UniAufgaben

Eine Webapplikation zur Verwaltung der eigenen Aufgaben im Studium

Bachelorarbeit

Kilian Maendly

Januar 2024

Unter der Aufsicht von:

Prof. Dr. Jacques Pasquier-Rocha
Software Engineering Group

UNI FR
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Software Engineering Gruppe
Department für Informatik
Universität Freiburg (Schweiz)

Software Engineering Group

Danksagung

Ich möchte mich bei Herrn Prof. Dr. Jacques Pasquier-Rocha bedanken für die Aufsicht und die wertvolle Unterstützung, welche diese Arbeit ermöglicht haben.

Zudem möchte ich mich bei Herrn Bernhard Kopp und Frau Lisa Maria Marti für das Testen des Programms und das Teilen ihrer Meinung bedanken.

Abstrakt

Diese Arbeit befasst sich mit dem Ziel, eine Webapplikation zu erstellen, mit welcher Personen ihre Aufgaben im Studium auf einfache Weise organisieren können. Für die Erstellung der Applikation wurde der MEAN-Stapel verwendet, welches sich aus den Technologien MongoDB, Express.js, Angular und Node.js zusammensetzt. Das Resultat der Arbeit ist, dass es möglich war, die Applikation zu entwickeln, so dass die Ziele erreicht werden konnten. Als Verbesserungsmöglichkeiten wurde festgehalten, dass die Applikation mit einem Administratorkonto und einem System zum Zurücksetzten von Passwörtern, sowie einer Ansicht für Mobilgeräte erweitert werden könnte. Der Bericht besteht aus einer Einführung, Problembeschreibung, Demonstration und Implementation des Programms, sowie einer Schlussfolgerung.

Schlüsselwörter: MEAN-Stack, MongoDB, Express.js, Angular, Node.js, Webapplikation, HTTP

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einführung

1.1. Motivation und Ziele dieser Arbeit

In manchen Kursen wird von den Studenten verlangt, dass sie nach den Vorlesungen Übungen lösen, häufig in Form von wöchentlichen Serien. Abhängig vom Kurs kann es sein, dass das Bestehen dieser Serien Bedingung ist, um am Ende des Semesters zur Prüfung zugelassen zu werden. Besucht ein Student jede Woche mehrere Kurse, so ist es nicht abwegig, eine Liste mit den zu bearbeitenden Aufgaben zu führen, um zu vermeiden, dass Serien vergessen werden. Dem Studenten stehen viele Optionen zur Verfügung, um die Serien zu organisieren. Folgend sind einige Beispiele:

- Ein gewöhnlicher Kalender, wie zum Beispiel jener, welcher auf dem Smartphone des Studenten vorinstalliert ist, um die Fälligkeiten der Serien zu verwalten.
- Eine Applikation zum Speichern von Aufgabenlisten, damit offene von bereits fertiggestellten Aufgaben unterschieden werden können.
- Microsoft Excel zur Berechnung der Anzahl bestandener Aufgaben, bzw. der noch benötigten Menge an zu bestehenden Aufgaben.

Allerdings besteht das Problem, dass keines dieser Tools alle benötigten Funktionen auf einmal bereitstellen kann. Zwar ist es möglich, sie zu kombinieren, allerdings ist es umständlich die Organisation der Aufgabenserien auf mehreren Programmen aufzuteilen. Das Ziel dieser Arbeit ist deshalb, ein Programm zu erstellen, welches folgende Funktionen anbieten soll:

– Die Verwaltung von Kursen und, falls zutreffend, der Darstellung des Standes der Bewertung der Serien.
– Die Auflistung von Aufgaben nach Kurs, sowie Organisation der Fälligkeit und Stand der Serien.
– Die Möglichkeit, Filter anzuwenden und die Aufgaben zu sortieren.

Zudem soll das Programm mehreren Personen zur Verfügung stehen können.

1.2. Strukturierung des Berichts

Kapitel 1: Einführung

Das aktuelle Kapitel. In der Einführung werden Motivation und Ziele dieser Arbeit geschildert und die Struktur des Rapports beschrieben, sowie Notation und Konventionen definiert.

Kapitel 2: Vorhaben

Dieses Kapitel befasst sich mit der Planung des Programms. Mit Anwendungsfällen werden die benötigten Funktionen beschrieben, welche dem Nutzer zur Verfügung stehen müssen. Die Struktur der Datenbank wird mit einem ER-Diagramm erklärt.

Kapitel 3: Vorstellung des Programms

Ziel dieses Kapitels ist, das Programm zu präsentieren, bevor die eigentliche Implementation vorgestellt wird. Anhand von Bildschirmaufnahmen wird beschrieben, wie ein Nutzer das Programm verwenden kann.

Kapitel 4: Implementation

Dieses Kapitel erklärt die Implementation des Frontend, welches die Benutzeroberfläche ist, welche dem Nutzer angezeigt wird, wenn das Programm im Webbrowser geöffnet wird, sowie die Implementation des Backend, welches verantwortlich ist für Anfragen an die Datenbank.

Kapitel 5: Schlussfolgerungen

Im letzten Kapitel wird berichtet, was das Resultat der Arbeit ist, ob die Ziele erreicht wurden und was verbessert werden kann.

Anhang

Der Anhang beinhaltet das Literaturverzeichnis, Referenzen zu Webressourcen, welche in dieser Arbeit genutzten wurden, sowie ein Verweis zur GitHub-Seite des Programms.

1.3. Notationen und Konventionen

- Die Arbeit besteht aus fünf Kapiteln. Jedes Kapitel ist jeweils in Sektionen (und Untersektionen) unterteilt.
- Ausschnitte aus Quelltexten (bzw. Listings) werden wie folgt dargestellt:

```
function division(x:number, y:number){
    var result:number;
    result = x / y;
    return result;
}
```

- Für Webadressen wird die Formatierung wie `http://localhost:3000/` verwendet.

- Abbildungen, Tabellen und Ausschnitte aus Quelltexten werden aufsteigend nummeriert.
- Der einfachheitshalber wird jeweils nur ein Geschlecht verwendet, gemeint sind aber alle Geschlechter. Alle Geschlechter sind gleichwertig.

2
Vorhaben

Dieses Kapitel beschäftigt sich mit dem Vorhaben, d.h. der konkreten Beschreibung, was das Endprodukt anbieten muss, und wie die Informationen in der Datenbank gespeichert werden.

2.1. Problembeschreibung

In einem Studium ist es vorteilhaft, gut organisiert zu sein. Gerade für Kurse, welche häufig Übungen zur Verfügung stellen, ist es ratsam, für die Aufgaben eine Art Agenda zu führen. Heutzutage existieren viele verschiedene Applikationen, welche die Organisation der Aufgaben vereinfachen können. So kann die vorinstallierte Kalender-App auf den meisten Smartphone-Geräten es ermöglichen, die Fälligkeiten der Serien aufzulisten und Erinnerungen zu setzten, allerdings ist es nicht gleichermassen einfach, die Aufgaben nach Kursen zu sortieren oder die Bewertungen aktuell zu halten. Eine ToDo-Listen-App erlaubt es, Aufgaben nach Kurs aufzulisten, kann aber nicht verwendet werden, um die Bewertungen der Aufgaben zu speichern. Um einen Überblick über den Stand der Bewertungen zu erschaffen, kann eine Excel-Tabelle erstellt werden, welche die aktuelle Anzahl bestandener Serien beinhaltet und mit der gesamten Anzahl an Serien vergleicht. Das Eintragen der Fälligkeit, sowie Sortieren und Filtern wären ebenfalls möglich, jedoch wäre diese Tabelle nicht geeignet, um die Aufgaben und Kurse von mehreren Personen zu verwalten.

