Institut für Formale Methoden der Informatik

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Entwicklung eines Frameworks für die Lehre in der Theoretischen Informatik

Marc Doderer

Studiengang:	Informatik
Prüfer/in:	PD Dr. Manfred Kufleitner
Betreuer/in:	PD Dr. Manfred Kufleitner
Beginn am:	08.07.2023
Beendet am:	08.01.2024

Inhaltsverzeichnis

Inhaltsverzeichnis iii

Abbildungsverzeichnis v

Tabellenverzeichnis vi

Programmcodeverzeichnis vii

Abkürzungsverzeichnis viii

1 Einleitung 2

2 Grundlagen 4
2.1 Rust 4
2.1.1 Das Ownership-Modell in Rust 4
2.2 Was ist eine REST-API? 7
2.2.1 Web API 7
2.2.2 Architekturstil REST 8
2.3 Was ist ein JSON Web Token? 10
2.3.1 Funktionsweise von JSON Web Tokens 11
2.4 Was ist eine Datenbank? 12
2.4.1 Grundlagen von relationalen Datenbanken 12
2.4.2 Beschreibung Entity-Relationship-Modell 14
2.4.3 Dokumentenorientierte Datenbank 15
2.5 Open Source Software 16
2.5.1 Actix Web 16
2.5.2 Diesel 18
2.5.3 Liste der verwendete Open Source Softwares 19

3 Benutzerverwaltung 22
3.1 Zugriffsrechte über Rollen 22
3.1.1 Vorteile von Zugriffsrechten über Rollen 22
3.1.2 Nachteile von Zugriffsrechten über Rollen 22
3.2 Einzelne Zugriffsrechte auf Ressourcen 23
3.2.1 Vorteile von Zugriffsrechten einzelner Ressourcen 23
3.2.2 Nachteile von Zugriffsrechten einzelner Ressourcen 23
3.3 Unsere Umsetzung der Benutzerverwaltung 23
3.3.1 Zugriffsarten auf Ressourcen 25

4 Authentifizierung . . . 30
4.1 JWT-Middleware . . . 30
4.2 Permission-Middleware . . . 31
4.2.1 Benutzerverwaltung durch Attribut-ähnliche Makros . . . 31
4.2.2 Benutzerverwaltung über eine Middleware . . . 32
4.2.2.1 Berechtigungstabelle für die Permission-Middleware . . . 32
4.2.2.2 Funktionsweise der Permission-Middleware . . . 34

5 Gruppen . . . 35
5.1 Datenstruktur der Gruppen . . . 35
5.2 Funktionalität von Gruppen . . . 35

6 Aufgaben . . . 36
6.1 Aufgabendokument . . . 36
6.2 Aufgabenpakete . . . 36
6.3 Datenstruktur von Aufgabenpaketen . . . 37
6.4 Anwendung der Aufgaben . . . 38
6.5 Schemas . . . 39
6.6 Auswertung des Lernerfolges . . . 39

7 Proof of Concept . . . 41
7.1 App . . . 41
7.2 Aufgaben erstellen . . . 45
7.2.1 Umsetzung der Erstellung von CYK Aufgaben . . . 45

8 Dokumentation und Tests . . . 47

9 Diskussion . . . 49
9.1 Diskussion zur Authentifizierung . . . 49
9.2 Diskussion zur Benutzerverwaltung . . . 49
9.3 Diskussion zu den Gruppen und Aufgaben . . . 50
9.4 Problem in der Entwicklung . . . 51

10 Fazit . . . 52

Quellenverzeichnis . . . 53

Abbildungsverzeichnis

Abbildung 1 Beispiel Entity-Relationship-Modell 15
Abbildung 2 ERM: Einfache Benutzerverwaltung 24
Abbildung 3 ERM Einfache Rollenverwaltung 25
Abbildung 4 ERM: Zugriffsarten 25
Abbildung 5 ERM: Benutzerverwaltung 27
Abbildung 6 ERM: Aufgaben 38
Abbildung 7 App: Anmeldebildschirm 43
Abbildung 8 App: Hauptbildschirm 43
Abbildung 9 App: Gruppenbildschirm 43
Abbildung 10 App: Mitglieder 43
Abbildung 11 App: Berechtigungen 44
Abbildung 12 App: Statistik über Lösungsversuche 44
Abbildung 13 App: Multiple Choice 44
Abbildung 14 App: CYK Algorithmus 44
Abbildung 15 OpenAPI Dokumentation als interaktive Webseite 47

Tabellenverzeichnis

Tabelle 1 Personen Relation 13
Tabelle 2 Bücher Relation 13
Tabelle 3 SQL Ergebnis 14
Tabelle 4 Antwort der SQL-Anfrage mit JOIN 14
Tabelle 5 Komponententest für den anmelde Endpunkt 48

Programmcodeverzeichnis

Code 1 Beispiel einer Ownership-Übergabe 5
Code 2 Beispiel der Übergabe einer borrowed reference 6
Code 3 Beispiel GET Anfrage und Antwort 9
Code 4 Beispiel POST Anfrage und Antwort 9
Code 5 Beispiel DELETE Anfrage und Antwort 10
Code 6 Beispiel HEAD Anfrage und Antwort 10
Code 7 Beispiel PUT Anfrage und Antwort 10
Code 8 JSON Web Token: Header 11
Code 9 JSON Web Token: Header 11
Code 10 Beispiel eines JSON Web Tokens 11
Code 11 Beispiel einer einfachen SQL Anfrage 14
Code 12 Beispiel einer SQL Anfrage mit JOIN 14
Code 13 Beispiel eines Dokumentes in der dokumentenorientierten Datenbank 16
Code 14 Actix Web 18
Code 15 Beispiel einer SQL Anfrage mit Diesel 19
Code 16 TRIGGER zum Validieren von Nutzer_Zugriffsart-Einträgen 28
Code 17 DELETE-TRIGGER zum Löschen von Nutzer_Zugriffsart-Einträgen 28
Code 18 UPDATE-TRIGGER zum Löschen von Nutzer_Zugriffsart-Einträgen 29
Code 19 INSERT-TRIGGER zum Löschen von Nutzer_Zugriffsart-Einträgen . 29
Code 20 Beispielverwendung der JWT-Middleware 31
Code 21 Beispiel der Benutzerverwaltung mit Makros 32
Code 22 Auschnitt permission.toml 33
Code 23 Beispiel eines Aufgabendokumentes 36
Code 24 JSON Schema 39
Code 25 Nutzerstatistik über Lernerfolg innerhalb eines Aufgabenpaketes . . . 40
Code 26 Programmcode: CYK-Algorithmus 46

Abkürzungsverzeichnis

GC	Garbage Collector
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
JSON	Java Script Object Notation
REST	Representational State Transfer
gRPC	gRPC Remote Procedure Calls
JWT	JSON Web Token
DBMS	Datenbankverwaltungssystemen
ID	Identifikationsnummer
SQL	Structured Query Language
ERM	Entity-Relationship-Modell
OSS	Open Source Software
OOP	Objekt-Orientierten Programmiersprachen
ORM	Objekt Relational Mapping
CORS	Cross-Origin Resource Sharing
UUID	Universally Unique Identifiers
TOML	Tom's Obvious, Minimal Language
PoC	Proof of Concept
CNF	Chomsky-Normalform
CYK	Cocke-Younger-Kasami

Kurzfassung

In dieser Arbeit präsentieren wir ein von uns entwickeltes Framework für eine Lernplattform. Das Framework stellt über eine REST-Api eine Vielzahl von Endpunkten zu Verfügung, die die Verwaltung von Aufgaben, Gruppen und Benutzern ermöglichen. Außerdem kann durch die von unserem Framework bereitgestellten Informationen ein Überblick über die Lernfortschritte der Nutzer abgerufen werden. Durch eine Proof of Concept App demonstrieren wir eine praxisnahe Anwendung unseres Frameworks. Insbesondere zeigen wir, wie das Framework effektiv Multiple-Choice-Aufgaben implementiert und Aufgaben zur manuellen Bearbeitung des CYK-Algorithmus ermöglicht. Hierbei wird verdeutlicht, wie das Framework zur Lehre von theoretischer Informatik verwendet werden kann. Durch die transparente Darstellung noch ausstehender Funktionen schaffen wir Anreize für mögliche Weiterentwicklungen.

Abstract

In this paper, we present a framework developed by us for an e-learning platform. The framework provides a variety of endpoints via a REST API, enabling the management of tasks, groups, and users. Additionally, an overview of users' learning progress can be obtained through the information provided by our framework. Through a Proof of Concept app, we demonstrate a practical application of our framework. In particular, we showcase how the framework effectively implements multiple-choice tasks and facilitates tasks for the manual execution of the CYK algorithm. This illustrates how the framework can be utilized for teaching theoretical computer science. By transparently presenting pending features, we provide incentives for potential future developments.

1 Einleitung

Lernplattformen spielen eine entscheidende Rolle in der effizienten Wissensvermittlung [1]. Besonders in der theoretischen Informatik können durch Lernplattformen Algorithmen interaktiv unterrichtet werden. Die Entscheidung, eine maßgeschneiderte Lernplattform zu entwickeln, anstatt auf bereits bestehende zurückzugreifen, kann für Bildungseinrichtungen wie Universitäten von großem Vorteil sein. Eigenentwickelte Plattformen ermöglichen nicht nur eine flexible Anpassung an die spezifischen Anforderungen und Lehrziele einer Institution, sondern fördern auch die bewusste Integration neuer Lehr- und Lernmethoden.

Im Mittelpunkt dieser Bachelor-Arbeit steht das Ziel, ein Framework für eine solche Lernplattform zu entwickeln. Besonderes Augenmerk liegt dabei auf der Strukturierung von Benutzern, Aufgaben und Lerngruppen. Um dies zu erreichen, haben wir eine REST-API entworfen, die über spezifische Endpunkte die Verwaltung von Gruppen, Aufgaben und Nutzern ermöglicht. Für eine deutliche Veranschaulichung der Funktionalitäten des Frameworks haben wir nicht nur an dessen Entwicklung gearbeitet, sondern auch eine Proof-of-Concept-App entwickelt. Diese App dient dazu, die Funktionsweise der Lernplattform zu demonstrieren. In diesem Artikel geben wir nicht nur Einblicke in den Aufbau des Frameworks, sondern gewähren auch einen Blick auf die Proof-of-Concept-App.

Der vorliegende Artikel richtet sich sowohl an erfahrene Entwickler als auch an Personen ohne Vorwissen in der Softwareentwicklung, die Interesse am Thema haben. Um eine breite Leserschaft anzusprechen, werden im ersten Abschnitt die grundlegenden Strukturen des Frameworks erläutert, begleitet von Definitionen der verwendeten Fachbegriffe. Dies ermöglicht einen leicht verständlichen Einstieg in die Thematik.

Nach der Einführung in die Grundlagen des Frameworks folgen detaillierte Erläuterung zur Benutzerverwaltung, wobei besonderes Augenmerk auf die Datenstruktur gelegt wird. Die Umsetzung dieser Benutzerverwaltung in der REST-API wird im Abschnitt *Kapitel 4 - Authentifizierung* ausführlich behandelt, inklusive Anweisungen zur Authentifizierung von Nutzenden.

In den folgenden Abschnitten *Kapitel 5 - Gruppen* und *Kapitel 6 - Aufgaben* beschreiben wir, wie die Aufgaben und Lerngruppen in unserem Framework strukturiert sind und angewandt werden.

Die Umsetzung der Proof-of-Concept App wird daraufhin im Abschnitt *Kapitel 7 - Proof of Concept* näher erläutert.

Das Framework ist unter der MIT-Lizenz veröffentlicht und auf GitHub verfügbar [2]. Wir haben deshalb Dokumentationen und Tests geschrieben. Auf diese gehen wir in dem *Kapitel 8 - Dokumentation und Tests* näher ein.

In dem *Kapitel 9 - Diskussion* interpretieren wir die Funktionalitäten unseres Frameworks und geben einen Einblick in die Entwicklungsphase. Zusätzlich werden Funktionen diskutiert, die bisher nicht umgesetzt wurden, aber zukünftig in der Weiterentwicklungen des Frameworks einfließen können. Lesende können somit nicht nur die Funktionsweise des entwickelten Frameworks verstehen, sondern auch die Herausforderungen und Ansätze der Entwicklung nachvollziehen.

Abschließend wird ein umfassendes Fazit gezogen, das die wichtigsten implementierten und noch fehlenden Funktionen des Frameworks auflistet.

2 Grundlagen

Für die Entwicklung des Frameworkes wurde die Sprache Rust [3] verwendet. Rust ist eine im Vergleich zu anderen Programmiersprachen, sehr neue Sprache, da sie erst 2015 veröffentlicht wurde [4]. Die REST-API, die von dem Projekt bereitgestellt wird, wurde mithilfe des Frameworks *Actix-Web* [5] umgesetzt.

2.1 Rust

Rust wurde von einem Mozilla-Mitarbeiter entwickelt und folgend mithilfe von Mozilla bekannt gemacht [6]. Die Programmiersprache zeichnet sich dabei besonders durch ihr *Ownership-Model* aus. Zusätzlich ist Rust so entworfen das „nebenläufige Behandlungen“ und „Parallelverarbeitungen“ von Grund auf unterstützt werden [7]. Auch mit unter wegen dieser Eigenschaften wurde Rust als Sprache für unser Framework verwendet.

2.1.1 Das Ownership-Modell in Rust

Rust benötigt keinen Garbage Collector (GC) ([6], [7]). Ein GC ist eine Software, die von Sprachen verwendet wird, um den Speicher automatisch zu Verwalten. Sprachen mit GC sind durch die wegfallende Speicherverwaltung, meist einfacher zu erlernen, als Sprachen ohne GC. Dabei übernimmt sie unter anderem das Aufräumen von ungenutzten Objekten im Speicher. Über die Beseitigung dieser Objekte müssen sich die Entwickler deshalb keine Gedanken mehr machen. Hierdurch entstehen seltener Speicherfehler. Speicherfehler treten zum Beispiel auf, wenn Entwickler Objekte aus dem Speicher löschen, jedoch Referenzen noch auf diese zeigen. Diese Referenzen, die auf einen ungültigen Wert zeigen, nennt man *Hängende Zeiger* [8]. Dabei kann es von Sicherheitsrisikos bis zu Programmabstürzen kommen. GC laufen jedoch parallel zum Programm, wodurch Rechenleistung anfällt.

Auch andere Programmiersprachen, beispielsweise C++ verwenden standardmäßig, keinen GC. Im Gegensatz zu diesen anderen Programmsprachen ist Rust jedoch so aufgebaut, dass weniger solcher Speicherfehler erzeugt werden können. Der Grund für diese Sicherheit ist das *Ownership-Modell*. Objekte in Rust können immer nur einer Variable angehören. Es kann zwar die Angehörigkeit, von einer Variable zu einer anderen, übertragen werden, jedoch ist diese zu jedem Zeitpunkt nur einer angehörig. Im *Code 1 - Beispiel einer Ownership-Übergabe* wird verdeutlicht, welche Auswirkungen dieses *Ownership-Modell* hat. Wie in den Kommentaren beschrieben, kann die *variable_one* nach der Übergabe an *variable_two* nicht mehr verwendet werden. Neben dem Übertragen der Ownership-Rechten an einem Objekt, gib es in Rust, die Möglichkeit auch eine *borrowed reference* zu übergeben. Diese *borrowed references* können nur im *lexical scope* [9] existieren. Der

Gültigkeitsbereich (scope) einer Variable gibt den Bereich vor, in dem auf diese zugegriffen werden kann. Lexikalisch (lexical) bedeutet in diesem Zusammenhang, dass dieser Zugriffsbereich durch den Quellcode und nicht durch die Laufzeit vorgegeben wird. Durch diese Eigenschaft kann Rust sicherstellen, das bei Ableben der original Referenz keine weitere Variable auf das Objekt zeigt. Referenzen werden bei dem Verlassen eines *scopes* [9], automatisch aus dem Speicher entfernt [7]. Im *Code 2 - Beispiel der Übergabe einer borrowed reference* ist ein Beispiel ersichtlich, wie eine solche *borrowed reference* aussieht und wie diese funktioniert.
Durch diesen Aufbau hat Rust den Performance-Vorteil von Programmiersprachen ohne GC und trotzdem eine ausreichende Speichersicherheit durch das *Ownership-Modell* [7].

```rust
fn main() {
    // Speicher String in der Variable "variable_one".
    let variable_one = String::from("Hello World");

    // Übergebe "variable_one" in "variable_two".
    // (Ownership-Übergabe)
    let variable_two = variable_one;

    // "variable_one" kann jetzt nicht mehr verwendet werden,
    // da der Eigentum an die "variable_two" weitergegeben wurde.
    // Folgender Befehl würde zum Beispiel in einem Compilerfehler enden.
    // println!("{}", variable_one);

    // "variable_two" kann normal verwendet werden.
    println!("{}", variable_two);
}
```

Code 1 Beispiel einer Ownership-Übergabe

```rust
fn main() {
    // Speicher String in der Variable "variable_one".
    let variable_one = String::from("Hello World");

    // Übergebe einer borrowed reference
    // von "variable_one" an "variable_two".
    let variable_two = &variable_one;

    // "variable_one" kann weiterhin verwendet werden.
    println!("{}", variable_one);

    // "variable_two" kann normal verwendet werden.
    println!("{}", variable_two);

    // "variable_two" ist eine borrowed reference,
    // kann demnach also auch nur als
    // borrowed reference weitergegeben werden
    // calc_length_one(variable_two) erzeugt ein Compilerfehler
    println!("{}", calc_length_two(variable_two));
}

fn calc_length_one(s: String) -> usize {
    s.len()
}

fn calc_length_two(s: &String) -> usize {
    s.len()
}
```

Code 2 Beispiel der Übergabe einer borrowed reference

2.2 Was ist eine REST-API?

Eine REST-API wird im Buch „REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces“ von Masse [10] folgend beschrieben: „*Eine REST API ist eine Art von Web Server, der Nutzer ermöglicht entweder selbst gesteuerten oder automatisierten Zugriff auf Ressourcen, welche die Daten und Funktionen eines Systems modellieren.*“

2.2.1 Web API

Ein Application Programming Interface (API) ist eine Schnittstelle die es zwei unabhängigen Systemen ermöglicht miteinander zu kommunizieren [11]. Im Fall unseres Frameworks wurde eine auf Hypertext Transfer Protocol (HTTP) basierende Web API erstellt, die mittels Java Script Object Notation (JSON) die Kommunikation zwischen einem Client - hier in unserer Anwendung die „Proof of Concept“ App - und der Datenbank, über das Internet ermöglicht. Mittels HTTP können Anfragen an die Web API gesendet werden. Dabei sind solche Anfragen wie nachfolgend beschrieben aufgebaut:

1. Einer Uniform Resource Locator (URL):
   Bei nachfolgender Beispiel-URL „https://www.google.com/search?client=firefox-b-d&q=api“, gibt „/search“ den Pfad zu der Ressource vor, auf die zugegriffen werden soll. Über weitere Attribute, die nach einem Fragezeichen aufgelistet werden, können der URL noch zusätzliche Informationen angefügt werden. Diese Attribute sind jeweils mit einem & Zeichen voneinander getrennt und werden *Query-String* genannt. In der angegebenen URL wurden die Informationen „client=firefox-b-d“ so wie die Suchanfrage „q=api“ als *Query-String* übergeben [12].

