUENCE "PUBLIC"."
    SYSTEM_SEQUENCE_9DCC8EEC_2537_4301_B901_5EC760A8C874",
    "BASE_DIFFICULTY_LEVEL" INTEGER NOT NULL,
    "FAMILY_SWITCH_COSTS" INTEGER NOT NULL,
    "SHOW_LEVEL_DIALOG" BOOLEAN NOT NULL,
    "STEPS_FOR_FACTOR" INTEGER NOT NULL,
```

```
    "UPDATE_LEARN_PATH" BOOLEAN NOT NULL
);
ALTER TABLE "PUBLIC"."USER_PREFERENCES" ADD CONSTRAINT "PUBLIC"."
    CONSTRAINT_2" PRIMARY KEY("ID");
```

```
CREATE MEMORY TABLE "PUBLIC"."USER_TASK_INFO"(
    "ID" INTEGER DEFAULT (NEXT VALUE FOR "PUBLIC"."
    SYSTEM_SEQUENCE_26986BAA_8577_4A1C_9503_B2C17FBD4417") NOT NULL
    NULL_TO_DEFAULT SEQUENCE "PUBLIC"."
    SYSTEM_SEQUENCE_26986BAA_8577_4A1C_9503_B2C17FBD4417",
    "DONE" BOOLEAN NOT NULL,
    "STEP" VARCHAR(255),
    "TASK_FAMILY" VARCHAR(255),
    "TASK_ID" INTEGER NOT NULL,
    "LEARN_PATH_ID" INTEGER
);
ALTER TABLE "PUBLIC"."USER_TASK_INFO" ADD CONSTRAINT "PUBLIC"."
    CONSTRAINT_7" PRIMARY KEY("ID");
```

```
CREATE MEMORY TABLE "PUBLIC"."USER_TASK_INFO_LOG"(
    "ID" INTEGER DEFAULT (NEXT VALUE FOR "PUBLIC"."
    SYSTEM_SEQUENCE_2A7F95E1_8D5C_4BFF_BD77_0A792A50689A") NOT NULL
    NULL_TO_DEFAULT SEQUENCE "PUBLIC"."
    SYSTEM_SEQUENCE_2A7F95E1_8D5C_4BFF_BD77_0A792A50689A",
    "STATEMENT" VARCHAR(255),
    "TIMESTAMP" TIMESTAMP,
    "TYPE" VARCHAR(255),
    "USER_TASK_INFO_ID" INTEGER
);
ALTER TABLE "PUBLIC"."USER_TASK_INFO_LOG" ADD CONSTRAINT "PUBLIC"."
    CONSTRAINT_1" PRIMARY KEY("ID");

ALTER TABLE "PUBLIC"."USER_TASK_INFO" ADD CONSTRAINT "PUBLIC"."
    FK8JAIC68R0VVMTK7RKYKC7WK3I" FOREIGN KEY("LEARN_PATH_ID") REFERENCES
    "PUBLIC"."LEARN_PATH"("ID") NOCHECK;
ALTER TABLE "PUBLIC"."USER_TASK_INFO_LOG" ADD CONSTRAINT "PUBLIC"."
    FKCQ9OEFS0CE61B7CCDXD5DNGV5" FOREIGN KEY("USER_TASK_INFO_ID")
    REFERENCES "PUBLIC"."USER_TASK_INFO"("ID") NOCHECK;
```