Das Ziel dieser Arbeit war deswegen, wie in der Einführung bereits erwähnt, eine Applikation zu entwickeln, welche mehreren Personen es erlaubt, ihre Kurse und Aufgaben zu verwalten. Zur Lösung des Problems wurde eine Webapplikation mithilfe des MEAN-Stapels

entwickelt. Genaueres zur Implementation wird in Kapitel 4 vorgestellt. Der Umfang an Funktionen, welche diese Software anbieten muss, wird in der Sektion 2.2 beschrieben und die Datenstruktur der Datenbank in der Sektion 2.3 aufgeführt. In der folgenden Untersektion 2.1.1 werden, zum besseren Verständnis, einige mögliche Kurse als Beispiele vorgestellt.

2.1.1. Beispiele

Universitäten bieten in der Regel eine breite Auswahl an Studiengängen und Kurse. Jeder Kurs hat unterschiedliche Anforderungen an die studierenden Personen, weswegen auch die Übungen unterschiedlich gestaltet sein können. Einige Beispiele aus dem Informatikstudium des Autors dieser Arbeit sind:

- Kurs A: Um die Zulassung zur Prüfung zu erhalten, müssen mindestens 9 von 12 Serien bestanden werden.
- Kurs B: Die Aufgaben sind auf 4 Blöcke unterteilt, wobei in jedem Block eine bestimmte Anzahl an Punkten erreicht werden muss, damit die Prüfung angetreten werden kann.
- Kurs C: Die Aufgaben sind optional, allerdings werden diese bei rechtzeitiger Abgabe korrigiert und mit Feedback zurückgegeben.
- Kurs D: Auch in diesem Kurs sind die Aufgaben optional. Bei erfolgreicher Bearbeitung von mindestens 4 von 5 Serien wird jedoch der Note, welche in der Prüfung erzielt wird, ein Bonus von +0.5 angerechnet.

Bereits anhand der vier gezeigten Beispiele ist erkenntlich, dass das Programm nicht alle Fälle einschliessen kann, die existieren, einerseits da nicht alle Fälle bekannt sind und andererseits, weil dadurch die Komplexität des Programms erheblich zunehmen würde. Stattdessen beschränkt sich die Software auf den in der Sektion 2.2 definierten Funktionsumfang.

2.2. Anwendungsfälle

Um festzulegen, welche Funktionen das Programm zur Verfügung stellen muss, können Anwendungsfälle definiert werden. Anwendungsfalldiagramme, bzw. Use-Case-Diagramme, sind Teil der Unified Modeling Language (UML) und beschreiben, wie ein Akteur mit dem Programm interagieren kann. Elemente dieser Diagramme sind [31]:

- **System** (Viereck): Stellt das Programm dar, welches erstellt werden und die Anwendungsfälle beinhalten soll.
- **Akteur** (Person): Anwender, welcher mit dem Programm, bzw. den Anwendungsfällen interagiert.
- **Anwendungsfall** (Ellipsen): Spezifische Funktion des Programms, welche vom Akteur oder auch im Zusammenhang mit anderen Anwendungsfällen aufgerufen wird.
- **Assoziation** (Verbindungslinien): Darstellung des Zusammenhangs zwischen den verbundenen Elementen. Assoziationen zwischen Use-Cases werden mit gestrichelten Linien dargestellt.

In den Untersektionen 2.2.1 bis 2.2.3 wird der benötigte Funktionsumfang des Programms anhand von Anwendungsfalldiagrammen beschrieben.

2.2.1. Authentifizierung und Kontoverwaltung

Jede Person, welche das Programm nutzen möchte, muss ein Konto besitzen und authentifiziert sein. Aufgaben und Kurse werden dem Nutzer zugewiesen, welcher sie erstellt hat, und können auch nur von diesem aufgerufen, bearbeitet und gelöscht werden.

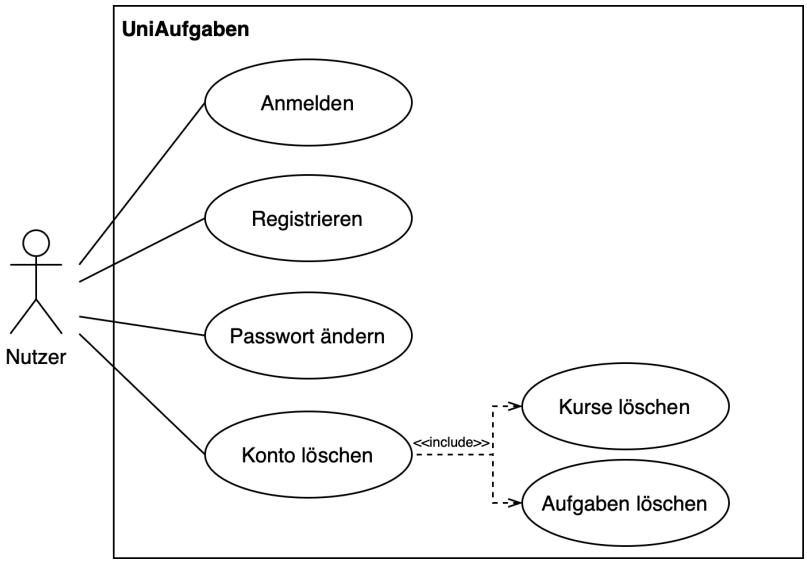

Abbildung 2.1.: Use-Case-Diagramm für Konten

Die Abbildung 2.1 zeigt ein Anwendungsfalldiagramm, bei welchem ein Nutzer sich authentifizieren, bzw. das eigene Konto verwalten kann. Ist ein Nutzer nicht angemeldet und versucht auf eine Seite zuzugreifen, welche den angemeldeten Zustand voraussetzt, leitet das Programm den Nutzer zur Anmeldeseite weiter und fordert ihn dort zum *Anmelden* auf. Nach erfolgreicher Eingabe des Nutzernamens und des dazugehörigen Passwortes, kann der Nutzer auf das Konto zugreifen. Wird ein falsches Passwort und/oder ein falscher Nutzername eingegeben, lehnt das Programm hingegen die Anmeldung ab.

Besitzt eine Person noch kein Konto, so muss sie sich *registrieren*. Für die Registrierung ist ein Nutzername, welches nicht bereits verwendet wird, und ein Passwort notwendig. Versucht die Person, einen Nutzernamen zu verwenden, welcher bereits in der Datenbank existiert, lehnt die Software die Registrierung ab. Wenn der Nutzer ein Passwort wählt, welches den Bedingungen nicht entspricht, oder stimmt das Passwort im Bestätigungsfeld nicht überein, kann das Formular nicht abgesendet werden.

Eine angemeldete Person kann auf der Kontoseite ihr eigenes *Passwort ändern*. Zur Änderung des Passworts ist das alte Passwort notwendig, d.h. das Programm akzeptiert die Änderung des Passworts nur dann, wenn das alte Passwort korrekt eingegeben wurde. Das neue Passwort muss ausserdem auch die gleichen Bedingungen erfüllen wie das alte Passwort. Ist dies nicht der Fall, kann der Nutzer die Änderung nicht speichern.

Benötigt ein Nutzer sein Konto nicht mehr, ist es möglich, es zu *löschen*. Nach korrekter Eingabe des Kontopasswortes werden alle Kurse und Aufgaben sowie das Konto selbst gelöscht, anschliessend wird der Nutzer zur Anmeldeseite zurückgebracht. Die Eingabe eines falschen Passwortes führt dazu, dass das Programm die Löschung nicht durchführt.

2.2.2. Kursverwaltung

Ein Kurs besteht aus einer Sammlung an Vorlesungen. Beispielsweise können Vorlesungen vom Kurs A aus Sektion 2.2 jeden Montag im Semester stattfinden. Ist die Bewertung der Serien von Relevanz, kann angegeben werden, wie viele Serien insgesamt im Kurs existieren und wie viele davon bestanden werden müssen.