2. Einer HTTP-Methode:
   Laut dem Buch „RESTful web services“ von Richardson und Ruby [12] sind die 5 meist verbreitetsten HTTP Methoden GET, HEAD, PUT, DELETE und POST. Die GET Methode kann verwendet werden, um Ressourcen anzufragen. Beispielsweise nutzt ein Browser standardmäßig diese HTTP Methode, um Webseiten von Web Servern anzeigen zu können. Im *Code 3 - Beispiel GET Anfrage und Antwort* ist ein Beispiel einer solchen Anfrage und Antwort ersichtlich. Dabei werden Informationen für den Nutzer mit der Identifikationsnummer (ID): 123 angefragt. HEAD kann verwendet werden, um Informationen über eine Ressource anzufragen. Diese Informationen nennen sich „header“ und beinhalten zum Beispiel das Format einer Ressource. Im Beispiel aus dem *Code 6 - Beispiel HEAD Anfrage und Antwort* wird gezeigt, dass die Header Informationen der GET Anfrage aus *Code 3 - Beispiel GET Anfrage und Antwort* als Antwort gegeben wird. Mittels PUT kann übermittelt werden, wenn die im Pfad hinterlegte Ressource ersetzt oder erstellt werden

soll. Ein Beispiel ist im *Code 7 - Beispiel PUT Anfrage und Antwort* ersichtlich. DELETE löscht die Ressource und mit POST kann eine Ressource erzeugt werden, die folgend unter einer neuen URL aufzufinden ist. Im *Code 5 - Beispiel DELETE Anfrage und Antwort* und *Code 4 - Beispiel POST Anfrage und Antwort* sind erfolgreiche Beispiele von POST und DELETE Anfragen und entsprechender Antworten abgebildet.

3. Einem HTTP-Body:
   Bei schreibenden Methoden, wie PUT können zusätzlich Informationen mitangegeben werden. Diese werden dann im sogenannten *HTTP Body* hinterlegt. In dem Beispiel aus *Code 7 - Beispiel PUT Anfrage und Antwort* ist der *HTTP Body* in den Zeilen 5 bis 7 ersichtlich.

4. Einem Header:
   In dem Header sind Informationen über den Anfragenden so wie die Anfrage selbst. Ein Beispiel solcher Information wäre das gewählte Format (JSON, XML) des *HTTP Body*. In dem Beispiel aus *Code 7 - Beispiel PUT Anfrage und Antwort* ist der *HTTP Body* in den Zeilen 2 und 3 ersichtlich.

2.2.2 Architekturstil REST

Representational State Transfer (REST) steht für einen Architekturstil, den man beim Erstellen einer Web API einhält. Eine Art Design Vorlage, die es anderen Entwicklern einfacher macht die Web API zu verstehen und zu verwenden. REST setzt laut IBM [13] folgende REST Designprinzipien voraus:

1. Alle API Anfragen, die auf ein und dieselbe Ressource gerichtet sind, besitzen dieselbe URL.

2. Client und Server sollen jeweils vollständig und unabhängig voneinander existieren.

3. Die API ist zustandslos, speichert demnach keine Server-Sitzungen. Jede Anfrage vom Client wird isoliert behandelt, ohne dass der Server, zwischen den Anfragen, spezifische Daten über den Zustand des Clients speichert.

4. Wenn möglich, können Ressourcen auf dem Server oder Client zwischengespeichert werden. Informationen über eine mögliche Zwischenspeicherung muss in der Antwort mitgegeben werden.

5. Kommunikation mit der REST-API soll auch möglich sein, wenn diese über eine zwischengeschobene dritte Instanz geführt wird.

REST ist ein Architekturstil, bei dem die Ressourcen selbst im Vordergrund stehen. Dies ist im Vergleich zu Alternativen, wie zum Beispiel gRPC Remote Procedure Calls (gRPC) [14], anders. Dort stehen die Funktionen im Vordergrund. Für das Framework haben wir entschieden eine REST-API zu entwickeln. Diese Entscheidung basierte auf mehreren Überlegungen:

Erstens vereinfacht die Unabhängigkeit von Client und Server die Arbeit für nachfolgende Entwickler, die am Projekt weiterarbeiten. REST fördert eine klare Trennung zwischen Client und Server, was die Wartbarkeit und Erweiterbarkeit des Projekts begünstigt. Zweitens ist REST ein weit verbreiteter API-Stil, der von vielen Entwicklern genutzt wird. Dies erleichtert anderen Entwicklern die Arbeit mit unserer API, da sie bereits mit dem gängigen REST-Paradigma vertraut sind. Drittens ermöglicht die Verwendung von REST eine einfache Integration des Frameworks mit anderen Services. Zum Beispiel könnte das Projekt in einer Microservices-Architektur [15] verwendet werden. Die REST-API könnte nahtlos mit anderen APIs kombiniert werden, beispielsweise mit einer, die sich auf das sichere Lösen von Aufgaben während Prüfungen konzentriert.

Diese Gründe unterstreichen die Vorteile von REST in Bezug auf Flexibilität, Verständlichkeit und Integrationsfähigkeit für die Entwicklung unseres Frameworks.

```
GET /users/123 HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 Ok
Content-Type: application/json

{
    "id": "123",
    "name": "Max Mustermann",
}
```

Code 3 Beispiel GET Anfrage (links) und Antwort (rechts)

```
POST /users HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "Otto
    ↪ Normalverbraucher"
}
```

```
HTTP/1.1 201 Created
Location: /users/124
Content-Type: application/json

{
    "id": "124",
    "name": "Otto
    ↪ Normalverbraucher",
}
```

Code 4 Beispiel POST Anfrage (links) und Antwort (rechts)

```
DELETE /users/123 HTTP/1.1
Host: example.com
```

```
HTTP/1.1 204 No Content
```

Code 5 Beispiel DELETE Anfrage (links) und Antwort (rechts)

```
HEAD /users/123 HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 Ok
Location: /users/124
Content-Type: application/json
```

Code 6 Beispiel HEAD Anfrage (links) und Antwort (rechts)

```
HEAD /users/124 HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "Lieschen Müller"
}
```

```
HTTP/1.1 200 Ok
Content-Type: application/json

{
    "id": 124,
    "name": "Lieschen Müller",
}
```

Code 7 Beispiel PUT Anfrage (links) und Antwort (rechts)

2.3 Was ist ein JSON Web Token?

Die Authentifizierung über einen JSON Web Token (JWT) ist ein einfacher Weg um Nutzende am Server zu legitimieren. Bei der Anmeldung an der REST-API stellt diese dem Anfragenden einen *Access-Token* bereit. Dieser kann verwendet werden, um sich bei weiteren Anfragen an der API auszuweisen. Die Verwendung von JWT bietet sich für REST-APIs an, da für diese keine Sitzungen gespeichert werden müssen. Der *Access-Token* ist mittels JSON nach RFC 7519 [16] genormt. Im Artikel „Introduction to JSON Web Tokens“ von Auth0 [17] wird beschrieben, dass ein JWT aus einem Header, einem Payload und einer Signatur besteht.

1. *Header*
   Ein Header besteht typischerweise aus der Angabe des Signatur-Algorithmus und dem Token Typ. Dieser Token Typ hat den Wert „JWT“. Der Header wird mit Base64Url kodiert und steht am Anfang des Tokens.

2. *Payload*
   Ein Payload beinhaltet sogenannte Claims. Claims sind Informationen über den Nutzenden oder anderen Daten. Wie auch der Header wird der Paylod mit Base64Url kodiert und am Header mit einem Punkt getrennt angefügt.

```
{
    "alg": "HS256",
    "typ": "JWT",
}
```

Code 8 JSON Web Token: Header

```
{
    "sub": "1234567890",
    "name": "John Doe",
    "admin": true
}
```

Code 9 JSON Web Token: Payload

3. *Signatur*
   Die Signatur wird mittels des im Header angegebenen Algorithmus, einem privaten Schlüssel und dem Payload zusammen mit dem Header errechnet. Das Ergebnis wird, wie auch schon die anderen beiden Komponenten, mittels Base64Url kodiert und nach einem Punkt an den Token angefügt.

   Ein *privater Schlüssel* ist eine geheime kryptografische Zeichenfolge, die für die Entschlüsselung von verschlüsselten Daten oder die digitale Signierung von Informationen verwendet wird und streng vertraulich gehalten werden muss.

Ein fertiger JWT könnte wie folgt aussehen:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
vS_i5VRtYot760a9J2HAFP7JyHCk2c31I0E8kjQHctA
```

Code 10 Beispiel eines JSON Web Tokens

2.3.1 Funktionsweise von JSON Web Tokens

Die Kodierung des Payloads und des Headers verhindert nicht, das Dritte den Inhalt lesen können. Bei der Validierung des *Access-Tokens* wird nur abgeglichen, dass die Signatur des Payloads und des Headers der beigefügten Signatur des Tokens gleicht. So kann die REST-API dem Inhalt vertrauen. Diese Sicherheit erhält die REST-API über die Signatur.
Digitale Signaturen, benutzen eine *Hashfunktion*, die zwei Werte miteinander kombiniert und ein Ergebnis ausgibt. Diese Hashfunktion sorgt dafür, dass eine *Kollision*, also dass zu zwei unterschiedlichen Eingabewerten das gleiche Ergebnis errechnet wird, nahezu nicht

auftreten kann. Über die Signierung des Payloads und des Headers, kann dies nun mit der angehängten Signatur abgeglichen werden. Stimmen beide überein ist gewährleistet, dass der Inhalt nicht verändert wurde. [18]

2.4 Was ist eine Datenbank?

Um Personen, Gruppen und Aufgaben zu Verwalten, brauchen wir die Möglichkeit Daten zu speichern und wieder aufzurufen. In der Software-Entwicklung werden dafür üblicherweise Datenbanken verwendet, da sie diese Aufgaben effizient übernehmen. Datenbanken werden in den meisten Fällen von Datenbankverwaltungssystemen (DBMS) verwaltet. [19]. Ein DBMS gibt dabei in der Regel vor, wie die Daten in der Datenbank strukturiert und verwaltet werden.
Im Laufe der Zeit haben sich mehrere Datenbankentypen und DBMS durchgesetzt. ([20], [21], [22]) Für das Framework haben wir eine relationale Datenbank mit PostgreSQL als DBMS und eine dokumentenorientierte Datenbank mit MongoDB als DBMS verwendet. Wir verwenden überwiegend die relationale Datenbank, weshalb wir zunächst auf diese näher eingehen.

2.4.1 Grundlagen von relationalen Datenbanken

Relationale Datenbanken haben eine tabellenartige Struktur. Diese Tabellen werden als Relationen bezeichnet.[23] Eine relationale Datenbank besteht also aus mehreren Relationen (Tabellen). Jede Zeile innerhalb einer Relation entspricht einem Datensatz. Dieser besteht aus mehreren *Attributwerten*, die durch die Spalten definiert sind. Datensätze werden in relationalen Datenbanken als Entitäten bezeichnet. Die Spalten der Tabelle werden als Attribut bezeichnet und geben jeweils den Datentyp, Namen und Eigenschaften der Attributwerte - also des Zellinhaltes einer Spalte - vor. In der Beispiels-Relation *Tabelle 1 - Personen Relation* sind die Entitäten (ID: 1, Vorname: Erika, Nachname: Mustermann), (ID: 2, Vorname: Otto, Nachname: Normalverbraucher) und (ID: 3, Vorname: Max, Nachname: Mustermann) eingetragen. Dabei stehen die Attributnamen in der Spaltenüberschrift der Tabelle. Relationen besitzen meist ein Attribut, das jeden Attributwert nur einmal vergeben kann und so einmalig für jede Entität ist. In unserem Beispiel wäre dies, das Attribut ID. Dieses Attribut wird *Primärschlüssel* genannt und kann zum Identifizieren von Entitäten verwendet werden. In unseren Beispielen sind diese, mit einem Unterstrich gekennzeichnet. Relationen welche keinen Primärschlüssel haben, müssen einen *zusammengesetzen Primärschlüssel* besitzen. Dieser besteht aus mehreren Attributen und stellt sicher, dass jede Kombination von Attributwerten nur einmal existieren kann. Entitäten können somit über die Kombination der Attributwerte dieses *zusammengesetzen Primärschlüssels* eindeutig zugeordnet werden.

Die Primärschlüssel einer Relation können von einer anderen Relationen als *Fremdschlüssel* verwendet werden. Ein Fremdschlüssel ist somit ein Attribut das auf ein Primärschlüssel, oder ein anderes einmaliges Attribut, einer anderen Entitäten verweist. In der Buchrelation aus *Tabelle 2 - Bücher Relation* ist der Fremdschlüssel Autor zusehen. Dieser zeigt auf den Primärschlüssel einer Personen-Entität.

Durch Fremdschlüssel können Beziehungen von Relationen zueinander beschrieben werden.

- In einer eins-zu-eins (*1:1*) Beziehung wird das Fremdschlüsselattribut einmalig definiert. Durch diese Eigenschaft können keine zwei Entitäten einer Relation auf die gleiche Entität einer anderen Relation verlinken. So kann jede Entität dieser Relation, auf nur eine Entität zeigen und auf jede Entität der anderen Relation, kann nur einmal gezeigt werden. Wenn in der Buchrelation sichergestellt werden soll, das jede Person nur ein Buch geschrieben hat, muss das Fremdschlüsselattribut Autor die Eigenschaft einmalig vorgeben. Dies könnte beispielsweise bei einem Schreibwettbewerb vorausgesetzt werden. Somit wurde jedes Buch von einem Autor geschrieben und jeder Autor hat nur ein Buch geschrieben.

- In einer eins-zu-viele (*1:n*) Beziehung wird diese Eindeutigkeit des Fremdschlüsselattributes nicht vorausgesetzt. Ein solches Beispiel ist in der Beziehung der *Tabelle 1 - Personen Relation* und der *Tabelle 2 - Bücher Relation* ersichtlich. Jedes Buch wurde nur von einem Autor geschrieben, jedoch kann ein Autor mehrere Bücher geschrieben haben.

- In einer viele-zu-viele (*m:n*) Beziehung muss eine dritte Relation erschaffen werden, das als Bindeglied beider Relationen agiert. Diese dritte Relation besteht aus Entitäten, welche Fremdschlüssel beider Relationen besitzt.

ID	Vorname	Nachname
1	Erika	Mustermann
2	Otto	Normalverbraucher
3	Max	Mustermann

Tabelle 1 Personen Relation

ISBN	Name	Autor
1111	Buch 1	2
2232	Buch 2	1
3334	Buch 3	2
4534	Buch 4	3

Tabelle 2 Bücher Relation

Die Manipulation oder Abfrage dieser Relationen wird durch die relationale Algebra [24] beschrieben. Häufig wird dabei die Datenbanksprache Structured Query Language (SQL) verwendet. Um beispielsweise Informationen zu erhalten, welche Bücher der Person mit der ID = 2, zugeordnet sind, kann folgende Anfrage verwendet werden:

```
1 SELECT ISBN, Name
2 FROM Bücher
3 WHERE Autor = 2;
```

Code 11 Beispiel einer einfachen SQL Anfrage

ISBN	Name
2232	Buch 1
3334	Buch 3

Tabelle 3 SQL Ergebnis

Das Ergebnis dieser Anfrage ist in der *Tabelle 3 - SQL Ergebnis* ersichtlich.

Über solche Anfragen können auch mehrere Relationen miteinander verbunden werden. Wollen wir wissen, welche Bücher einem Autor mit dem Nachnamen „Mustermann“ zugeordnet sind, kann folgende Anfrage verwendet werden:

```
1 SELECT ISBN, Name, ID, Vorname, Nachname
2 FROM Bücher b
3 INNER JOIN Personen p ON p.ID = b.Autor
4 WHERE Nachname = 'Mustermann';
```

Code 12 Beispiel einer SQL Anfrage mit JOI

ISBN	Name	ID	Vorname	Nachname
1111	Buch 2	1	Erika	Mustermann
4534	Buch 4	3	Max	Mustermann

Tabelle 4 Antwort der SQL-Anfrage mit JOIN

Als Antwort ergibt sich die *Tabelle 4 - Antwort der SQL-Anfrage mit JOIN*, die sowohl Informationen über Bücher als auch Personen enthält.

2.4.2 Beschreibung Entity-Relationship-Modell

Um die Beziehung von Relationen zu verbildlichen, verwenden wir in diesem Artikel das Entity-Relationship-Modell (ERM) [25]. Das Modell aus der *Abbildung 1 - Beispiel Entity-Relationship-Modell* beschreibt die Beziehung zwischen den Relationen in der *Tabelle 1 - Personen Relation* und der *Tabelle 2 - Bücher Relation*. Wichtig ist, dass Relationen aus der relationalen Datenbank im ERM als Entitäten beschrieben sind. Entitäten, also Datensätze, aus der relationalen Datenbank sind demnach nicht mit den Entitäten aus dem ERM gleichzusetzen.
Aus dem Modell kann die Beziehung der Relationen sowie deren Attribute gelesen werden. Ausgeschrieben wird das Modell folgend gelesen: *Eine Person kann Autor von mehreren Büchern sein. Ein Buch, hat genau einen Autor*

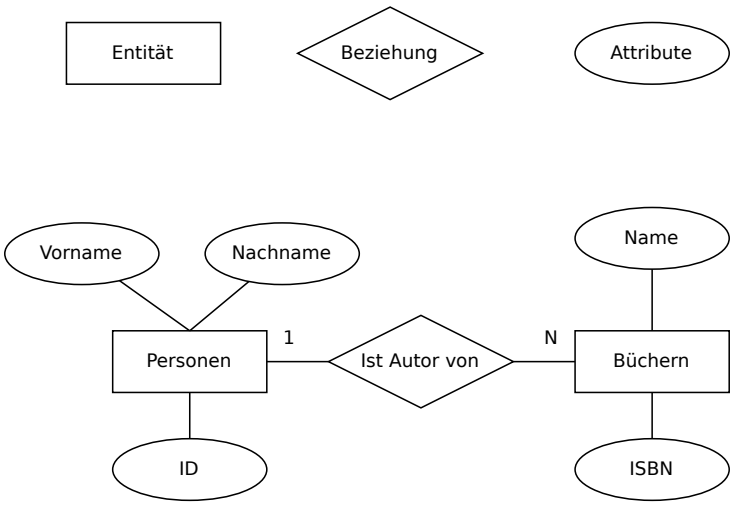

Abbildung 1 Beispiel Entity-Relationship-Modell

2.4.3 Dokumentenorientierte Datenbank

Relationale Datenbanken sind fest strukturiert, da sie die Attribute der Datensätze, also der Entitäten, vorgeben. Wenn wir uns das Speichern von Aufgaben anschauen, existiert bei der relationalen Datenbank das Problem, dass jeder Aufgabentyp unterschiedliche Attribute braucht. So braucht es um eine Multiple-Choice-Aufgabe zu speichern, eine Relation, die eine Antwort und eine Aufgabenliste besitzt. Eine Graph-Aufgabe, bräuchte wiederum eine andere Relation, mit der ein Graph in der Relationalen Datenbank abgespeichert werden kann. So müsste die Relationale Datenbank also für jeden zukünftigen Aufgabentyp erweitert werden. Um dies zu vermeiden, verwenden wir in dem Framework neben der relationalen Datenbank zusätzlich eine dokumentenorientierte Datenbank. Diese speichert die Daten nicht als Entitäten in Relationen, sondern als Dokumente. Diese Dokumente sind in unserem Fall in dem Format JSON und unterliegen somit keiner vorgegebenen Datenstruktur. Ein solches Dokument kann in dem *Code 13 - Beispiel eines Dokumentes in der dokumentenorientierten Datenbank* gesehen werden. Dokumentenorientierte Datenbanken können, im Gegensatz zur relationalen Datenbank, jedoch keine Geschäftslogik auf Ebene der Datenbank umsetzen

```
{
    "_id": 1,
    "Vorname": "Erika Müller",
    "Nachname": "Mustermann",
    "Geschriebene Bücher": [
        {
            "ISBN": 2232,
            "Name": "Buch 2"
        }
    ]
}
```

Code 13 Beispiel eines Dokumentes in der dokumentenorientierten Datenbank

2.5 Open Source Software

In der Software Industrie ist es gängig, sogenannte Open Source Software (OSS) zu verwenden und zu schreiben. Damit ist Software gemeint, die unter *Open source* Lizenzen stehen, wodurch diese von Dritten verwendet oder sogar modifiziert werden dürfen. Es bietet sich an, bei komplexen oder zeitintensiver Problemstellungen, OSS Projekte anderer Personen oder Gruppen zu verwenden. Dabei gibt es mehrere Arten solcher Lizenzen. Zwei gängige Lizenzen, unter welchen häufig OSS lizenziert sind, sind die *MIT License* [26] und die *Apache License 2.0* [27]. Beide Lizenzen setzen eine Lizenzen-Notiz des ursprünglichen Projektes voraus, können dann aber beliebig verwendet und modifiziert werden.
Neben den Vorteilen von OSS, besitzen diese jedoch auch einen Mangel. Die Entwickler von Code unter freier Lizenz halten sich nicht immer an eine wünschenswerte Korrektheit des Codes. Häufig werden diese von einzelnen Entwicklern oder kleiner Entwickler Gruppen frei umgesetzt. Sollten die Entwickler sich später dafür entscheiden die Software nicht mehr weiterzuentwickeln, hat man bei Verwendung des betroffenen Codes im ungünstigsten Fall ein nicht mehr funktionierendes Paket in seinem Programm und muss dies umschreiben. Aufgrund dieser Limitierung haben wir uns dazu entschieden nur wenige solcher Bibliotheken, Frameworks oder Pakete mit ins Projekt einzubinden.

2.5.1 Actix Web

Ein Framework, das in unserem Projekt verwendet wird, ist *Actix Web*. Nach Angabe der Webseite des Actix Teams, ist Actix Web ein leistungsstarkes und flexibles Web-Framework für Rust [5]. Dabei unterstützt es unter anderem bei folgende Aufgaben:

1. Validierung der Datenstrukturen von ein- und ausgehenden Objekten.

2. Verwendung von Middleware, welche beispielsweise die Authentifizierung übernehmen können.

3. Erstellung eines HTTP-Servers.

4. Ein komponentenbasiertes Routing-System, mit welchem die Endpunkte für Ressourcen definiert werden können.

Im Codeabschnitt aus dem *Code 14 - Actix Web*, welches aus der Homepage von *Actix Web* übernommen wurde, wird erläutert wie eine einfache Web-API erstellt werden kann. In der *main* Funktion des Beispiels, wird ein *HTTP-Server* gehostet, der unter *"localhost:8080"* erreicht werden kann. Das Routing innerhalb des *HTTP-Servers*, ist dabei baumartig strukturiert. In den Zeilen 17 und 18, werden über den *.service()* Aufruf, dem Baum (*HTTP-Server*) zwei Blätter (*API Endpunkte*) hinzugefügt. Durch die Codeabschnitte

```
#[get("/")]
```

und

```
#[get("/{name}")]
```

werden jeweils die *HTTP-Methoden* und der Lokale Pfad dieser Endpunkte definiert. Der Routing-Baum beinhaltet auch Möglichkeiten von Abzweigungen, diese können in der ausgiebigen Dokumentation [28] von *Aktix Web* nachgelesen werden.

```rust
use actix_web::{get, web, App, HttpServer, Responder};

#[get("/")] // Endpunkt unter dem Pfad "/"
async fn index() -> impl Responder {
    "Hello, World!"
}

#[get("/{name}")] // Endpunkt mit der PathVariable "name"
async fn hello(name: web::Path<String>) -> impl Responder {
    format!("Hello {}!", &name)
}

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(||
            App::new()
            .service(index) // Hinzufügen des "index" endpunktes
            .service(hello) // Hinzugügen des "hello" endpunktes
        )
        .bind(("127.0.0.1", 8080))? // Hosten eines HTTP-Servers unter
↪   localhost:8080
        .run()
        .await
}
```

Code 14 Beispiel Actix Web

Neben *Actix Web* gibt es auch weitere Alternativen, die unterstützend bei der Entwicklung von REST-APIs agieren. Mögliche Alternativen sind *Rocket* [29], *axum* und *warp*. Die Entscheidung der Wahl von *Actix Web* wurde unter folgenden Punkten getroffen:

1. *Actix Web* wird anders als *Rocket* von einem Team aus mehreren Personen entwickelt. Dadurch existiert eine geringere Wahrscheinlichkeit, dass die Entwicklung aufgrund von Vernachlässigung stoppt.

2. Die Dokumentation ist ausgiebiger als die von *warp* und *axum* was den Einstieg in das Framework vereinfacht.

2.5.2 Diesel

Eine weitere Bibliothek, die von uns verwendet wird, ist *Diesel* [30]. *Diesel* übernimmt das Wandeln von Objekten aus Objekt-Orientierten Programmiersprachen (OOP) - in unserem Fall Rust - in Relationen aus der relationalen Datenbank. Wie in dem *Abschnitt 2.4.1 - Grundlagen von relationalen Datenbanken* erläutert, wird standardmäßig über die Datenbanksprache SQL mit der Datenbank kommuniziert. Objekt Relational Mapping (ORM) Werkzeuge wie *Diesel* bieten Funktionen in der OOP, mit denen das Schreiben von SQL

übernommen wird. Dadurch können Entitäten aus der Datenbank in Objekte aus der OOP übersetzt werden. In dem *Code 15 - Beispiel einer SQL Anfrage mit Diesel* ist ersichtlich, wie die SQL Anfrage aus dem *Code 12 - Beispiel einer SQL Anfrage mit JOIN* mithilfe von *Diesel* nach Rust übersetzt wird.

```rust
pub fn fetch_user_and_books() -> Result<(),Error> {
    use crate::schema::books;
    use crate::schema::users;