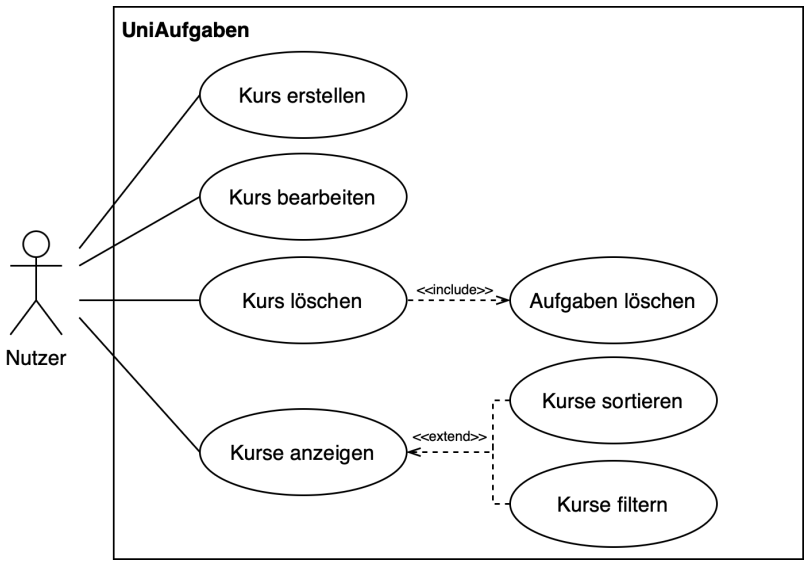

Abbildung 2.2.: Use-Case-Diagramm für Kurse

Auf der Kursseite wird einem angemeldeten Nutzer eine Liste der mit dem Konto erstellten *Kurse angezeigt*. In dieser Liste können die *Kurse nach Kategorie sortiert* und *gefiltert* werden. Ist der Nutzer nicht angemeldet, wird er zu Anmeldeseite gebracht.

Jede Aufgabe muss einem Kurs angehören. Ein Nutzer kann auf der Kursseite einen *Kurs erstellen*. Das Formular verlangt einen Namen, das Semester und das Fach, welchem der Kurs angehört. Werden die Serien bewertet ist zusätzlich die gesamte Menge an Serien und die Menge an zu bestehenden Serien anzugeben. Wurde das Formular nicht korrekt ausgefüllt, kann der Nutzer den Kurs nicht speichern.

Der Nutzer kann auch nachträglich einen *Kurs bearbeiten*. Wählt die Person einen Kurs zur Bearbeitung aus, wird das gleiche Formular wie bei der Erstellung von Kursen geöffnet und mit den vorhandenen Informationen gefüllt. Es gelten bei der Bearbeitung die gleichen Bedingungen wie bei der Erstellung, d.h. der Nutzer kann das Formular erst dann speichern, wenn es korrekt ausgefüllt wurde. Zudem ist es einem Nutzer nicht möglich, einen Kurs zu bearbeiten, welcher ihm nicht gehört. In diesem Fall wird eine Fehlernachricht angezeigt, da der Kurs für diesen Nutzer nicht auffindbar ist.

Wird ein Kurs nicht mehr benötigt, kann der Nutzer den *Kurs löschen*. Das Löschen des Kurses hat nicht nur zur Folge, dass der Kurs selbst aus der Datenbank entfernt wird, auch die darin enthaltenen Aufgaben werden aus der Datenbank gelöscht.

2.2.3. Aufgabenverwaltung

Eine Aufgabe (bzw. Serie) besteht aus einem Set an Übungen, welche vom Studenten bearbeitet werden. Beispielsweise müssen für den Kurs A aus Sektion 2.3 die Serien bis zum Ende der Woche, in welcher sie zur Verfügung gestellt wurden, bearbeitet und abgegeben werden.

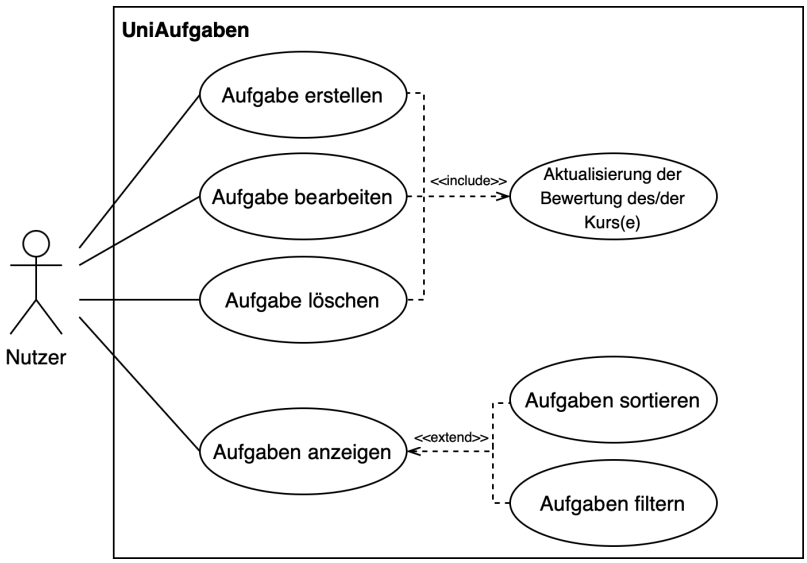

Abbildung 2.3.: Use-Case-Diagramm der Aufgaben

Abhängig davon, ob ein Kurs gewählt wurde oder nicht, werden dem Nutzer auf der Aufgabenseite *Aufgaben angezeigt* von einem oder von allen Kursen, in Form einer Liste, in welcher die *Aufgaben sortiert* und *gefiltert* werden können. Es ist nur möglich die Aufgaben von Kursen abzurufen, welche dem Konto angehören. Versucht ein Nutzer die Aufgaben von einem Kurs abzurufen, welche nicht ihm gehören, kann die Software die Aufgaben, bzw. den Kurs nicht finden und dem Nutzer wird eine Fehlernachricht gezeigt.

Auf der Aufgabenseite kann ein Nutzer eine neue *Aufgabe erstellen*. Um eine Aufgabe zu erstellen, ist es nötig anzugeben, welchem Kurs sie angehört, welche Seriennummer sie hat, bis wann sie erledigt sein muss, sowie welchen Status sie hat (d.h. offen, abgegeben, bestanden oder nicht bestanden). Die Aufgabe kann nur dann gespeichert werden, wenn alle Eingabefelder korrekt ausgefüllt wurden.

Es ist nötig, dass der Nutzer auch *Aufgaben bearbeiten* kann, da z.B. sich der Status der Aufgabe ändern kann. Bei der Bearbeitung wird das gleiche Formular wie bei der Erstellung geöffnet, ausser dass die Felder mit den bereits vorhandenen Daten ausgestattet werden. Es gelten hierbei die gleichen Bedingungen wie bei der Erstellung einer Aufgabe: Gespeichert werden kann erst dann, wenn alle Eingabefelder korrekt ausgefüllt sind.

Ein Nutzer kann auch *Aufgaben löschen*, indem er auf die entsprechende Schaltfläche klickt und den Löschvorgang bestätigt. Die Aufgabe wird dann aus der Datenbank entfernt und die Aufgabenliste wird aktualisiert.

Sollte eine Aufgabe erstellt, bearbeitet oder gelöscht werden, so ist es nötig, dass die Bewertung des Kurses aktualisiert wird, da sich die Menge an bestandenen und/oder nicht bestandenen Serien verändert haben könnte.

2.3. Datenbank

Um Nutzer, Kurse und Aufgaben speichern und abrufen zu können, ist eine Datenbank notwendig. Die Struktur einer Datenbank kann mit einem Entity-Relationship-Diagramm (bzw. ER-Diagramm) dargestellt werden. Ein ER-Diagramm besteht unter anderem aus:

- **Entität** (Rechtecke): Stellt Personen, Objekte oder Konzepte dar.
- **Relation** (Rauten): Beschreibt und zeigt den Zusammenhang zwischen Entitäten, jeweils mit Verbindungslinien verbunden.
- **Kardinalität** (Ziffern und Buchstaben neben Verbindungslinien): Gibt die Menge an Entitäten an, mit welcher die Entität in der jeweiligen Relation steht.