    books::table
        .inner_join(users::table)
        .select((
            books::ISBN,
            books::name,
            users::ID,
            users::firstname,
            users::lastname,
        ))
        .filter(
            users::firstname.eq("Mustermann")
        )
        .load(conn)
}
```

Code 15 Beispiel einer SQL Anfrage mit Diesel

Zusätzlich bietet die Bibliothek Funktionen an, um einen Connection Pool zu erzeugen. Jede Verbindung mit der Datenbank kostet Zeit und Ressourcen. Um nicht bei jeder Datenbank Anfrage eine neue Verbindung aufzubauen und trotzdem mehrere Anfragen gleichzeitig zu stellen, werden Connection Pools verwendet. Ein Connection Pool hält mehrere Verbindungen offen und verteilt diese unter allen Anfragen.

2.5.3 Liste der verwendete Open Source Softwares

Neben *Diesel* und *Actix-Web* wurden noch weiter OSS verwendet, welche nun folgend aufgelistet sind.

chrono Das „chrono“-Paket ermöglicht die Arbeit mit Datums- und Uhrzeitangaben in Rust.
Version: 0.4.26
Lizenz: MIT

serde *serde* ist eines der am häufigsten verwendeten Rust-Pakete und ermöglicht die einfache *Serialisieren* und *Deserialisieren* von Rust-Strukturen.
Version: 1.0.174
Lizenz: Apache-2.0, MIT

serde_json Als Erweiterung von „serde“ spezialisiert sich „serde_json“ auf das *Serialisieren* und *Deserialisieren* von Daten im JSON-Format.
Version: 1.0.103
Lizenz: Apache-2.0, MIT

jsonschema Mit dem Paket „jsonschema“ können JSON Objekte mittels JSON Schema in Rust validiert werden.
Version: 1.17.1
Lizenz: MIT

utoipa Das Paket „utoipa“ automatisiert die Generierung von OpenAPI-Dokumentationen für Rust-Projekte.
Version: 4
Lizenz: Apache-2.0, MIT

utoipa-swagger-ui Dieses Paket stellt eine Benutzeroberfläche für OpenAPI-Dokumentationen bereit. Es ermöglicht die interaktive Erkundung und Nutzung der API über einen grafischen Webbrowser.
Version: 4
Lizenz: Apache-2.0, MIT

dotenv „dotenv“ ermöglicht das einfache Einlesen von Umgebungsvariablen aus einer „.env-Datei“.
Version: 0.15.0
Lizenz: MIT

argon2 Das Paket „argon2“ implementiert den Argon2-Algorithmus, ein modernes und sicheres Verfahren für das Hashen von Passwörtern. Es bietet eine zuverlässige Methode zum Speichern von Passwörtern in verschlüsselter Form.
Version: 0.5.1
Lizenz: Apache-2.0, MIT

rand_core „rand_core“ ist Teil des Rust-Random-Crates und ermöglicht die Generierung von Zufallsdaten. In diesem Kontext wird es verwendet, um Zufalls-Salt für das sichere Passwort-Hashing zu erstellen. Ein Zufalls-Salt ist eine zufällige Zeichenkette, die beim Passwort-Hashing mit berechnet wird. Dadurch entstehen auch bei

gleiche Passwörter trotzdem unterschiedliche Hashwerte.
Version: 0.6.4
Lizenz: Apache-2.0, MIT

jsonwebtoken Das Paket „jsonwebtoken“ bietet Funktionen zur Generierung von JWTs in Rust.
Version: 8.3.0
Lizenz: MIT

serial_test „serial_test“ ermöglicht das sequenzielle Ausführen von Tests in Rust.
Version: 2.0.0
Lizenz: MIT

toml Mit diesem Parser, wandeln wir *.toml* Dateien in Rust Strukturen.
Version: 0.8.2
Lizenz: Apache-2.0, MIT

actix-cors Das Paket „actix-cors“ erleichtert uns das Verwalten von Cross-Origin Resource Sharing (CORS) in Actix.
Version: 0.6.4
Lizenz: Apache-2.0, MIT

diesel-derive-enum „diesel-derive-enum“ ist ein Makro für Diesel, um Rust-ENUMS aus PostgreSQL-ENUMs zu generieren. Erleichtert das Arbeiten mit ENUMs
Version: 2.1.0
Lizenz: Apache-2.0, MIT

uuid Mit dem Paket „uuid“ erzeugen wir Universally Unique Identifiers (UUID) in Rust.
Version: 1.4.1
Lizenz: Apache-2.0, MIT

mongodb „mongodb“ ist eine Rust-Bibliothek für die Interaktion mit MongoDB. MongoDB ist eine DBMS für eine dokumentenorientierte Datenbank.
Version: 2.2.0
Lizenz: Apache-2.0

3 Benutzerverwaltung

Damit sich Personen an der REST-API zuordenbar und kontrolliert anmelden können, ist eine definierte Benutzerverwaltung erforderlich, um Personen und deren Rechte im System abzubilden. Die grundlegenden Informationen, die in der Datenbank über Nutzende hinterlegt werden müssen, sind eine E-Mail-Adresse, die eindeutig einer Person zuordenbar ist, ein Passwort und ein Benutzername. Die E-Mail-Adresse im Zusammenspiel mit dem Passwort kann bei jeder Neuanmeldung zur Identifizierung des Nutzers verwendet werden. Zusätzlich zu den grundlegenden Informationen über die Anwender, ist auch ein Konzept erforderlich, mit dem die Zugriffsrechte auf Ressourcen des Systems beschränkt werden können. Mit Ressourcen sind hierbei sowohl Daten, als auch Funktionalitäten des Projektes gemeint. Da das Framework offen für Weiterentwicklung sein soll, soll die Benutzerverwaltung Möglichkeiten zu Erweiterung bieten.

3.1 Zugriffsrechte über Rollen

Der erste Ansatz war, die Benutzerverwaltung über Berechtigungsgruppen zu konzeptionieren. So können die im System angemeldeten Individuen jeweils einer oder mehreren Berechtigungsgruppen zugeordnet werden. Eine Berechtigungsgruppe spiegelt dabei eine Rolle wider, welche die Person im System innehat. Der Zugriff auf eine Ressource ist nur Personen vorbehalten, die eine Rolle haben, die den Zugriff auf diese explizit erlaubt. Dabei kann definiert werden, ob eine Person diese Rolle im ganzen System oder nur innerhalb einer Lerngruppe besitzt. So könnte, wenn diese Berechtigung auf unser Framework angewandt wird, zum Beispiel ein Benutzer aufgrund seiner Rolle „Student“, in der Lerngruppe „Theo 3“ nur Aufgaben einsehen und lösen, aber in der „Theo 1“ Gruppe auch Aufgaben erstellen, da er dort die Rolle „Tutor“ begleitet.

3.1.1 Vorteile von Zugriffsrechten über Rollen

Der Ansatz, die Zugriffsrechte über Rollen zuzuweisen, hat den Vorteil, dass es übersichtlich ist, welcher Nutzer, welchen Rollen angehört und damit auf welche Ressourcen zugreifen kann. Zusätzlich, sind die benötigten Datenstrukturen, minimalistisch und übersichtlich. Bei neuen Ressourcen können diese für bestehende Rollen zugelassen werden und sind damit direkt für alle Mitglieder dieser Rollen erreichbar.

3.1.2 Nachteile von Zugriffsrechten über Rollen

Das Konzept bietet keine Möglichkeit Rechte individuell ins Detail für den Nutzer anzupassen. Auch bietet dieses Konzept keine Möglichkeit, einzelne Ressourcen individuell zu begrenzen. Zwar könnten sowohl die individuellen Rechte der Nutzenden, als auch die

Sicherung einzelner Ressourcen, durch die Anlage von extrem vielen Rollen umgesetzt werden. Jedoch geht dabei der wesentliche Vorteil in Bezug auf die Übersichtlichkeit der Rollen verloren.

3.2 Einzelne Zugriffsrechte auf Ressourcen

Da ein Konzept gesucht wird, das für jede Eventualität gewappnet ist, wurde in der Folge das Konzept der Zugriffsrechte über Rollen ausgeschlossen. Wir folgen nun dem Ansatz Rechte durch Einzel Berechtigungen abzubilden. Rechte können jedem Nutzenden einzeln zugeordnet werden und beliebig, zum Schutz vor unbefugtem Zugriff, auf die Ressourcen verteilt werden. Ein solches Recht kann innerhalb einzelner Lerngruppen oder im globalen System gelten. In unserem Framework könnte dadurch zum Beispiel „Max Mustermann“ nur Informationen zu der „Theo 2“ Gruppe einsehen, während „Lieschen Müller“ Informationen jeder Gruppe sieht, da sie das gleiche Recht jedoch im globalen System besitzt.

3.2.1 Vorteile von Zugriffsrechten einzelner Ressourcen

Dieses Konzept bietet die größte Freiheit im Umgang mit Berechtigungen, da individuell für alle Ressource Berechtigungen definiert und einzeln den Benutzern zugewiesen werden können. So bleibt die Zuweisung von Berechtigungen beliebig feingranular und kann bei weiterer Entwicklung möglichst viele Anwendungszwecke abbilden.

3.2.2 Nachteile von Zugriffsrechten einzelner Ressourcen

Dieser Ansatz ist deutlich komplexer im Vergleich zur Benutzerverwaltung über Rollen. Die Übersichtlichkeit, welche Person welches Recht besitzt, ist aufgrund der Individualisierung deutlich reduziert. Außerdem muss bei neuer Ressource, das benötigte Zugriffsrecht im Zweifelsfall an die relevanten Personen einzeln verteilt werden.

3.3 Unsere Umsetzung der Benutzerverwaltung

In diesem Framework haben wir ein Zwischenweg von den Zugriffsrechten über Rollen *Abschnitt 3.1 - Zugriffsrechte über Rollen* und dem der Einzel Rechte *Abschnitt 3.2 - Einzelne Zugriffsrechte auf Ressourcen* umgesetzt. Rechte werden einzeln auf Personen verteilt und Rollen dienen als Schablone um die Rechte effizienter und einfacher auf die Nutzer zu übertragen.

Benutzer stehen in der relationalen Datenbank mit den Ressourcen in einer *n:m* Beziehung. Sprich, eine Person kann auf mehrere Ressourcen Berechtigungen haben und eine Ressource kann von mehreren Personen verwendet werden. Um diese *n:m* Beziehung

umzusetzen, wurde eine dritte Relation verwendet, die einen Fremdschlüssel der Ressource sowie einen Fremdschlüssel der Person hat. Diese Relation hat den Namen *Nutzer_Berechtigung*. Um den Wirkungsbereich der Berechtigungen einzuschränken, besitzt diese Nutzer_Berechtigungs-Relation noch einen dritten Fremdschlüssel. Dieses Fremdschlüsselattribut hat die Eigenschaft *NULLABLE* und gibt die Lerngruppe an, in der die Berechtigung gilt. Sollte eine Entität für dieses Attribut den Wert *NULL* (Leer) haben, gilt die Berechtigung im globalen System und gibt somit in allen Gruppen die Erlaubnis auf die Ressource. Die *Abbildung 2 - ERM: Einfache Benutzerverwaltung* beschreibt diese Relationen und ihre Beziehungen. In diesem Modell sind nur die für die Benutzerverwaltung erforderlichen Attribute abgebildet. Die Relation *Ressourcen* besteht aus den

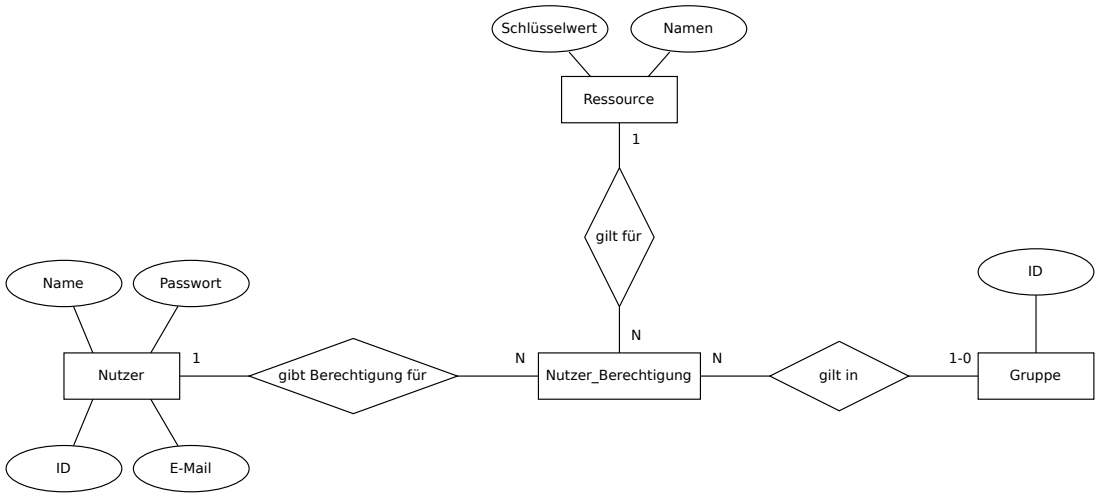

Abbildung 2 ERM: Einfache Benutzerverwaltung

Attributen *Name* und *Schlüsselwert*. Der *Name* ist dabei für die Benutzerverwaltungslogik irrelevant und dient dem Menschen zur Benennung der Ressource. Der Primärschlüssel *Schlüsselwert* hat den Datentyp *VARCHAR(45)* (Zeichenkette mit maximaler Größe von 45 Bytes) und wird im Programmcode verwendet, um diese zu identifizieren.

Rollen können in unserem Konzept als Sammlung von Zugriffsrechte gesehen werden, welche schablonenartig auf Personen übertragen werden können. Wichtig ist, das Rollen nicht wie in dem *Abschnitt 3.1 - Zugriffsrechte über Rollen* Berechtigungsgruppen für Nutzende darstellen. Veränderung von Rollen führt bei Personen demnach zu keiner Zugriffsrechtsänderung.

Rollen stehen in der relationalen Datenbank in einer *n:m* Beziehung zu den Ressourcen. Äquivalent zu der Nutzerberechtigungs-Relation existiert eine Relation mit dem Namen *Rollen_Berechtigung*. Diese hat jedoch im aktuellen Status, des Frameworks, keine Beziehung zu den Gruppen. Der Aufbau dieser Relationen kann in der *Abbildung 3 - ERM Einfache Rollenverwaltung* gesehen werden.

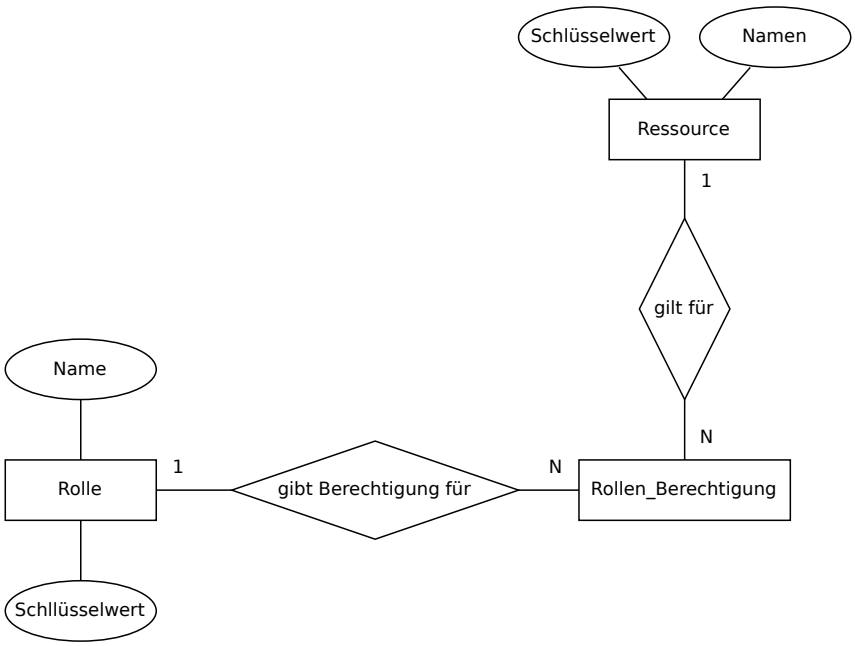

Abbildung 3 ERM Einfache Rollenverwaltung

3.3.1 Zugriffsarten auf Ressourcen

Entitäten in der *Nutzer_Berechtigung* bilden den Zugriff auf die Ressourcen ab. Um die Übersichtlichkeit der Berechtigungen nicht zu verlieren, wird eine Unterscheidung von Zugriffsarten verwendet. Diese Unterscheidung der Zugriffsarten wird in der Datenbank durch die Verwendung der Relation *Nutzer_Zugriffsart* abgebildet. Diese steht in einer *1:n* Beziehung zu den Nutzer_Berechtigungen und gibt vor, wie ein Nutzer auf eine Ressource zugreifen darf. In der *Abbildung 4 - ERM: Zugriffsarten* ist die *Nutzer_Zugriffsart*-Relation ersichtlich.

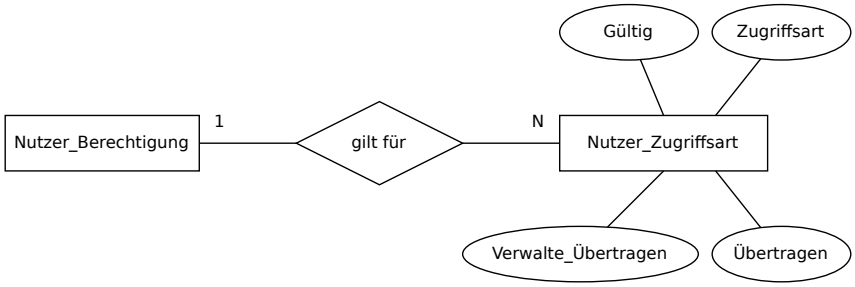

Abbildung 4 ERM: Zugriffsarten

Die Relation besitzt neben dem Fremdschlüssel der Nutzer_Berechtigung noch vier weitere Attribute. Das Attribut *Zugriffsart* mit dem Datentyp Aufzählung (*ENUM*) gibt die Art des Zugriffes an. Mögliche Zugriffsarten sind: Lesend, Schreibend, Löschend und Fremdzugriff. Diese können jedoch zukünftig auch erweitert werden. Über das Attribut *Gültig* mit dem Datentyp Wahrheitswert (*Boolean*), wird definiert, ob eine Person diese Zugriffsart verwenden darf. Die verbleibenden beiden Boolean-Attribute *Übertragen* und *Verwalte_Übertragen* haben administrative Aufgaben. Über das Attribut mit dem Namen

Übertragen, wird definiert, ob ein Individuum einer anderen Person dieses Recht der Zugriffsart auf die Ressource übergeben darf. *Verwalte_ Übertragen* ist wiederum definiert als das Recht, um das Attribut *Übertragen* von anderen Nutzenden zu ändern. Damit diese Kette nicht endlos lange wird, kann mit Besitz dieses *Verwalte_ Übertragen* Recht auch dasselbige bei Anderen hinzugefügt oder entfernt werden.

Um die Anzahl der Datensätze in der Datenbank zu reduzieren, werden alle Entitäten der Relation *Nutzer_ Zugriffsart*, die die Booleans *Gültig*, *Übertragen* und *Verwalte_ Übertragen* auf *FALSE* haben, gelöscht. Dies passiert über einen Trigger, welcher nach einem Datenupdate automatisch ausgeführt wird. Dieser Trigger löscht beim Updaten eines Eintrages, den Eintrag aus der Datenbank, sollten alle dieser Attributwerte FALSE lauten. Zusätzlich verhindert ein zweiter Trigger, dass ein Eintrag mit nur FALSE Werten erzeugt werden kann. Diese beiden Trigger sind in dem *Code 18 - UPDATE-TRIGGER zum Löschen von Nutzer_ Zugriffsart-Einträgen* und dem *Code 19 - INSERT-TRIGGER zum Löschen von Nutzer_ Zugriffsart-Einträgen* abgebildet.

Da nicht jede Zugriffsart auf jede Ressource angewandt werden kann, wurden die Ressourcenrelation um eine Liste von möglichen Zugriffsarten erweitert. Diese Liste entsteht durch die Relation *Ressourcen_ Zugriffsart*, die mit einem Fremdschlüssel zu einer Ressource und dem Attribut *Zugriffsart* ausgestattet ist. Um sicherzustellen, dass zu keinem Zeitpunkt eine Zugriffsart existiert, welche sich nicht in dieser Liste befindet, werden zwei Trigger verwendet. Diese Trigger sind in dem *Code 16 - TRIGGER zum Validieren von Nutzer_ Zugriffsart-Einträgen* und dem *Code 17 - DELETE-TRIGGER zum Löschen von Nutzer_ Zugriffsart-Einträgen* aufgeführt. Der *valid_ access_ type TRIGGER* sorgt dafür, das nur Entitäten der *Nutzer_ Zugriffsart* Relation erzeugt werden können, die eine Zugriffsart besitzt, die in der List der möglichen Zugriffsarten dieser Ressource angegeben ist. Beim Löschen einer *Ressourcen_ Zugriffsart* Entität werden auch alle entsprechenden *Nutzer_ Zugriffsart* Entitäten gelöscht. Dies wird durch den zweiten *TRIGGER* durchgesetzt.

In der *Abbildung 5 - ERM: Benutzerverwaltung* sind alle Relationen der Benutzerverwaltung zu sehen.

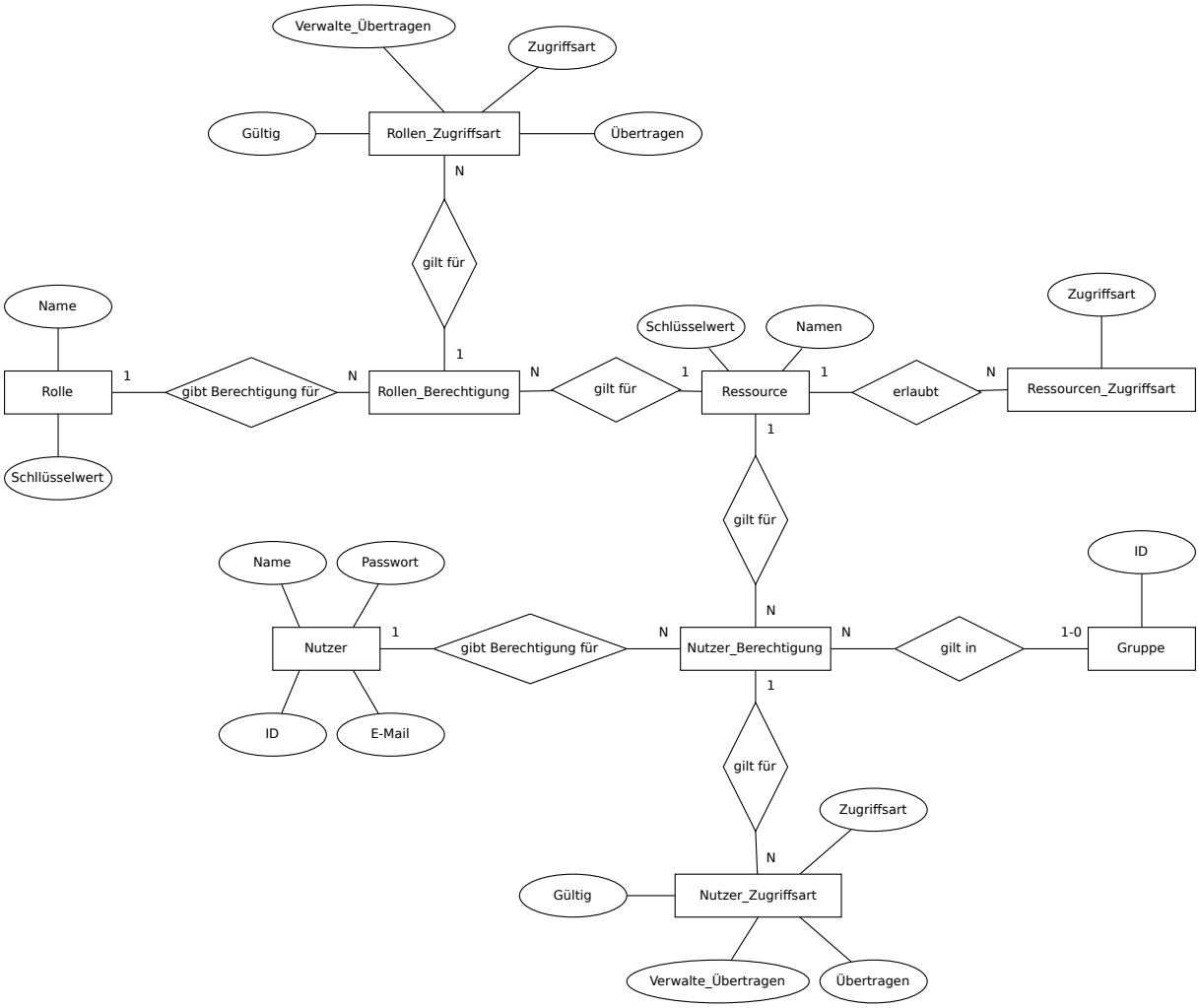

Abbildung 5 ERM: Benutzerverwaltung

```
CREATE FUNCTION valid_access_type() RETURNS trigger AS $valid_access_type$
    DECLARE
        ressource_key_value VARCHAR(45);
        valid boolean;
    BEGIN
        ressource_key_value := (
            SELECT up.ressource
            FROM user_permissions up
            WHERE up.id = NEW.user_permission_id
        );
        valid := (EXISTS (
            SELECT 1
            FROM ressources r
            INNER JOIN ressources_access_types rat ON rat.ressource =
↪   r.key_value
            WHERE rat.access_type = NEW.access_type
        ));

        IF valid = False THEN
            RAISE EXCEPTION 'Ressource do's not allow this access_type';
        END IF;

        RETURN NEW;
    END;
$valid_access_type$ LANGUAGE plpgsql;

CREATE TRIGGER valid_access_type BEFORE INSERT or UPDATE ON
↪   user_access_types
    FOR EACH ROW EXECUTE PROCEDURE valid_access_type();
```

Code 16 TRIGGER zum Validieren von Nutzer_Zugriffsart-Einträgen

```
    CREATE FUNCTION delete_access_types() RETURNS trigger AS
↪   $delete_access_types$
    BEGIN
        DELETE FROM user_access_types uat
        USING user_permissions up
        WHERE up.id = uat.user_permission_id
        AND up.ressource = OLD.ressource
        AND uat.access_type = OLD.access_type;
    END;
$delete_access_types$ LANGUAGE plpgsql;

CREATE TRIGGER delete_access_types AFTER DELETE ON ressources_access_types
    FOR EACH ROW EXECUTE PROCEDURE delete_access_types();
```

Code 17 DELETE-TRIGGER zum Löschen von Nutzer_Zugriffsart-Einträgen

```
1   CREATE FUNCTION delete_if_all_false() RETURNS trigger AS
    ↪ $delete_if_all_false$
2   BEGIN
3       IF NEW.permission = False
4       AND NEW.set_permission = False
5       AND NEW.set_set_permission = False
6       THEN
7           DELETE FROM user_access_types
8           WHERE user_permission_id = NEW.user_permission_id
9           AND access_type = NEW.access_type;
10      END IF;
11
12      RETURN NEW;
13  END;
14 $delete_if_all_false$ LANGUAGE plpgsql;
15
16 CREATE TRIGGER delete_if_all_false AFTER UPDATE ON user_access_types
17      FOR EACH ROW EXECUTE PROCEDURE delete_if_all_false();
```

Code 18 UPDATE-TRIGGER zum Löschen von Nutzer_Zugriffsart-Einträgen

```
1   CREATE FUNCTION valid_user_access_type() RETURNS trigger AS
    ↪ $valid_user_access_type$
2   BEGIN
3       IF NEW.permission = False
4       AND NEW.set_permission = False
5       AND NEW.set_set_permission = False
6       THEN
7           RAISE EXCEPTION 'Access_type is invalid';
8       END IF;
9
10      RETURN NEW;
11  END;
12 $valid_user_access_type$ LANGUAGE plpgsql;
13
14 CREATE TRIGGER valid_user_access_type BEFORE INSERT ON user_access_types
15      FOR EACH ROW EXECUTE PROCEDURE valid_user_access_type();
```

Code 19 INSERT-TRIGGER zum Löschen von Nutzer_Zugriffsart-Einträgen

4 Authentifizierung

Da die REST-API im Internet Schnittstellen bereitstellt, müssen Anfragen von Unbekannten blockiert werden. Es dürfen nur Zugriffe von Personen auf die Datenbank zugelassen werden, welche in dieser registriert sind. Das Validieren der Person gegenüber der Datenbank wurde mittels JWTs gelöst. Was JWTs sind, haben wir in dem *Abschnitt 2.3 - Was ist ein JSON Web Token?* erläutert. Zusätzlich zur Überprüfung von Unbekannten Zugriffen braucht es auch einen Weg die Benutzerverwaltung aus dem *Kapitel 3 - Benutzerverwaltung* im Framework anzuwenden.

4.1 JWT-Middleware

Um sicherzustellen, dass nur im System angemeldete Personen auf Endpunkte der REST-API zugreifen, muss bei jedem Zugriff der JWT der jeweiligen Anfrage auf ihre Korrektheit überprüft werden. Um nicht in jedem Endpunkt aufs Neue die Überprüfung des JWTs programmieren zu müssen, wurde eine Möglichkeit gesucht, wie dies zentral zu lösen ist. Eine gängige Methode dies umzusetzen ist es eine Middleware zu schreiben. Diese wird zentral programmiert und vor die gewünschten Endpunkte geschaltet. Da wir uns zu Beginn der Entwicklung an dem Artikel, „Rust - JWT Authentication with Actix Web“ von Edem Kwame [31], orientiert haben, ist diese Middleware mittels einer Implementierung von *FromRequest* umgesetzt. Dabei gleicht die Umsetzung in diesem Framework nahezu der aus dem genannten Artikel. Einzig die Überprüfung der im JWT übergebene User ID ist als Funktion hinzugekommen. Der Extraktor FromRequest kann auf eine *Rust-Struktur* implementiert werden, wodurch diese eine Funktion bereitstellt, welche vor Aufruf eines Endpunktes ausgeführt wird.
In dem *Code 20 - Beispielverwendung der JWT-Middleware* ist ein Beispiel zu sehen, wie die Middleware angewandt werden kann. Der Endpunkt „\id“ gibt als Antwort die ID des Nutzenden zurück. Ist in der Anfrage kein gültiger *Access-Token* enthalten, wird die Middleware automatisch mit folgender Antwort antworten:

```
HTTP/1.1 401 Unauthorized
```

Rückblickend wäre die Umsetzung einer Middleware über „Service trait“ und „Transform trait“ [32] besser gewesen. Durch eine solche Umsetzung hätte die Middleware, vor ganze Abzweigungen des Routing-Baumes geschaltet werden können. Diesen haben wir in dem *Abschnitt 2.5.1 - Actix Web* benannt. Hierdurch muss die Middleware nicht in jedem Endpunkt angegeben werden.

```rust
use actix_web::{get, Responder};
use crate::jwt;

#[get("/id")] // Endpunkt unter dem lokalen Pfad "/id"
async fn get_index(
    jwt: jwt::JwtMiddleware, // Hier wird die Middleware angewandt
) -> impl Responder {
    jwt.user_id // Die ID des users als Antwort
}

```

Code 20 Beispielverwendung der JWT-Middleware

4.2 Permission-Middleware

Neben der JWT-Middleware, welche überprüft, ob eine Person im System angemeldet ist, braucht es auch eine Lösung die Benutzerverwaltung aus dem *Kapitel 3 - Benutzerverwaltung* in der REST-API anzuwenden. Um die Benutzerverwaltung, die in der relationale Datenbank modelliert ist umzusetzen, wurde anfangs der Ansatz verfolgt Attribut-ähnliche Makros zu verwenden.

4.2.1 Benutzerverwaltung durch Attribut-ähnliche Makros

Der Ansatz ist, die Zugriffskontrolle der Ressourcen, über Makros umzusetzen. Attribut-ähnliche Makros sind eine Eigenschaft der Sprache Rust. Diese können unter anderem auf Funktionen angewandt werden. Eine *Makrodefinitionsfunktion* nimmt dann diese Funktion als *TokenStream* zusammen mit Attributen entgegen. Innerhalb dieser Makrodefinitionsfunktion können Veränderungen auf die Funktion angewandt und als Rückgabewert zurückgegeben werden [33]. Wie ein solches Attribut-ähnliches Makro in der Anwendung hätte aussehen können, ist in dem *Code 21 - Beispiel der Benutzerverwaltung mit Makros* ersichtlich. Über den Code in der ersten Zeile hätte der Schlüsselwert der Ressourcen Entität zusammen mit der benötigten Zugriffsart angegeben werden können. Das Makro hätte dann die Funktion so verändert, dass Anfangs die Anfrage auf eine passende Berechtigung überprüft wird.
Durch diesen Ansatz steht die Deklaration der benötigten Rechte an derselben Stelle, wie der Endpunkt selbst. Somit ist bei Lesen des Codes einfach ersichtlich, welche Berechtigung für welchen Endpunkt gefordert ist.
Letztendlich haben wir uns jedoch dazu entschieden den Ansatz fallen zu lassen. Die Umsetzung hatte sich als komplizierter herausgestellt als Anfangs erwartet. Aufgrund dessen wurde die Kontrolle der Rechte über eine Middleware umgesetzt.

```rust
#[permission("group", "create")]
#[post("/")]
pub async fn create_group(
    ...
) -> HttpResponse { ... }
```

Code 21 Beispiel der Benutzerverwaltung mit Makros

4.2.2 Benutzerverwaltung über eine Middleware

Wie auch schon in der Umsetzung der JWT-Middleware aus dem *Abschnitt 4.1 - JWT-Middleware*, wird die Middleware über eine Implementierung von FromRequest umgesetzt. Der Grund für diese Entscheidung war der bereits bekannte Umgang dieser Implementierung. Auch hier wird mit Sicherheit eine Umsetzung über „Service trait“ und „Transform trait“ sinnvoller sein. Inhaltlich ändert sich dadurch jedoch nichts an der Funktionsweise. Abweichend zu dem Ansatz über die Attribut-ähnliche Makros aus dem *Abschnitt 4.2.1 - Benutzerverwaltung durch Attribut-ähnliche Makros*, werden Middleware vor der Funktion der Endpunkte ausgeführt. Durch diese Eigenschaft können von der Funktion keine Attribute an die Middleware übergeben werden. Folgend muss die Middleware von sich aus für jede Anfrage, die angefragt Ressource, sowie die Zugriffsart kennen. Wir lösen dieses Problem über eine Tabelle (*Map*). Diese Tabelle bildet jede Anfrage auf die benötigten Rechte ab. Da die Rechte der Benutzerverwaltung abhängig von Lerngruppen sein können, braucht die Middleware demnach auch die Information, ob sich eine Ressource in einer Lerngruppe befindet. Eine einfache Möglichkeit ist zu definieren, dass die Lerngruppen ID als Pfad-Variable immer gleich benannt wird. So kann die Middleware, immer wenn in dem Pfad diese Pfad-Variable vorhanden ist, die entsprechende Lerngruppe identifizieren. Das Definieren von einheitlichen Benennungen ist grundsätzlich eine gute Idee, kann jedoch nicht erzwungen werden. Eine Implementierung auf diesem Konzept würde demnach den Umgang mit dieser Middleware nur verkomplizieren. In unserer Implementierung wurde die Gruppeninformation, neben den Berechtigungen mit in die Tabelle gelistet.

4.2.2.1 Berechtigungstabelle für die Permission-Middleware

Für die Tabelle verwenden wir die Datenstruktur *HashMap* [34]. Die Datenstruktur HashMap hat die Eingenschaft, dass Zugriffe auf die Einträge in konstanter Zeit möglich ist. Wir speichern unter jedem Pfad eine weitere HashMap, die mit einer HTTP-Methode drei Attribute indexiert. Das Attribut *Ressource* gibt den Schlüsselwert der zugehörigen Ressourcen Entität an. Über das Attribut *Group_ID_Prameter* wird angegeben, wo die Middleware die Gruppen ID aus dem Pfad lesen kann. Dass Listenattribut *Benötigte_Zugriffsarten* gibt an, welche Zugriffsarten von einem Endpunkt vorausgesetzt werden. Wir befüllen diese Tabelle beim Starten der REST-API. Wir lesen die benötigten Infor-

mationen aus einer Einstellungs-Datei, welche das Dateiformat Tom's Obvious, Minimal Language (TOML) hat. In dem *Code 22 - Auschnitt permission.toml* ist ein Ausschnitt dieser Einstellungs-Datei ersichtlich. In dem Ausschnitt sind die Ressourcen: „group“ und „schema“ angegeben. Die Endpunkte in den Zeilen 6, 11 und 17 werden durch die Datei so konfiguriert, dass anfragende Nutzer die Berechtigung auf die Ressource mit dem Schlüsselwert „group“ benötigen. Die in den Zeilen 26 und 31 angegebenen Endpunkte, setzen bei Anfragen eine Berechtigung auf die Ressource mit dem Schlüsselwert „schema“ voraus.

```
1  [config]
2
3  [[config.ressources]]
4  value = "group" #Der Ressourcen-Schlüsselwert.
5
6  [[config.ressources.routes]]
7  path = "/api/group/"     #Pfad des Endpunktes.
8  method = "POST"          #Die HTTP Methode des Endpunktes.
9  required_access_types = ['Create']  #Liste der benötigten Zugriffsarten.
10
11 [[config.ressources.routes]]
12 path = "/api/group/{group_id}"
13 param = "group_id"       #Wo im Pfad die Gruppen ID zu finden ist.
14 method = "DELETE"
15 required_access_types = ['Delete']
16
17 [[config.ressources.routes]]
18 path = "/api/group/{group_id}"
19 param = "group_id"
20 method = "GET"
21 required_access_types = ['Read']
22
23 [[config.ressource]]
24 value = "schema"
25
26 [[config.ressource.routes]]
27 path = "/api/tasks/schemas"
28 method = "GET"
29 required_access_types = ["Read"]
30
31 [[config.ressource.routes]]
32 path = "/api/tasks/schemas"
33 method = "POST"
34 required_access_types = ["Create"]
```

Code 22 permission.toml

(Ein wenig auf TOML eingehen)

4.2.2.2 Funktionsweise der Permission-Middleware

Um die Kontrolle der Zugriffsrechte sicherzustellen wird vorausgesetzt, dass die JWT-Kontrolle erfolgreich abgeschlossen wurde. Über die Berechtigungstabelle aus dem *Abschnitt 4.2.2.1 - Berechtigungstabelle für die Permission-Middleware*, werden die benötigten Informationen für die Anfrage gesucht. Über den Nutzenden Identifikator, welcher durch die JWT-Middleware zur Verfügung gestellt wird, werden die Berechtigungen aus der Datenbank gelesen. Sollten der Anfragende nicht die benötigten Zugriffsrechte besitzen, antwortet die Middleware auf die Anfrage mit:

```
HTTP/1.1 403 Forbidden
```

5 Gruppen

Gruppen bilden die Grundstrukturen des Frameworks. Die meisten Funktionen, welche das Framework bereitstellt, beziehen sich auf eine Gruppe. Diese Gruppen verhalten sich wie Ordner. In einer solchen Gruppe können andere Gruppen und Aufgabenpakete liegen.

5.1 Datenstruktur der Gruppen

Die Gruppen werden in der relationalen Datenbank abgebildet. Die Gruppenrelation besteht dabei hauptsächlich aus den Attributen: Name, Status und einer übergeordneten Gruppe (Elternteil). Der Status einer Gruppe kann entweder „Aktiv“ oder „Gelöscht“ annehmen. Über diesen Status können Gruppen *vorläufig gelöscht* (soft deleted) werden. Vorläufiges Löschen bedeutet in diesem Zusammenhang, dass der Status eines Eintrages auf „Gelöscht“ geändert und nicht direkt aus der Datenbank entfernt wird. Dies hat den Vorteil, dass das Löschen der Einträge unabhängig von Nutzeranfragen durchgeführt werden kann. Auch können noch nicht vollständig gelöschte Einträge wieder hergestellt werden, da sich die Einträge noch in der Datenbank befinden.

Das Elternteil-Attribut ist ein *NULLABLE* Fremdschlüssel auf die eigene Relation. Durch diesen Fremdschlüssel ist die Gruppenrelation in einer *(0..1):n* Beziehung mit sich selbst. *Eine Gruppe hat eine oder keine übergeordnete Gruppe (Elternteil). Einer Gruppe kann mehrere Untergruppen haben.*

Über die Relation *Gruppen_Mitglied* können einer Gruppe Benutzer als Mitglieder hinzugefügt werden. Dies ist eine einfache Relation, welche die Gruppenrelation und die Nutzerrelation in einer *n:m* Beziehung miteinander stehen lässt.

5.2 Funktionalität von Gruppen

Wie einleitend erklärt, verhalten sich die Gruppen aus diesem Framework wie Ordner. Sie bündeln die Benutzer und Aufgabenpakete unter einem Gruppennamen. Über die Rekursion, durch das Elternattribut können Gruppen beliebig tief angelegt werden.

Beim Erstellen einer neuen Gruppe wird dem Ersteller automatisch alle Berechtigung der Rolle „created_group“ übertragen. Dadurch wird vermieden, dass einem Erzeuger die Rechte fehlen, auf seine neu erzeugte Gruppe zugreifen. Dies wäre sonst der Fall, wenn er nicht das globale Recht hat, auf alle Gruppen zuzugreifen. Mittels dieser Rolle können also die Rechte angepasst werden, die ein Ersteller einer neuen Gruppe grundsätzlich erhalten soll.

Das gleiche Prinzip wird auch beim Hinzufügen eines Gruppenmitgliedes über die Rolle „new_member“, angewandt. Bei Entfernen eines Gruppenmitgliedes werden dem Nutzer zusätzlich alle auf diese Gruppe spezifischen Rechte entzogen.

6 Aufgaben

Eine zentrale Funktion des Frameworks besteht darin Aufgaben bereitzustellen, die von Nutzern gelöst werden können. Diese Aufgaben werden von anderen Benutzern in Form von Aufgabenpaketen innerhalb von Lerngruppen angeboten.

6.1 Aufgabendokument

Die Speicherung von Aufgaben erfolgt, wie im *Abschnitt 2.4.3 - Dokumentenorientierte Datenbank* erläutert, in der dokumentenorientierten Datenbank. Ein Aufgabendokument enthält die Informationen: Aufgabe, Lösung, Aufgabentyp und Status. Der Aufgabentyp gibt dabei die Art der Aufgabe an, zum Beispiel "Multiple-Choice", wie in dem *Code 23 - Beispiel eines Aufgabendokumentes* dargestellt.