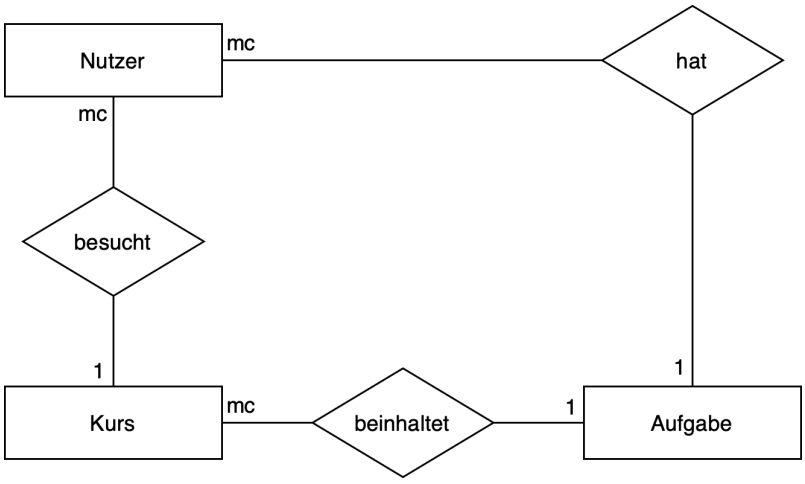

Abbildung 2.4.: ER-Diagramm der Datenbank

Abbildung 2.4 zeigt ein ER-Diagramm mit der modifizierten Chen-Notation [1]. Aus dem Diagramm lässt sich folgendes schliessen:

- Ein Nutzer besucht 0, 1 oder mehr Kurse. Jeder Kurs gehört genau einem Nutzer an.
- Ein Nutzer hat 0, 1 oder mehr Aufgaben. Jede Aufgabe gehört genau einem Nutzer an.
- Ein Kurs beinhaltet 0, 1 oder mehr Aufgaben. Jede Aufgabe gehört genau einem Kurs an.

Für die Entitäten wurden folgende Attribute gewählt:

Nutzer:

- **Nutzername** (Zeichenkette): Der Name, welcher vom Nutzer gewählt wird, und vom Nutzer verwendet wird, um sich anzumelden. Jeder Nutzername darf deswegen nur einmal in der Datenbank vorkommen, da sich sonst Nutzer mit demselben Nutzernamen nicht mehr anmelden könnten.
- **Passwort** (Zeichenkette): Das Passwort, welches vom Nutzer gewählt wird, um sein Konto zu sichern. Dieses wird in verschlüsselter Form in der Datenbank gesichert.
- **ID** (Zeichenkette): Der eindeutige Identifikator, welcher von der Datenbank gesetzt wird und für jeden Nutzer unterschiedlich ist.

Kurs:

- **Name** (Zeichenkette): Der Name des Kurses, welcher ins Programm eingetragen wird.
- **Semester** (Nummer): Das Semester, in welchem der Kurs stattfindet.
- **Fach** (Zeichenkette): Das Fach, welchem der Kurs angehört (bspw. Informatik).
- **Bewertet** (Boolean): Trägt einen `true`- oder `false`-Wert, welcher bestimmt, ob die Serien des Kurses bewertet werden oder nicht.
- **Serien insgesamt** (Zahl): Bestimmt die gesamte Anzahl an Serien im Kurs.
- **Serien benötigt** (Zahl): Bestimmt die benötigte Anzahl an Serien, welche bestanden werden müssen, z.B. um zur Prüfung zugelassen zu werden.
- **Serien bestanden** (Zahl): Die Anzahl bestandener Serien.
- **Serien nicht bestanden** (Zeichenkette): Die Anzahl nicht bestandener Serien.
- **Nutzer** (Zeichenkette): Die ID des Nutzers, welcher den Kurs erstellt hat.
- **ID** (Zeichenkette): Der eindeutige Identifikator, welcher von der Datenbank gesetzt wird und für jeden Kurs unterschiedlich ist.

Aufgabe:

- **Seriennummer** (Zahl): Die Nummer, welche die Reihenfolge der Serien bestimmt.
- **Fälligkeit** (Datum): Das Datum und die Uhrzeit, bis wann die Serie bearbeitet und abgegeben sein muss.
- **Status** (Zahl): Der aktuelle Stand der Serie (offen, abgegeben, bestanden, nicht bestanden).
- **Kommentar** (Zeichenkette): Optionaler Kommentar, welcher der Serie gegeben werden kann.
- **Nutzer** (Zeichenkette): Die ID des Nutzers, welcher die Aufgabe erstellt hat.
- **Kurs** (Zeichenkette): Die ID des Kurses, welcher die Aufgabe angehört.
- **ID** (Zeichenkette): Der eindeutige Identifikator, welcher von der Datenbank gesetzt wird und für jede Aufgabe unterschiedlich ist.

Greift man auf die Beispiele aus der Untersektion 2.1.1 zurück, stellt sich heraus, dass die Kurse A, C und D mit dem in dieser Sektion beschriebenen Datenstruktur und dem Funktionsumfang aus Sektion 2.2 problemlos verwaltet werden können. Für Kurs B besteht jedoch das Problem, dass die Blöcke nicht ohne Weiteres mit dem Programm organisiert werden können, da das Programm die Verwaltung von Punkten nicht unterstützt. Alternativ kann jeder Block jeweils als eine Aufgabe gespeichert und die erhaltenen Punkte für den Block als Kommentar angefügt werden, wobei eine "Aufgabe" als bestanden gilt, wenn die benötigte Punktzahl erreicht wird, bzw. als nicht bestanden gilt, falls dies nicht der Fall ist.

3
Vorstellung des Programms

Das dritte Kapitel befasst sich mit der Vorstellung des Programms basierend auf den in Sektion 2.1.1 vorgestellten Kursen.

3.1. Authentifizierung

Um auf die Kurse und Aufgaben zuzugreifen, ist es notwendig, dass sich der Nutzer mit seinem Konto angemeldet hat. Das Programm überprüft automatisch, ob ein Token mit Authentifikationsdaten im Speicher des Nutzers vorhanden ist, und meldet den Nutzer an, angenommen der Token ist gültig. Fehlt ein solcher Token oder ist dieser nicht mehr gültig, muss der Nutzer sich anmelden, oder falls er noch kein Konto besitzt, sich registrieren.

Abbildung 3.1 zeigt das Formular für die Registrierung, bestehend aus den Feldern "Benutzername", "Passwort" und "Passwort wiederholen", welche zwingend ausgefüllt werden müssen. Der Benutzername kann vom Nutzer frei gewählt werden, muss jedoch ein Unikat sein, da jeder Nutzername nur einmal in der Datenbank vorhanden sein kann. Das Passwort kann ebenfalls frei gewählt werden, muss jedoch die angezeigten Anforderungen (8 bis 32 Zeichen, mind. ein Grossbuchstabe, mind. ein Kleinbuchstabe, mind. eine Zahl, mind. ein Sonderzeichen, keine Leerzeichen) erfüllen. Da der Nutzer nicht sehen kann, was eingegeben wurde, ist die erneute Eingabe des gleichen Passworts im "Passwort bestätigen"-Feld erforderlich. Weicht das dort eingegebene Passwort ab von jenem, welches

im oberen Feld eingegeben wurde, kann die Registrierung nicht abgeschlossen werden, bis die eingegebenen Passwörter miteinander übereinstimmen. Nach der Registrierung wird der Nutzer zur Anmeldeseite weitergeleitet.

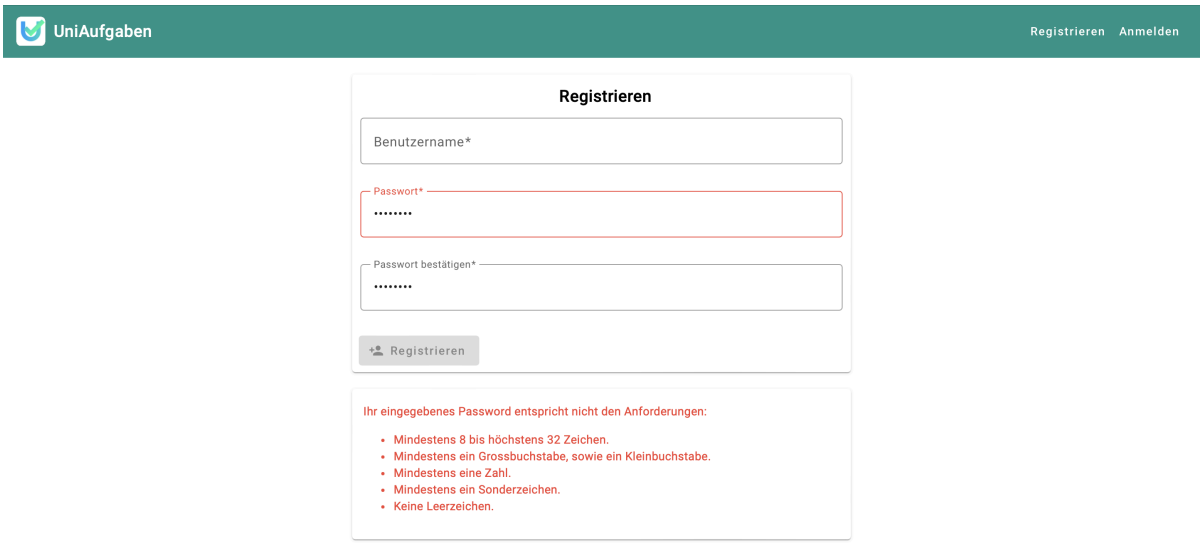

Abbildung 3.1.: Registrierungsseite

Besitzt eine Person bereits ein Konto, kann sie sich mit dem in Abbildung 3.2 abgebildeten Anmeldeformular authentifizieren. Im Feld “Benutzername” ist der bei der Registrierung angegebene Nutzername anzugeben, im Feld “Passwort” das aktuelle Passwort des Kontos. Werden in einem der Felder falsche Daten eingegeben, akzeptiert das Programm die Anmeldung nicht und leert das “Passwort”-Feld. Sind die Daten hingegen korrekt, wird ein Authentifizierungstoken im lokalen Speicher abgesichert, damit der Nutzer automatisch angemeldet wird, solange dieser Token gültig ist[1].

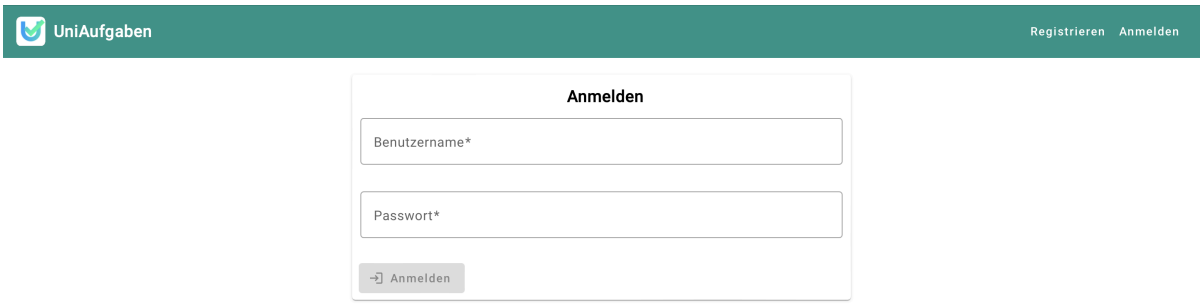

Abbildung 3.2.: Anmeldeseite

[1]Die Gültigkeitsdauer des Tokens ist in Zeile 9 von Listing 4.7 definiert und beträgt 24 Stunden.

3.2. Dashboard

Nach erfolgter Anmeldung wird der Nutzer zum Dashboard weitergeleitet. Das Dashboard, wie auf Abbildung 3.3 ersichtlich, zeigt in der linken Spalte eine Informationsbox, welche die Anzahl insgesamt offener Aufgaben, sowie die Anzahl offener Aufgaben, welche in den nächsten 14 Tagen fällig sind.

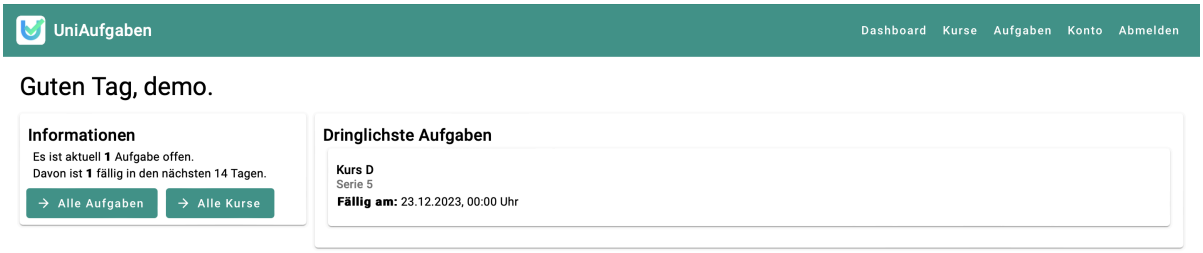

Abbildung 3.3.: Dashboard

Die rechte, grössere Spalte, zeigt die dringlichsten, offenen Aufgaben. Diese werden nach Fälligkeit sortiert und können angeklickt werden, um direkt auf den Kurs zu greifen, welchem die Aufgabe angehört. Die Liste zeigt bis zu 5 Aufgaben an. Sind mehr als 5 Aufgaben vorhanden, wird am unteren Ende angegeben, wie viele der Aufgaben nicht in dieser Liste aufgeführt werden, wie in Abbildung 3.4 dargestellt.

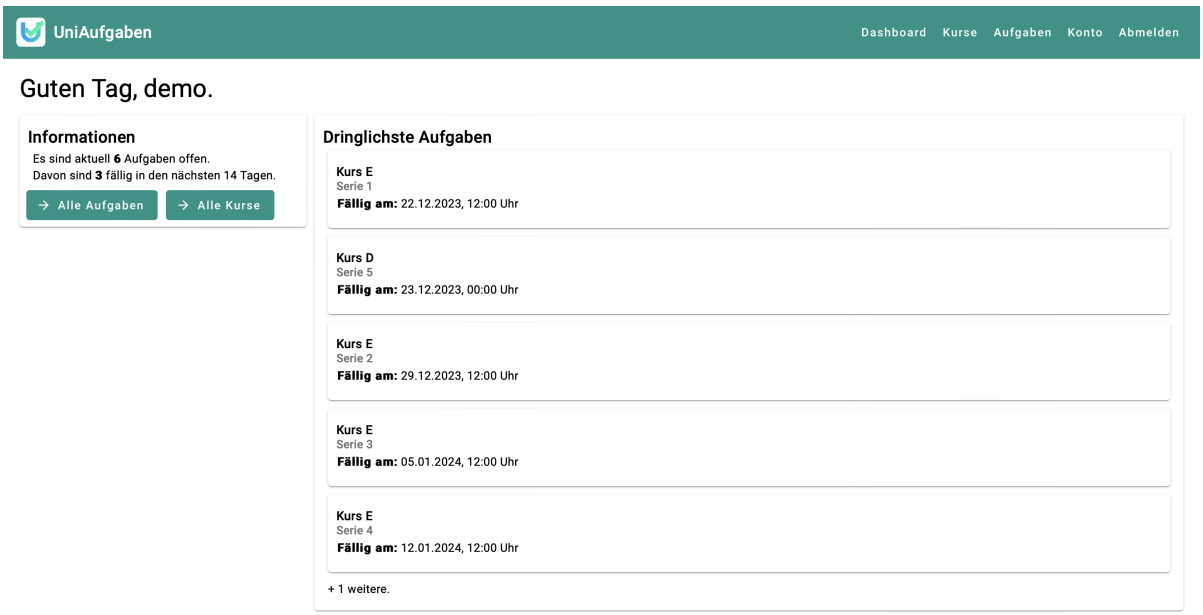

Abbildung 3.4.: Dashboard bei mehr als 5 offenen Aufgaben

3.3. Verwalten der Kurse

3.3.1. Kursliste

Der Nutzer kann auf die Liste seiner Kurse zugreifen, indem er bspw. in der Navigationsleiste auf "Kurse" klickt. Die Kursliste (in Abbildung 3.5 auch "Kursübersicht" genannt) ist eine Tabelle, welche alle Kurse beinhaltet, welche der angemeldete Nutzer erstellt hat. Die Person kann einen Kurs anklicken, um die Übersicht des ausgewählten Kurses erscheinen zu lassen. Die Übersicht besteht aus dem Namen des Kurses, dem Semester, in welchem der Kurs stattfindet, dem Fach, welchem der Kurs angehört, den aktuellen Stand der Bewertungen der Serien des Kurses (falls die Serien bewertet werden), sowie Schaltflächen zum Anzeigen der Aufgaben, Bearbeiten und Löschen des Kurses.