```
{
    "_id": "17e907ed-f1dc-4d41-ad11-a1963d36a60e",
    "task_type": "Multiple-Choice",
    "task": {
      "question": "Welcher Begriff beschreibt Tabellen in der Relationalen
↪  Datenbank?",
      "answers": [
        "Entität",
        "Attribut",
        "Attributwert",
        "Relation"
      ]
    },
    "state": "ACTIVE",
    "solution": {
      "solution": 3
    }
  }
```

Code 23 Beispiel eines Aufgabendokumentes

6.2 Aufgabenpakete

Aufgaben werden innerhalb von Lerngruppen in sogenannten Aufgabenpaketen verteilt. Diese Pakete sind Sammlungen von Aufgaben, die den Lernenden zu Verfügung stehen. Für jedes Aufgabenpaket können Nutzer ein oder mehrere Lösungsversuche starten. Ein Lösungsversuch ermöglicht es dem Studenten, zu jeder Aufgabe innerhalb des Aufgabenpaketes eine Antwort zu Speichern. Nach dem Beenden (Abgeben) einer Antwort, kann diese nicht mehr bearbeitet werden.

6.3 Datenstruktur von Aufgabenpaketen

Aufgabenpakete sind als Relationen in der relationalen Datenbank definiert. Diese Relationen sind mit dem Primärschlüssel ID und weiteren Attribute wie Name, Status, Pakettyp und dem Fremdschlüssel *Gruppen_ID* ausgestattet. Durch diesen Fremdschlüssel steht die Aufgabenpakete-Relation in einer *1:n* Beziehung mit der Gruppen-Relation. *Ein Aufgabenpaket liegt in einer Gruppe. Eine Gruppe kann mehrere Aufgabenpakete besitzen.* Um ein vorläufiges Löschen zu ermöglichen wird das Status Attribut verwendet. Über den Pakettypen eines Aufgabenpaketes sollen Eigenschaften eines Aufgabenpaketes deklariert werden können. Zum Beispiel kann für manche Aufgabentypen jeder Nutzende nur ein Lösungsversuch anlegen. Dies wurde jedoch in dem jetzigen Status des Frameworks noch nicht umgesetzt.
Die Aufgabendokumente stehen in einer *n:m* Beziehung zu der Aufgabenpaketrelationen. *Jede Aufgabe kann in mehreren Aufgabenpaketen angeboten werden. Ein Aufgabenpaket kann mehrere Aufgaben beinhalten.* Eine zusätzliche Aufgaben-Relation in der relationalen Datenbank bildet diese Beziehung ab. Die Relation hat einen Fremdschlüssel zu den Aufgabenpaketen und speichert die *_ID* des Aufgabendokuments. Um die Anzahl der Datenbankanfragen zu verringern, steht in dieser Relation noch einmal der Aufgabentyp des Aufgabendokuments. Auch Entitäten dieser Relation können über einen Status vorläufig gelöscht werden.
Lösungsversuche stehen in einer *1:n* Beziehung zu den Aufgabenpaketen und den Nutzenden. Zusätzlich zu den benötigten Fremdschlüsselattributen enthalten sie ein Sichtbarkeit-Attribut. Dieses Attribut gibt an, ob ein Lösungsversuch von anderen Benutzern angeschaut und beendet werden kann.
Antworten der Benutzer werden ebenfalls in der dokumentenorientierten Datenbank gespeichert. Das machen wir aus den gleichen Gründen wie auch schon bei den Aufgaben. Antwortdokumente stehen in einer *1:n* Beziehung zu den Lösungsversuchen. *Jedes Antwortdokument gehört einem Lösungsversuch an. Ein Lösungsversuch kann mehrere Antwortdokumente beinhalten.* Eine Antwort-Relation in der relationalen Datenbank bildet die Antworten in der relationalen Datenbank ab. Diese Relation besitzt ein Attribut, welches auf das zugehörige Antwortdokument verweist, sowie die Fremdschlüssel zu den Nutzenden, Lösungsversuchen und der Aufgaben-Relation. Ein zusätzliches *Boolean* Attribut gibt an, ob das Antwortdokument, der korrekten Lösung der Aufgabe entspricht.
Die komplette Datenstruktur kann in der *Abbildung 6 - ERM: Aufgaben* eingesehen werden.

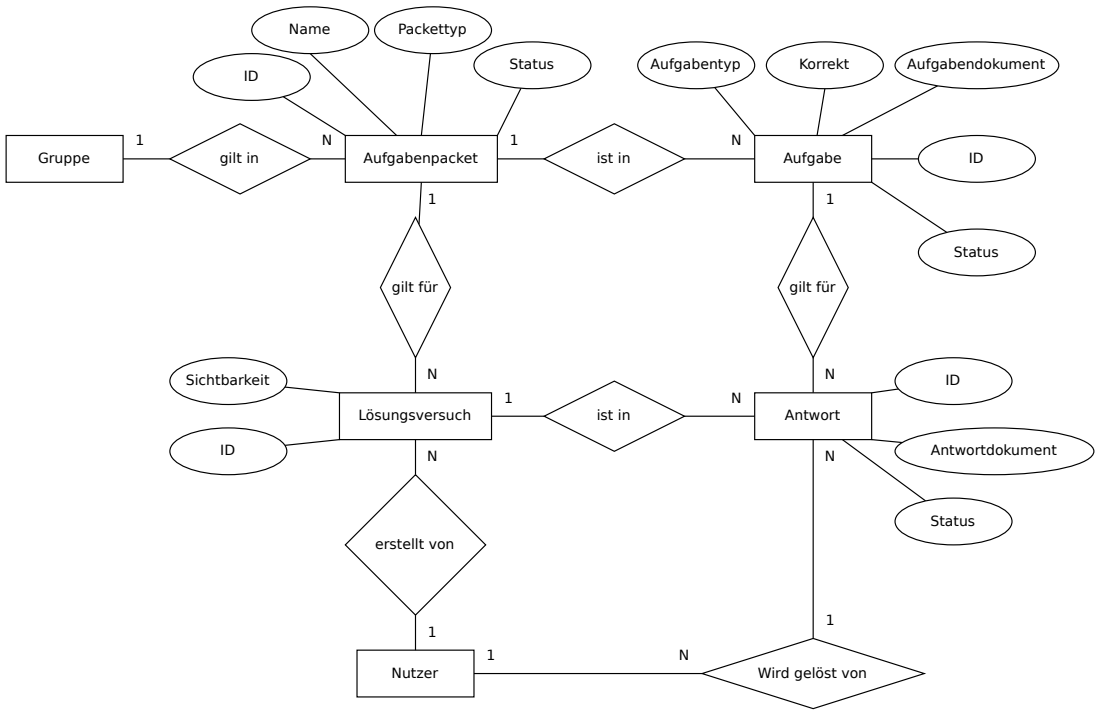

Abbildung 6 ERM: Aufgaben

6.4 Anwendung der Aufgaben

Über Endpunkte der REST-API, können Aufgabendokumente erstellt werden. Aufgabendokumente, die im Framework hochgeladen wurden, können nicht wieder verändert werden. Dies hat den Hintergrund, dass Aufgaben nicht nachträglich angepasst werden, wenn schon Antworten dafür erstellt wurden. Bei dem Hinterlegen von Aufgabendokumente in die Aufgabenpakete wird automatisch eine Aufgaben-Entität erzeugt. Wird nun auf ein Aufgabenpaket ein Lösungsversuch erstellt, stellt das Framework für jede Aufgabe, die sich in diesem Paket befindet, eine Antwort-Entität. Diese Antwort-Entitäten werden mit dem Benutzer verbunden, der den Lösungsversuch beginnt. Da zu diesem Zeitpunkt die Antwortdokumente, also die eigentliche Antwort des Lernenden, nicht hochgeladen wurde, wird der Attributwert dieses Attributs auf ein Standard UUID-Wert gesetzt. Über einen Endpunkt kann nun der Student zu diesen Antworten, Antwortdokumente hochladen und verändern. Ist eine Antwort beendet, lässt das Framework keine Veränderung an dem Antwortdokument mehr zu. Dieser Statuswechsel einer Antwort kann durch den Nutzenden selbst oder einer anderen Person erfolgen. Beim Beenden des Lösungsversuches vergleicht das Framework alle Antwortdokumente der Antworten mit dem Lösungsdokumente der Aufgabe. Diese Information wird in den Antwort-Entitäten hinterlegt.

Da jedes Dokument abhängig des Aufgabentypen unterschiedlich aufgebaut ist, ist das Überprüfen des Formates schwierig. Wir müssen zur Laufzeit des Frameworks den Aufbau eines abgegebenen Dokuments abhängig seines Aufgabentyps validieren. Dabei soll jedoch auch möglich sein, dem Framework zur Laufzeit neue Aufgabentypen beizubringen.

6.5 Schemas

Diese Schwierigkeit der Validierung des Dokumentenformates, lösen wir über JSON Schemas [35]. JSON Schemas sind JSON Objekte, die das Format von anderen JSON Objekten beschreiben und validieren können. In dem JSON Schema aus dem *Code 24 - JSON Schema* ist ein Schema für die Aufgabe im Aufgabendokument aus dem *Code 23 - Beispiel eines Aufgabendokumentes* ersichtlich. Mit diesem Schema können wir nun JSON Objekte auf die Attribute „question“ - einer Zeichenkette - und „answers“ - einer Liste an Zeichenketten - überprüfen.

Die REST-API bietet über ein Endpunkt die Möglichkeit, neue Schemas zu speichern. Diese werden in der dokumentenorientierten Datenbank zusammen mit dem dazugehörigen Aufgabentyp gespeichert. Dabei wird immer ein Aufgabenschema und ein Lösungsschema zu jedem Aufgabentyp hinterlegt. Bei Hochladen eines Lösungsdokumentes oder einem Aufgabendokument wird nun das richtige JSON Schema aus der Datenbank gelesen und damit das Dokument validiert.

```
{
  "task_schema": {
    "type": "object",
    "properties": {
      "question": {
        "type": "string"
      },
      "answers": {
        "type": "array",
        "items": {
          "type": "string"
        }
      }
    },
    "required": [
      "question",
      "answers"
    ],
    "additionalProperties": false
  }
}
```

Code 24 JSON Schema

6.6 Auswertung des Lernerfolges

Über die Antwort Relation kann das Framework den Lernerfolg der Nutzer ausgeben. Dies findet in Form von korrekten Aufgaben innerhalb von Lösungsversuchen statt. Über diese

Berichte können Informationen, über die abgeschlossenen Lösungsversuche der Teilnehmer eingesehen werden. Benutzer können den Lernerfolg aller ihrer Lösungsversuche einsehen und, sollten sie das Recht dazu haben, bei anderen Nutzern deren öffentliche Lösungsversuche. Im *Code 25 - Nutzerstatistik über Lernerfolg innerhalb eines Aufgabenpaketes* ist ein Beispiel diese Auswertung ersichtlich. Über „amount_task“ wird die Anzahl der Aufgaben angegeben und mittels der „values“ Liste werden alle einsehbaren Lösungsversuche mit Zeitpunkt und Anzahl korrekter Lösungen ausgegeben. Mittels diesen Informationen kann der zeitliche Lernerfolg zu einem Aufgabenpaket eingesehen werden.