Für den Kurs A wurden 10 von 12 Serien bestanden, weshalb der Kurs als bestanden gilt. Der Stand der Bewertungen der Serien wird in der Übersicht des Kurses mit einem Fortschrittsbalken dargestellt. Wie in Abbildung 3.5 gesehen werden kann, ist der Balken für Kurs A zu 100% gefüllt, da 10 von 9 benötigten Serien bestanden wurden.

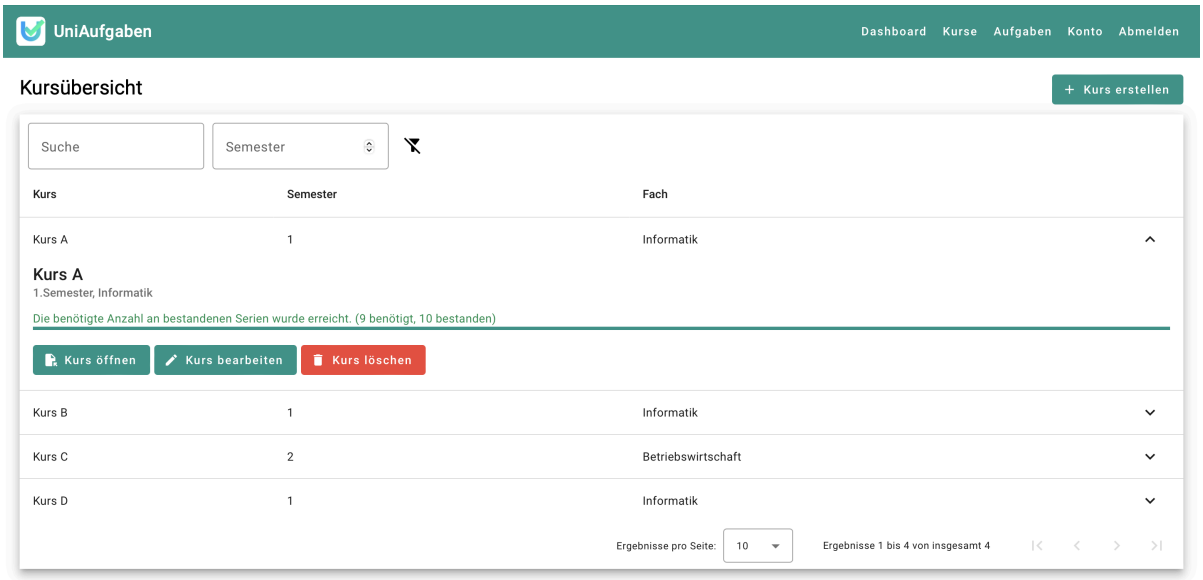

Abbildung 3.5.: Kursliste mit offener Übersicht eines bestandenen Kurses

Bei einem noch nicht abgeschlossen Kurs, wie Kurs D aus Abbildung 3.6, zeigt der Balken den Fortschritt relativ zur benötigten Anzahl an Serien, welche bestanden werden müssen. Bei Kurs D wurden von den 4 benötigten Serien 3 bestanden. Der Balken ist deswegen zu 75% gefüllt.

Abbildung 3.6.: Übersicht eines Kurses, bei welchem 75% der Serien bestanden wurden

Im bedauerlichen Fall, dass es in einen Kurs nicht mehr möglich ist, die benötigte Anzahl an zu bestehende Serien zu erreichen, wird der Fortschrittsbalken rot gefärbt und mit dem entsprechenden Hinweis versehen, wie es für den Kurs B in Abbildung 3.7 der Fall ist.

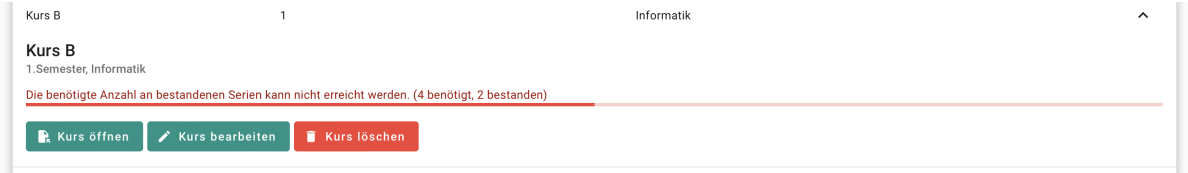

Abbildung 3.7.: Übersicht eines Kurses, welcher nicht mehr bestanden werden kann

Kurse, deren Aufgaben nicht bewertet werden, zeigen keinen Fortschrittsbalken. Wie bei Kurs C aus Abbildung 3.8, wird stattdessen der Hinweis gezeigt, dass der Kurs nicht bewertet wird.

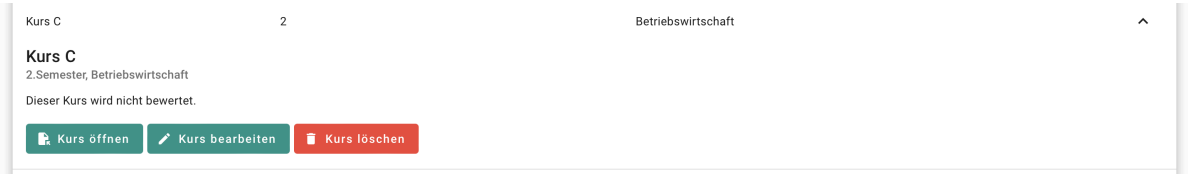

Abbildung 3.8.: Übersicht eines Kurses, welcher nicht bewertet wird

Eine der benötigten Funktionen, welche im Kapitel 2 genannt wurde, ist die Sortierung. In der obersten Zeile der Tabelle, in welcher die Kurse aufgelistet sind, können die Kategorien angeklickt werden, um die Sortierung umzuschalten (aufsteigend, absteigend, aus). In Abbildung 3.9 werden die Kurse bspw. in aufsteigender, alphabetischer Reihenfolge der Namen der Fächer aufgelistet, dargestellt durch einen kleinen Pfeil, welcher sich auf der rechten Seite des Titels der Kategorie befindet.

Abbildung 3.9.: Kursliste, sortiert nach Fach

Die Kurse können auch nach Kategorie gefiltert werden. Wie Abbildung 3.10 zeigt, ist es möglich, in den Eingabefeldern oberhalb der Tabelle eingrenzende Begriffe einzutragen. Wie im Bild gezeigt, kann bspw. im Suchfeld "Suche" der Begriff "Informatik" und im Feld "Semester" eine 1 eingegeben werden, um nur die Kurse vom Fach "Informatik" aus dem 1. Semester anzuzeigen. Die Filter können entfernt werden, indem der Nutzer auf den durchgestrichenen Trichter neben den Eingabefeldern klickt.