```
{
  "amount_tasks": 2,
  "values": [
    {
      "solution_attempt_id": "8cd88ea0-c008-47d9-85bc-521d8c7c8c05",
      "date": "2023-12-28T22:58:27.998673Z",
      "amount_correct": 0
    },
    {
      "solution_attempt_id": "2b8e8312-7b26-4575-a178-fef19f26030d",
      "date": "2023-12-28T23:45:15.039018Z",
      "amount_correct": 0
    },
    {
      "solution_attempt_id": "a6aba409-a6e8-458e-86a5-0660d434992d",
      "date": "2023-12-31T13:08:26.596293Z",
      "amount_correct": 1
    },
    {
      "solution_attempt_id": "06b48a1e-17c3-423d-a283-2cdaca68f1de",
      "date": "2023-12-31T13:10:45.595689Z",
      "amount_correct": 2
    }
  ]
}
```

Code 25 Nutzerstatistik über Lernerfolg innerhalb eines Aufgabenpaketes

7 Proof of Concept

Um die Funktionsweise unseres Frameworks zu demonstrieren, haben wir eine App entwickelt. Diese Anwendung ist mithilfe des Flutter-Frameworks [36] in der Programmiersprache Dart verfasst. Flutter, von Google entwickelt, ermöglicht die Erstellung von sogenannten *Cross-Platform Apps*. Dies bedeutet, dass die App einmal geschrieben werden kann und das Framework sie dann auf verschiedenen Plattformen wie Android, Apple oder Windows lauffähig macht.

Die erstellte App fungiert als Proof of Concept (PoC) für unser entwickeltes Framework. Durch diesen PoC soll die Funktionalität des Frameworks veranschaulicht und erläutert werden. Da es sich bei dieser App um einen PoC handelt, lag der Fokus hauptsächlich in der Entwicklung und nicht auf der Fehlerfreiheit und dem Design der Anwendung.

7.1 App

Beim Starten der Anwendung zeigt die App einen Anmeldebildschirm, auf dem der Nutzer eine E-Mail sowie ein Passwort eingeben kann (7). Bei Bestätigung des Anmeldeknopfes werden diese Daten durch eine Anfrage an den Anmeldeendpunkt der REST-API geschickt. Bei einer erfolgreichen Anmeldung wird der Nutzer in den Hauptbildschirm weitergeleitet und der von der REST-API erzeugte JWT wird für nachfolgende Anfragen gespeichert. In dem Hauptbildschirm zeigt die App alle Gruppen an, die der angemeldete Nutzer einsehen darf (8). Dafür wird ein Endpunkt verwendet, der alle einsehbare Gruppen des Nutzers zurückgibt. Über eine Schaltfläche kann eine neue Gruppe erzeugt werden.

Beim Antippen einer Gruppe wechselt die Oberfläche zu einer Gruppendarstellung (9). In dieser Gruppendarstellung fragt die App mittels der *Gruppen ID* alle Aufgabenpakete dieser Gruppe an und zeigt diese in einer Liste. Durch einen Klick auf ein Aufgabenpaket sieht der Nutzende alle seine Lösungsversuche zu diesem Aufgabenpaket. Zusätzlich kann ein neuer Lösungsversuch erstellt werden. Sollte nun auf einen bereits bestehenden Lösungsversuch geklickt, oder ein neuer erstellt werden, wird auf eine Oberfläche gewechselt, in der die Aufgaben gelistet sind. Der *PoC* App sind zwei Typen von Aufgaben bekannt: „Multiple-Choice“ und „Cocke-Younger-Kasami (CYK)-Algorithmus“. Über einen Abfrageparameter fragt die App nur Aufgaben dieser beiden Typen an.

Will der Nutzer einer der Aufgaben bearbeiten, entscheidet die App anhand des Aufgabentypus welches Darstellungsformat verwendet wird.

- *Multiple-Choice*
  Dem Benutzer wird die Frage zusammen mit allen Antwortmöglichkeiten angezeigt

(13). Bei Klick auf eine Antwortmöglichkeit wird diese als einzige Antwortmöglichkeit markiert. Diese Antwortmöglichkeit wird beim Speichern an die REST-API übergeben.

- *CYK-Algorithmus*
  Um einen aktiven Anwendungsfall in der Lehre der theoretischen Informatik zu zeigen, haben wir beschlossen eine Aufgabe zum Lösen des CYK-Algorithmus einzubauen. In dem Abschnitt 7.2 kann näheres zu dem CYK-Agorithmus gelesen werden. Dem Nutzer wird die kontextfreie Grammatik als auch das zu überprüfende Wort angezeigt. Unter diesen Informationen steht ein rechtwinkliges, gleichschenkliges Dreieck aus Quadraten. Die Länge der Seiten entspricht der Anzahl an Buchstaben des Wortes in Quadraten (14). Über das Antippen eines Quadrates öffnet sich eine Schaltfläche, in der die Variablen der Grammatik dem Quadrat hinzugefügt werden können. Die Nutzer sollen den CYK-Algorithmus per Hand durchführen. Beim Speichern wird der Inhalt der Quadrate in einer List gespeichert und der REST-API mitgeteilt.

Über die Aufgabenliste kann das Aufgabenpaket beendet werden. Beendete Aufgaben können nicht mehr gespeichert werden, stattdessen kann der Benutzer die abgegebene Antwort mit der Lösung vergleichen.

Zurück in der Gruppendarstellung ist ein Schalter ersichtlich, mit dem die Liste der Aufgabenpakete zu der List mit allen Mitglieder der Gruppe wechselt (10). In einem Menü können die Berechtigungen der Mitglieder eingesehen und verändert werden (11). Auch können dort die öffentlichen Lösungsversuche aller Gruppenmitglieder angezeigt werden. Zusammen mit den öffentlichen Lösungsversuchen wird dort ein Graph angezeigt, der die Entwicklung der korrekt gelösten Aufgaben der Lösungsversuche darstellt (12).

Die App zeigt dem Nutzenden nur Aktionen an, die dieser mit seinen Rechten ausführen darf. Um diese Entscheidung zu treffen, wird eine Anfrage an das Framework gesandt, die den Nutzer auf entsprechende Rechte anfragt.

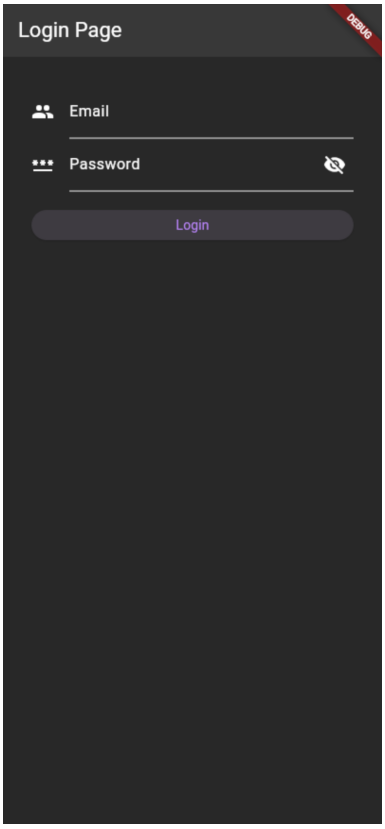

Abbildung 7 App: Anmeldebildschirm

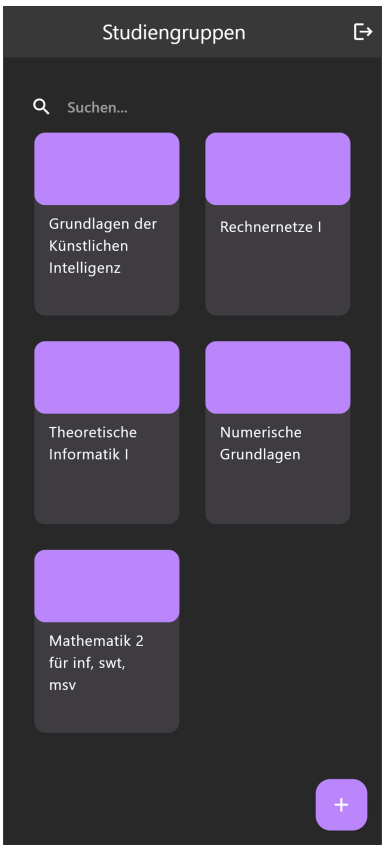

Abbildung 8 App: Hauptbildschirm

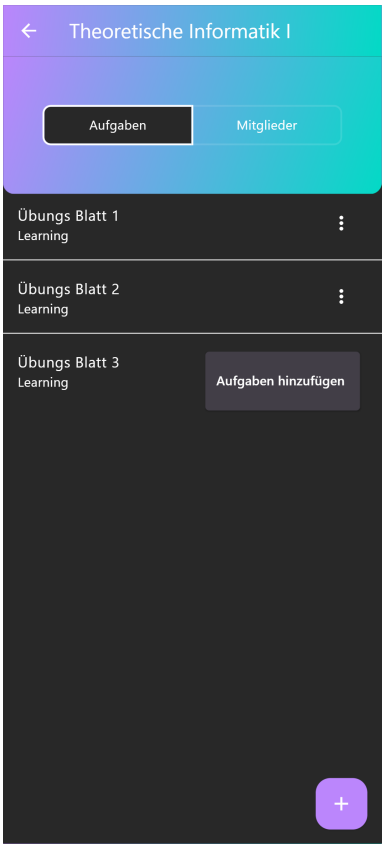

Abbildung 9 App: Gruppenbildschirm

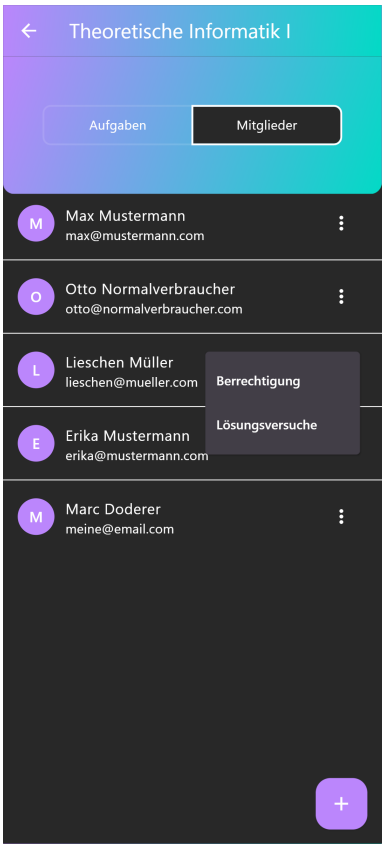

Abbildung 10 App: Mitglieder

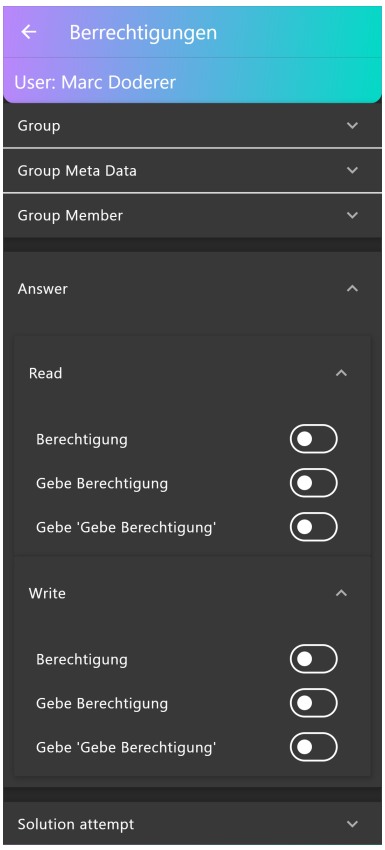

Abbildung 11 App: Berechtigungen

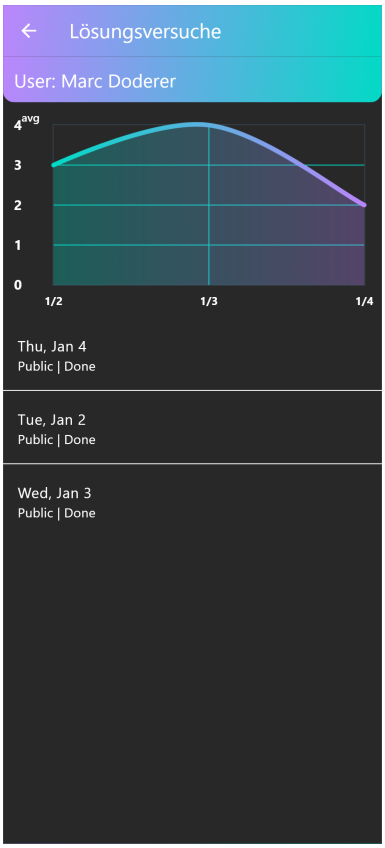

Abbildung 12 App: Statistik über Lösungsversuche

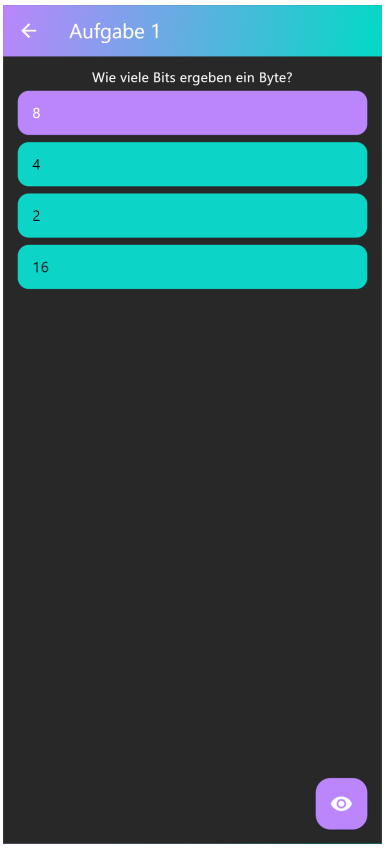

Abbildung 13 App: Multiple Choice

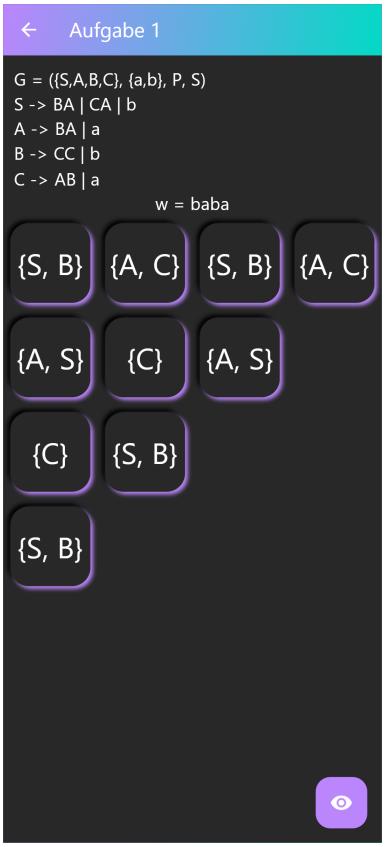

Abbildung 14 App: CYK Algorithmus

7.2 Aufgaben erstellen

Neben dem Framework und der PoC App, haben wir noch ein drittes Programm entwickelt. Beim Starten des Programmes nimmt dieses über die Konsole Eingaben entgegen. Je nach Eingabe wird zusammen mit dem Nutzer eine Kontextfreie Grammatik erstellt oder mittels einer kontextfreien Grammatik in der Chomsky-Normalform (CNF) der CYK-Algorithmus berechnet. In der theoretischen Informatik werden Grammatiken unter anderem benutzt, um die Lösbarkeit von Problemen durch Automaten zu erkennen. Ein Automat ist hierbei eine Maschine, die eine Folge von Zeichen entgegennimmt und bei jedem Zeichen abhängig des aktuellen Zustandes einen Zustandswechsel durchführen kann [37]. Eine Grammatik wird Mathematisch durch ein 4-Tupel beschrieben: $G = (V, T, P, S)$
V eine endliche Menge an Variablen.
T eine endliche Menge an Terminalen, wobei gilt: $V \cap T = \emptyset$.
$P \subset V^* \times (V^* \cup T^*)$ eine Menge an Produktionsregeln.
$S \in V$ das Startsymbol.
Eine von der Grammatik beschrieben Sprache ist die Menge aller nur aus Terminalen bestehenden Wörter, die aus dem Startsymbol durch die Produktionsregeln abgeleitet werden können. Kontextfreie Grammatiken sind Grammatiken, die in allen Produktionsregeln auf der linken Seite nur eine Variable haben. Demnach gilt: $P \subset V \times (V^* \cup T^*)$. Alle von kontextfreie Grammatiken beschriebenen Sprachen können auch durch kontextfreie Grammatiken in der CNF beschrieben werden [38]. Eine solche Grammatik ist in der CNF wenn die Produktionsregeln wie folgt aussehen:

- $V \times V^2$
- $V \times T$
- $V \times \epsilon$ (ϵ beschreibt das *leere Wort*)

Über den CYK-Algorithmus kann überprüft werden, ob sich ein Wort in der von einer kontextfreien Grammatik beschriebenen Sprache befindet. Dabei wird jedoch die kontextfreie Grammatik in der CNF vorausgesetzt.

Das von uns erstelle Programm, kann für kontextfreie Grammatiken in der CNF den CYK-Algorithmus [39] berechnen und in einem Format zurückgeben, sodass die Antwort direkt in unserem Framework als Aufgabendokument hochgeladen werden kann. Es können also durch das Programm schnell und automatisch Aufgaben erstellt werden.

7.2.1 Umsetzung der Erstellung von CYK Aufgaben

Das Programm für das Erstellen der CYK-Aufgaben ist, wie auch das Framework, in der Programmiersprache RUST entwickelt. Der Algorithmus nutzt die Form der Produktionsregeln von kontextfreie Grammatiken in der CNF um herauszufinden, ob die Grammatik

ein bestimmtes Wort bilden kann. Dabei errechnet der Algorithmus für jedes Teilwort des Wortes die angewandten Produktionsregeln. In dem *Code 26 - Programmcode: CYK-Algorithmus* ist ein Ausschnitt des von uns implementierten Algorithmus zu sehen. Die durch den Algorithmus berechnete Tabelle wird anschließend in eine zwei-dimensionale Liste überführt und ausgegeben.

```
let n = w.len();
let mut table: Vec<Vec<HashSet<String>>> = vec![vec![HashSet::new(); n]; n];

for (i, c) in w.chars().enumerate() {
    for rule in &grammar.P {
        if rule.r.iter().any(|right| right.len() == 1 && right[0] == c.to_string()) {
            table[i][0].insert(rule.l.clone());
        }
    }
}

for j in 2..=n {
    for i in 1..=(n + 1 - j) {
        for k in 1..=(j-1) {
            for rule in &grammar.P {
                for right in &rule.r {
                    if right.len() == 2 {
                        let (a, b) = (right[0].clone(), right[1].clone());
                        if table[i - 1][k - 1].contains(&a) && table[i + k - 1][j-k - 1].contains(&b) {
                            table[i - 1][j - 1].insert(rule.l.clone());
                        }
                    }
                }
            }
        }
    }
}
```

Code 26 Programmcode: CYK-Algorithmus

8 Dokumentation und Tests

Um das Framework für nachfolgende Entwickler nachvollziehbar zu machen, haben wir Dokumentationen erstellt. Für die Dokumentation der Endpunkte verwenden wir OpenAPI [40]. OpenAPI ist ein formaler Standard, um Web-APIs zu dokumentieren. Dieser Standard ist so aufgebaut, dass ihn sowohl Menschen als auch Maschinen verstehen. Anhand dieser OpenAPI Dokumentation, hostet das Framework eine interaktive HTML-Webseite, auf der die Endpunkte beschrieben sind. In der *Abbildung 15 - OpenAPI Dokumentation als interaktive Webseite* ist ein Ausschnitt dieser Webseite abgebildet. Dort können wir die beiden Endpunkte: *GET: /api/tasks* und *POST: /api/tasks* erkennen. Der *POST* Endpunkt wurde ausgeklappt, wodurch eine Beschreibung ersichtlich ist, sowie eine Oberfläche um eine Anfrage an die REST-API zu senden.

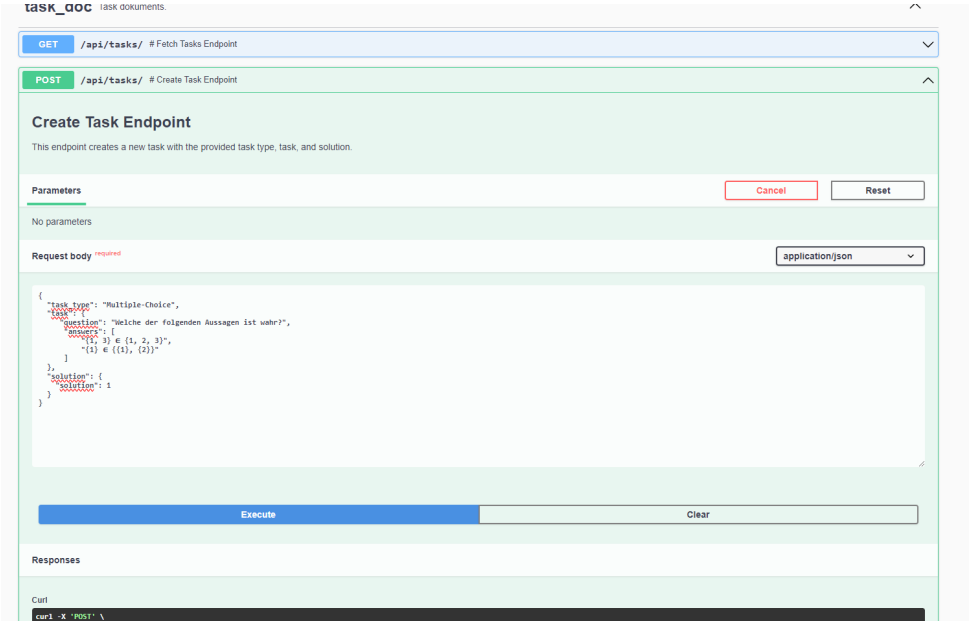

Abbildung 15 OpenAPI Dokumentation als interaktive Webseite

Neben der Dokumentation der Endpunkte haben wir auch Dokumentation für die implementierten Funktionen geschrieben. Um die Erstellung der Dokumentation zu beschleunigen, haben wir diese mithilfe von künstlicher Intelligenz verfassen lassen.

Zusätzlich zu den Dokumentationen haben wir Komponententest geschrieben. Komponententests sind spezielle Prüfungen, bei denen die Funktionalitäten der Funktionen im Detail überprüft werden. Hierbei werden sämtliche möglichen Kombinationen von Eingabewerten simuliert und die darauf folgenden Reaktionen auf ihre Korrektheit überprüft. Diese Tests gewährleisten, dass sich die Funktionen gemäß den Erwartungen verhalten. Auch helfen sie nachfolgenden Entwicklern die beabsichtigten Funktionalitäten der Endpunkte nachvollziehbar zu machen. Wenn wir beispielsweise den Endpunkt zur Anmeldung betrachten, sind folgende Kombinationen zu testen:

Eingabe	erwartetes Ergebnis
korrekte E-Mail-Adresse mit korrektem Passwort	erfolgreiche Anmeldung
korrekte E-Mail-Adresse mit korrektem Passwort (E-Mail-Adresse ohne Beachtung Groß- und Kleinschreibung)	erfolgreiche Anmeldung
korrekte E-Mail-Adressen mit falschem Passwort	Anmeldung abgelehnt
falsche E-Mail-Adressen mit falschem Passwort	Anmeldung abgelehnt

Tabelle 5 Komponententest für den anmelde Endpunkt

9 Diskussion

In diesem Abschnitt interpretieren wir die Funktionalitäten und Umsetzungen des Frameworks. Wir geben ein Einblick in die Entwicklungsphase und welche Schwierigkeiten aufgetreten sind.

9.1 Diskussion zur Authentifizierung

Die implementierte Authentifizierung erfüllt die Aufgabe, Benutzer am Framework auszuweisen. Dies erreichen wir über die Verwendung von JWT. Uns war bei der Entwicklung die Möglichkeit des Anmeldens an der REST-API wichtig, die Sicherheit wurde von uns nicht priorisiert. Die Umsetzung kann separat vom bestehenden Framework verändert und dadurch sicherer gemacht werden, ohne die Funktionen des Frameworks verändern zu müssen.

Zudem könnten noch weitere Änderungen an der Authentifizierung von Nutzern vorgenommen werden. Über ein *Refresh-Token* kann beispielsweise der Umgang mit der Lernplattform für Endnutzer angenehmer gestaltet werden. Ein *Refresh-Token* ist wie der *Access-Token* aufgebaut, hat jedoch eine längere Gültigkeit. Aufgabe dieses Tokens ist es, einen neuen *Access-Token* an der REST-API generieren zu lassen. So muss der Client bei Ablauf des *Access-Tokens* den Nutzer nicht direkt neu anmelden, sondern kann dafür den *Refresh-Token* verwenden. Auch kann über eine Blacklist, ein JWT unbrauchbar gemacht werden, sollte sich ein Nutzer abmelden oder ein *Refresh-Token* verwendet worden sein.

Die Registrierung von Benutzer kann auch angepasst werden. Es kann beispielsweise die Rückbestätigung der E-Mail-Adresse zu Voraussetzung gemacht werden, um das Registrieren zu limitieren. Alternativ könnte auch die Registrierung von Personen nur fremdgesteuert funktionieren, sodass nur bereits registrierte und berechtigte Personen weitere Nutzende im System erstellen können.

9.2 Diskussion zur Benutzerverwaltung

Die Umsetzung der Benutzerverwaltung bietet feingranulare Möglichkeiten, Berechtigungen von Ressource auf die Nutzenden zu übertragen. Durch Rollen können zusammenhängende Ressource gebunden und schablonenartig an die Personen verteilt werden. Ob jedoch bei wachsenden Strukturen der Lernplattform diese Benutzerverwaltung weiterhin verwendet werden sollte, ist offen. Die feingranulare Verteilung von Rechten könnte zu einer zu großen administrative Aufgabe werden. Zusätzlich ist die Verteilung von neuen Berechtigungen umständlich. Neue Berechtigungen müssen allen Personen individuell ver-

teilt werden, da in der aktuellen Umsetzung ein Indikator fehlt, um diese automatisch zu Verteilen. Die Verwendung von Berechtigungen auf Ressourcen Ebene sollte jedoch genug Möglichkeiten bieten, um bei Weiterentwicklung darauf aufzubauen.

Eine denkbare Anpassung wäre es, die Benutzerverwaltung so zu ändern, dass Gruppen Berechtigungen weitervererben. So würden beispielsweise alle Berechtigungen der übergeordneten Gruppe auch auf die Eigene gelten.

Auch unterlegen die Aufgabendokumente momentan keinen individuellen Berechtigungen. Jeder Nutzer, der auf ein Aufgabendokument zugreifen kann, kann auch auf alle anderen zugreifen. Auch die Lösungsdokumente unterliegen diesem Problem. Es benötigt ein Ansatz um die Aufgaben- und Lösungsdokumente feiner zu berechtigen.

9.3 Diskussion zu den Gruppen und Aufgaben

Die verwendeten Umsetzungen der Gruppen und Aufgaben bieten alle grundlegenden Eigenschaften für die Verwendung der Lernplattform. Wir haben uns im Rahmen der Umsetzung großteils darauf konzentriert die wichtigsten Eigenschaften der Datenstrukturen zu implementieren. Für die Individualisierung von Gruppen oder Aufgaben wären weitere beschreibende Eigenschaft wie beispielsweise Beschreibungen oder Bilder ein hilfreicher Zusatz.

Über die Endpunkte der REST-API können zwar Einträge der Datenbank vorläufig gelöscht werden, jedoch werden diese Einträge zum aktuellen Status des Frameworks nicht aus der Datenbank entfernt. Es müsste ein Cronjob hinzugefügt werden, der sich in bestimmten Zeitintervallen alle vorläufig gelöschten Einträge aus der Datenbank anschaut und diese bei Erfüllung bestimmter Kriterien aus der Datenbank entfernt. Ein Cronjob ist ein Prozess der in vorgegebenen Zeitintervallen eine Aufgabe ausführt.

Die Korrektur eines Antwortenddokuments wird durch einen Objektvergleich umgesetzt. Bei möglichen zukünftigen Aufgabentypen könnte es vorkommen, dass diese Art von Korrektur nicht ausreicht. Dies ist beispielsweise der Fall, wenn zu einer Aufgabe mehrere richtige Antworten existieren. Für diese Fälle bräuchte man einen Ansatz, wie die Korrektur der Antworten dynamisch umgesetzt werden kann.

Der Lernerfolg der Nutzer wird an den Lösungsversuchen im Aufgabenpaket ausgewertet. Da jeder Lösungsversuch innerhalb eines Aufgabenpaketes die gleichen Aufgaben besitzt, kann nicht zwischen: *Der Nutzer hat sich alle bereits gelösten Aufgaben auswendig gemerkt* oder *Der Nutzer hat die Aufgaben verstanden* unterschieden werden. Ein möglicher Lösungsansatz wäre, die Aufgaben in dem Aufgabenpaket durch Aufgabenpools zu ersetzen.

Ein Lösungsversuch zieht sich dann zufällig aus jedem Aufgabenpool eine Aufgabe. So würden sich Lösungsversuche innerhalb des Aufgabenpaketes auf unterschiedliche Aufgaben beziehen und der Lernerfolg könnte besser ausgewertet werden.

9.4 Problem in der Entwicklung

In der Entwicklung des Frameworks entstanden Probleme, die die Entwicklungszeit verlängerten. Viele dieser Probleme konnten im Laufe der Entwicklung beseitigt werden, einzelne jedoch nicht.

Da die Programmiersprache Rust noch nicht lange am Markt ist, ist die im Internet zu findende Informationsbasis geringer als die anderer etablierter Programmiersprachen. In Kombination mit weniger bekannten OSS, entstanden teilweise Fehler bei deren Beseitigung wir uns schwertaten. Insbesondere die Pakete *utoipa* und *Diesel* sind uns negativ hier aufgefallen.

Wir haben das Paket *utoipa* verwendet, um eine OpenAPI Dokumentation für unsere Endpunkte zu schreiben. *Utoipa* ist zum Zeitpunktes der Arbeit das verbreitetste Paket, für OpenAPI Dokumentation unter Rust. Fehlende Dokumentation und nicht hilfreiche Fehlermeldungen haben uns jedoch das Auffinden von Fehlern und das Schreiben von richtiger Syntax erschwert. Da diese OpenAPI-Dokumentation jedoch nicht in die Logik des Programmes eingreift, sollte, sobald eine bessere Alternative gefunden wird, das Ersetzen dieses Paketes einen geringen Mehraufwand darstellen.

Anders sieht es mit dem Paket *Diesel* aus. Wie in dem *Abschnitt 2.5.2 - Diesel* erläutert, ist *Diesel* ein ORM Werkzeug und hat aufgrund dessen eine hohe Präsenz im Code. Wir haben beispielsweise keine einfache Lösung gefunden SELECT-Anfragen dynamisch zu ordnen. Das Problem hätte durch *Raw-SQL* umgangen werden können, dies ist jedoch nicht der Sinn des ORM-Werkzeugs und war für unsere Arbeit in der vorhandenen nicht relevant genug. Zu Zeit des Artikels haben Features wie *JOINS* in *UPDATE* Operationen gefehlt [41], wodurch mehr Anfragen verwendet werden mussten als grundsätzlich nötig waren. Unter anderem wegen dieser Eigenschaften ist uns das Einarbeiten in das Paket schwergefallen, dies ist jedoch nicht nur diesem Paket verschuldet, sondern liegt auch in der Natur von ORM-Werkzeugen allgemein.

Neben den Schwierigkeiten in der Entwicklung ist uns auch ein Problem beim Testen der Endpunkte aufgefallen. Das Ausführen zu vieler gleichzeitiger Tests erzeugt beim Zugriff auf die Datenbank einen Fehler. Ob dieses Problem exklusiv nur beim Testen auftritt, oder auch im Betrieb auftreten kann, haben wir nicht evaluieren können.

10 Fazit

Wir haben ein Framework entwickelt, das die grundlegenden Funktionen einer Lernplattform abbildet. Über eine REST-API werden Schnittstellen bereitgestellt, die mit einer Benutzerverwaltung Zugriffe auf eine Datenbank ermöglicht. Benutzer können auf Gruppen zugeordnet werden, um Aufgaben zu lösen. Jedem Nutzer können die Rechte innerhalb dieser Gruppen individuell zugewiesen werden. Aufgaben werden in sogenannten Aufgabenpaketen verteilt und Benutzer können innerhalb dieser Pakete, Lösungsversuche starten und Aufgaben lösen.

Zur Absicherung der Funktionen der Endpunkte haben wir umfangreiche Komponententests durchgeführt.

Die Dokumentation unserer API haben wir mithilfe von OSS umgesetzt und eine OpenAPI Dokumentation erstellt. Diese beschreibt jeden Endpunkt ausführlich und ist als interaktive Webseite unter dem Pfad „/swagger-ui/“ gehostet. Eine PoC App zeigt einen möglichen Verwendungszweck dieses Frameworks.

Die App nutzt die Endpunkte der REST-API, um die Ressourcen des Frameworks interaktiv darzustellen. Wir haben mit den Aufgabentypen „Multiple-Choice“ und „CYK-Algorithmus“ zwei Aufgaben implementiert, die zeigen sollen, wie das Lernen mit dem Framework in der theoretischen Informatik angewandt werden kann.

Durch eine ausführliche Dokumentation der meisten Funktionen soll nachfolgenden Entwicklern ein einfacher Einstieg in das Projekt ermöglicht werden.

Für das Verwenden der Lernplattform in einer Produktivumgebung, sollten dennoch Anpassungen und Erweiterungen am Framework vorgenommen werden. Die Authentifizierung und Registrierung der Nutzer wäre noch zu optimieren. Cronjobs, die vorläufig gelöschte Objekte aus der Datenbank entfernen, sollten dem Framework hinzugeführt werden. Es können auch noch weitere Tests durchgeführt werden, um beispielsweise die Performance und Stabilität unter großen Anfragemengen zu gewährleisten.

Quellenverzeichnis

[1] KEM, Deepak, „Personalised and adaptive learning: Emerging learning platforms in the era of digital and smart learning," *International Journal of Social Science and Human Research*, Jg. 5, Nr. 2, S. 385–391, 2022

[2] Marc Doderer: Framework-e-learning-platform, https://github.com/MarcDod/Framework-e-learning-platform, 2024

[3] The Rust Team. „Rust." (15. Dez. 2023a), In: https://www.rust-lang.org/ (15. 12. 2023)

[4] TEAM, The Rust Core, „Announcing Rust 1.0," *Rust Blog*, 2015

[5] The Actix Team. „Actix." (15. Dez. 2023a), In: https://actix.rs/ (15. 12. 2023)

[6] BUGDEN, William; ALAHMAR, Ayman, „Rust: The programming language for safety and performance," *arXiv preprint arXiv:2206.05503*, 2022

[7] MATSAKIS, Nicholas D; KLOCK, Felix S, „The rust language," *ACM SIGAda Ada Letters*, Jg. 34, Nr. 3, S. 103–104, 2014

[8] XU, Baowen; u. a.; CHU, Bei; FAN, Hongcheng: „An Analysis of the Rust Programming Practice for Memory Safety Assurance," in *Web Information Systems and Applications*, L. Yuan; S. Yang; R. Li; u. a., Hrsg., Singapore: Springer Nature Singapore, 2023, S. 440–451

[9] University of Washington. „CSE 341 – Lexical and Dynamic Scoping." (15. Dez. 2023), In: https://courses.cs.washington.edu/courses/cse341/14wi/general-concepts/scoping.html (15. 12. 2023)

[10] MASSE, Mark: REST API design rulebook: designing consistent RESTful web service interfaces. O'Reilly Media, Inc., 2011

[11] BIEHL, Matthias: API Architecture. API-University Press, 2015, Bd. 2

[12] RICHARDSON, Leonard; RUBY, Sam: RESTful web services. O'Reilly Media, Inc., 2008

[13] Was ist eine REST-API? https://www.ibm.com/de-de/topics/rest-apis, Accessed: 2023-12-07

[14] gRPC Authors. „gRPC.“ (15. Dez. 2023), In: https://grpc.io/ (15. 12. 2023)

[15] IBM. „Was sind Microservices?“ (15. Dez. 2023), In: https://www.ibm.com/de-de/topics/microservices (15. 12. 2023)

[16] JONES, M.; BRADLEY, J.; SAKIMURA, N.: RFC 7519: JSON Web Token (JWT), USA, 2015

[17] Introduction to JSON Web Tokens, https://jwt.io/introduction, Accessed: 2023-12-12

[18] MAHINDRAKA, P, „Insights of JSON Web Token,“ *International International Journal of Recent Technology and Engineering (IJRTE) ISSN*, S. 2277–3878, 2020

[19] Oracle. „Was ist eine Datenbank?“ (18. Dez. 2023), In: https://www.oracle.com/de/database/what-is-database/ (18. 12. 2023)

[20] MCHUGH, Jason; u. a.; ABITEBOUL, Serge; GOLDMAN, Roy, „Lore: A Database Management System for Semistructured Data,“ *SIGMOD Rec.*, Jg. 26, Nr. 3, S. 54–66, Sep. 1997. DOI: 10.1145/262762.262770. In: https://doi.org/10.1145/262762.262770

[21] STONEBRAKER, Michael; KEMNITZ, Greg, „The POSTGRES next generation database management system,“ *Communications of the ACM*, Jg. 34, Nr. 10, S. 78–92, 1991

[22] DAYAL, Umeshwar: „Active database management systems,“ in *Proceedings of the Third International Conference on Data and Knowledge Bases*, Elsevier, 1988, S. 150–169

[23] CODD, Edgar F: „Relational database: A practical foundation for productivity,“ in *ACM Turing award lectures*, 2007, S. 1981

[24] CODD, E. F., „A Relational Model of Data for Large Shared Data Banks,“ *Commun. ACM*, Jg. 13, Nr. 6, S. 377–387, Juni 1970. DOI: 10.1145/362384.362685. In: https://doi.org/10.1145/362384.362685

[25] SONG, Il-Yeol; EVANS, Mary; PARK, Eun K, „A comparative analysis of entity-relationship diagrams,“ *Journal of Computer and Software Engineering*, Jg. 3, Nr. 4, S. 427–459, 1995

[26] The MIT License, https://opensource.org/license/mit/, Accessed: 2023-12-07

[27] Apache License, Version 2.0, https://www.apache.org/licenses/LICENSE-2.0, Accessed: 2023-12-07

[28] The Actix Team. „Actix documentation.“ (16. Dez. 2023b), In: https://actix.rs/docs (16. 12. 2023)

[29] Sergio Benitez. „Rocket.“ (16. Dez. 2023), In: https://rocket.rs/ (16. 12. 2023)

[30] The Diesel Core Team. „Diesel is a Safe, Extensible ORM and Query Builder for Rust.“ (4. Jan. 2023), In: https://diesel.rs/ (04. 01. 2023)

[31] KWAME, Ziddah Edem: Rust – JWT Authentication with Actix Web, 2023

[32] TEAM, The Actix, „Actix Documentation - Middleware,“ *actix*, 2023

[33] The Rust Team. „Makros.“ (19. Dez. 2023b), In: https://rust-lang-de.github.io/rustbook-de/ch19-06-macros.html (19. 12. 2023)

[34] The Rust Team. „Storing Keys with Associated Values in Hash Maps.“ (21. Dez. 2023c), In: https://doc.rust-lang.org/book/ch08-03-hash-maps.html?highlight=HashMap#creating-a-new-hash-map (21. 12. 2023)

[35] JSON Schema. „JSON Schema.“ (29. Dez. 2023), In: https://json-schema.org/ (29. 12. 2023)

[36] Flutter. „Flutter.“ (27. Dez. 2023), In: https://flutter.dev/ (27. 12. 2023)

[37] HOPCROFT, John E; MOTWANI, Rajeev; ULLMAN, Jeffrey D, „Introduction to automata theory, languages, and computation,“ *Acm Sigact News*, Jg. 32, Nr. 1, S. 60–65, 2001

[38] HOTZ, Günter, „Normal-form transformations of context-free grammars,“ *Acta Cybernetica*, Jg. 4, Nr. 1, S. 65–84, 1978

[39] YOUNGER, Daniel H., „Recognition and parsing of context-free languages in time n3,“ *Information and Control*, Jg. 10, Nr. 2, S. 189–208, 1967. DOI: https://doi.org/10.1016/S0019-9958(67)80007-X. In: https://www.sciencedirect.com/science/article/pii/S001999586780007X

[40] The Linux Foundation. „The world's most widely used API description standard.“ (31. Dez. 2023), In: https://www.openapis.org/ (31. 12. 2023)

[41] csirkeee. „Allow update/delete with joined tables.“ (29. Dez. 2023), In: https://github.com/diesel-rs/diesel/issues/1478 (29. 12. 2023)

Erklärung:

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Stuttgart

Ort, Datum	Unterschrift