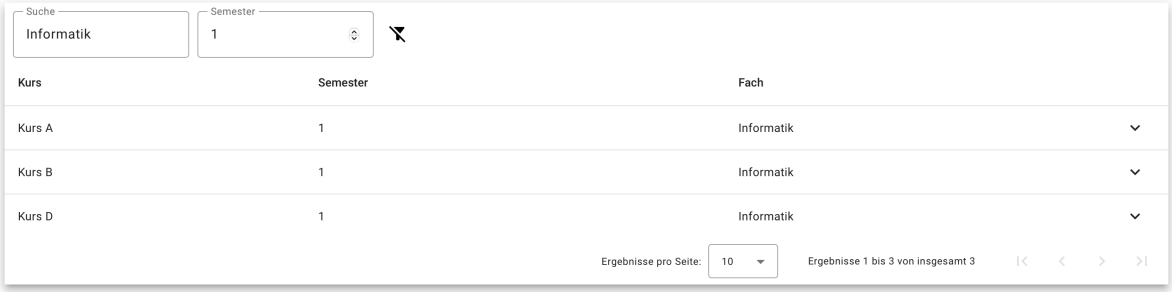

Abbildung 3.10.: Kursliste, gefiltert nach den Suchkriterien "Informatik" und 1. Semester

In der Übersicht eines Kurses kann dieser auch gelöscht werden. Klickt eine Person auf den Knopf "Kurs löschen", erscheint ein Dialogfenster, welches um Bestätigung der Löschung bittet, wie in Abbildung 3.11 gezeigt. Wird die Löschung bestätigt, bleibt das Dialogfenster solange offen, bis die Löschung des Kurses und dessen Aufgaben vollständig abgeschlossen ist. Danach wird das Fenster geschlossen und die Kursliste wird aktualisiert.

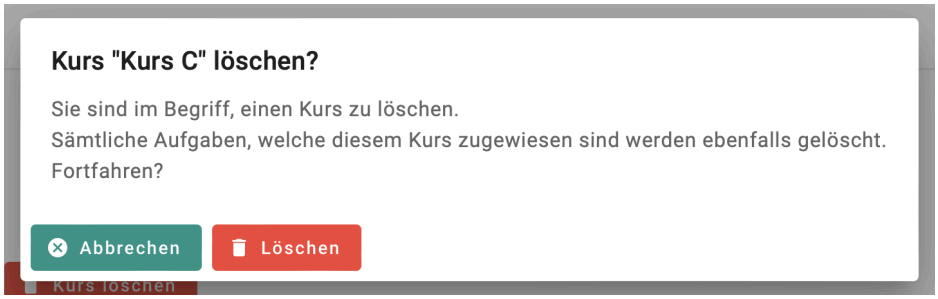

Abbildung 3.11.: Dialogfenster zur Bestätigung der Löschung

3.3.2. Erstellen und Bearbeiten eines Kurses

Möchte eine Person einen neuen Kurs erstellen, so kann sie auf den Knopf "Kurs erstellen" aus Abbildung 3.5 klicken, und wird dann zur Erstellungsseite weitergeleitet. Die Erstellungsseite, gezeigt auf Abbildung 3.12, beinhaltet ein Formular, welches vom Nutzer ausgefüllt werden muss. Der Nutzer muss den Namen des Kurses, das Semester, sowie das Fach, welches der Kurs angehört, angeben. Unterhalb dieser Eingabefelder befindet sich ein Schalter, welcher eingeschaltet werden kann, wenn die Serien bewertet werden. Ist der Schalter aktiviert, werden zwei weitere Eingabefelder (Anzahl Serien insgesamt, Anzahl zu bestehende Serien) eingeblendet. Ist dieser nicht aktiviert, sind diese zwei Eingabefelder nicht sichtbar.

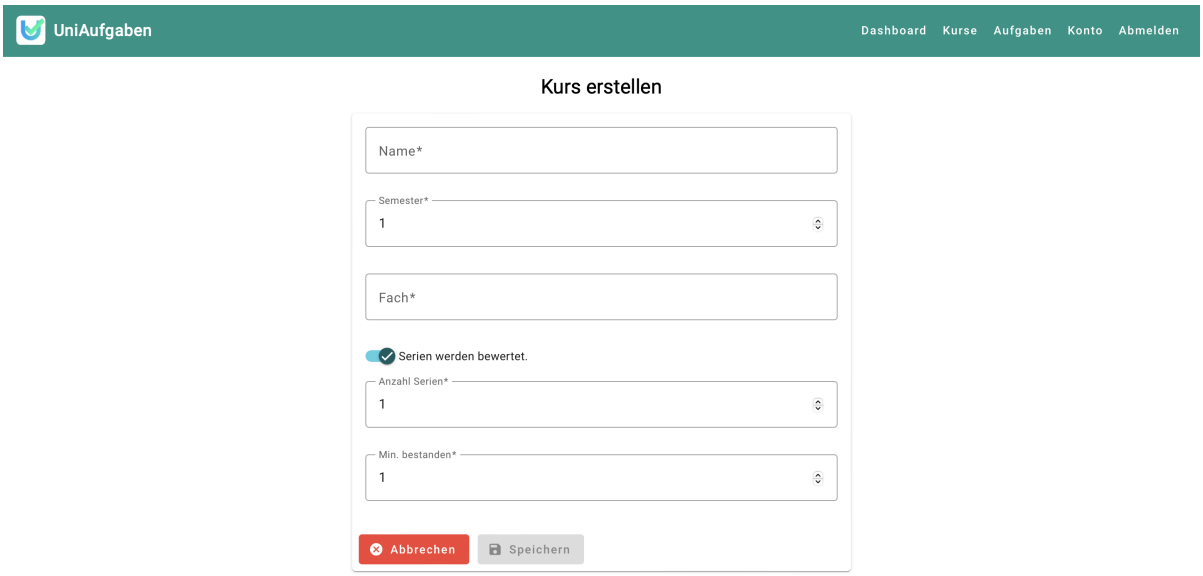

Abbildung 3.12.: Seite zur Erstellung eines Kurses

Möchte eine Person stattdessen einen Kurs bearbeiten, kann sie bei der Kursübersicht auf den Knopf “Kurs bearbeiten” des gewünschten Kurses klicken. Das gleiche Formular wird geladen wie bei der Erstellung, allerdings befindet sich dieses stattdessen im Bearbeitungsmodus, d.h. es wird mit den Daten des gewählten Kurses ausgestattet (siehe Abbildung 3.13) und erstellt keinen neuen Kurs, sondern ändert lediglich die Daten des gewählten Kurses.

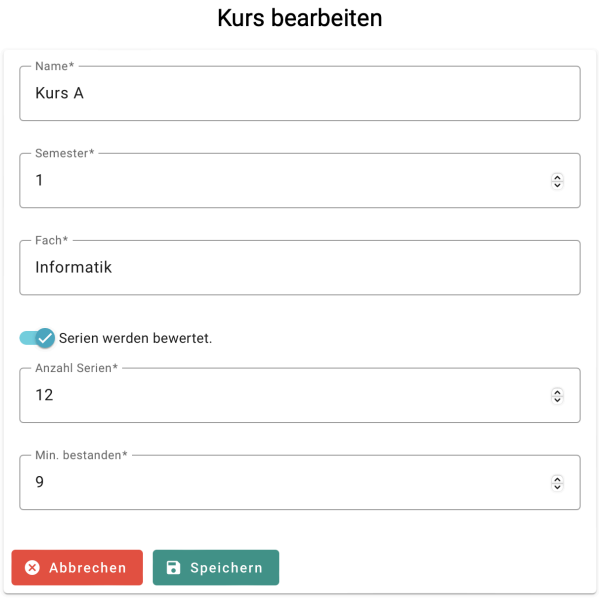

Abbildung 3.13.: Seite zur Bearbeitung eines Kurses

3.4. Verwalten der Aufgaben

3.4.1. Aufgabenliste

Um auf die Liste mit allen Aufgaben zuzugreifen, kann in der Navigationsleiste auf "Aufgaben" geklickt werden. Dem Nutzer wird dann, wie Abbildung 3.14 zeigt, eine Tabelle mit den Aufgaben aller Kurse präsentiert. Für jede Aufgabe werden Kurs, Seriennummer, Fälligkeit, Status, Kommentar, sowie die Option zum Bearbeiten und die Option zum Löschen der Aufgabe angezeigt.

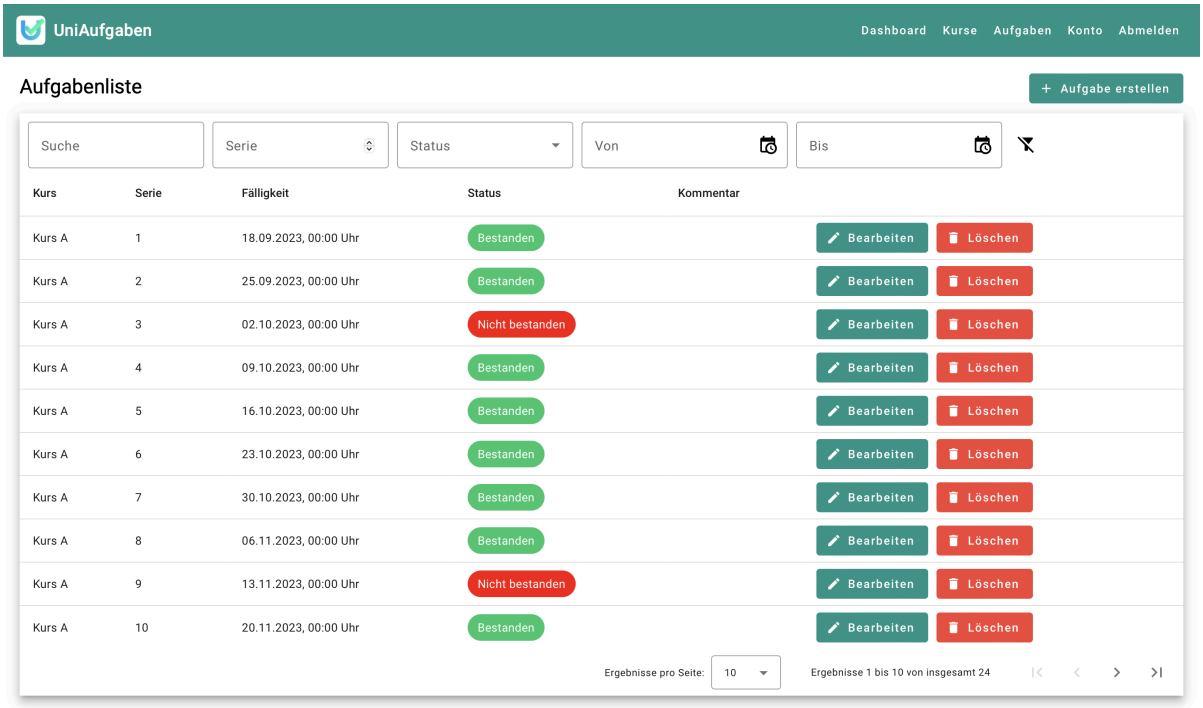

Abbildung 3.14.: Aufgabenliste

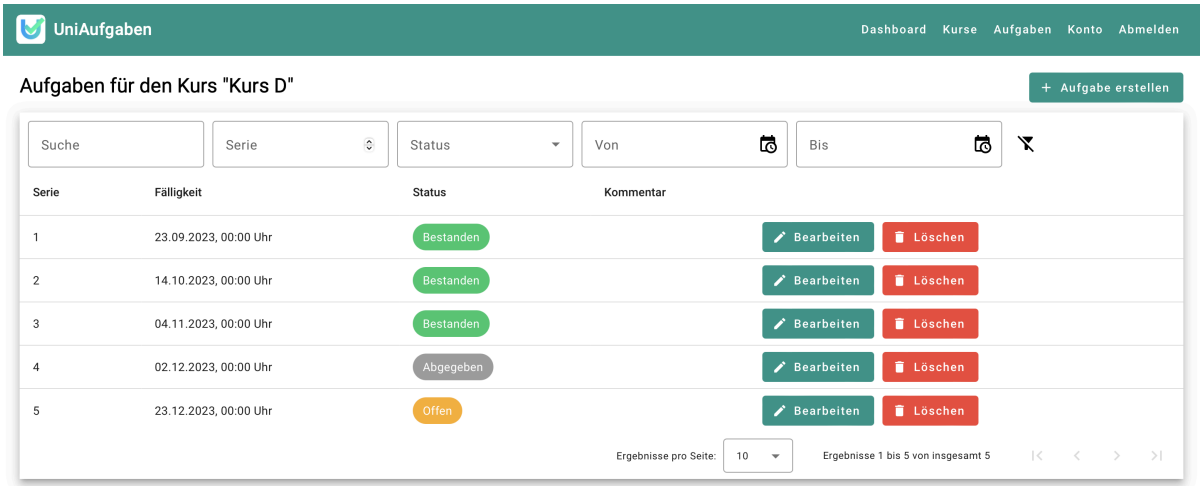

Abbildung 3.15.: Aufgabenliste innerhalb eines Kurses

Öffnet eine Person hingegen einen Kurs, werden nur die Aufgaben des gewählten Kurses angezeigt. Anders als bei der Liste aller Aufgaben, wie in Abbildung 3.15 ersichtlich, ist hier die Spalte "Kurs" nicht vorhanden, da diese hier redundant ist.

Die Aufgaben können in aufsteigender oder absteigender Reihenfolge nach Kurs, Serie, Fälligkeit oder Status sortiert werden. In Abbildung 3.16 werden die Aufgaben nach absteigendem Fälligkeitsdatum sortiert, dargestellt durch den kleinen Pfeil neben dem Namen der Spalte "Fälligkeit".

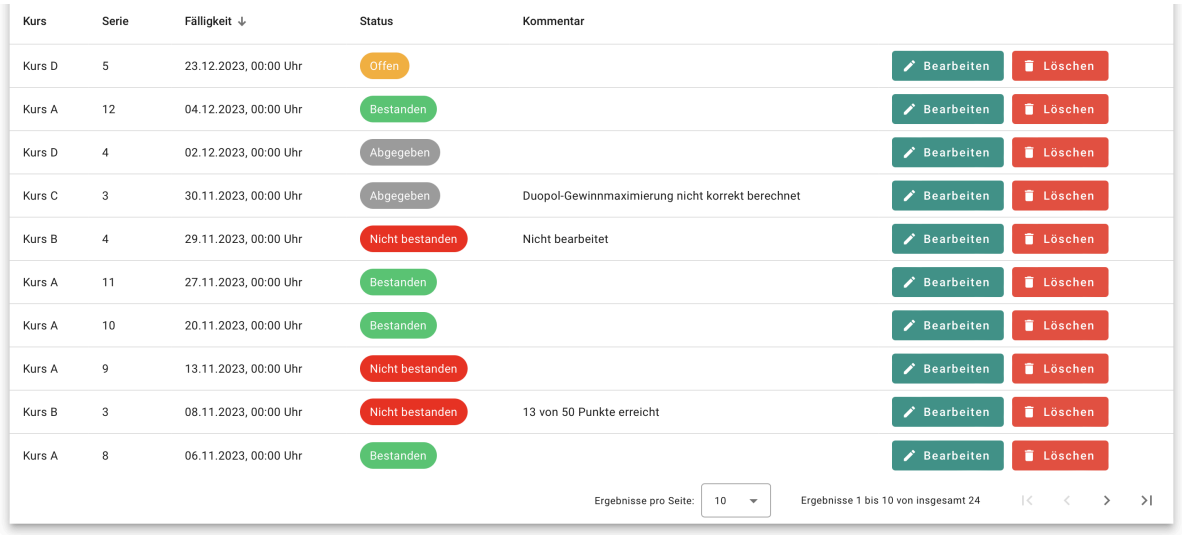

Abbildung 3.16.: Aufgabenliste, sortiert nach Fälligkeit

Es kann auch nach Suchkriterien gefiltert werden. Eingegebener Text im Feld "Suche" findet alle Aufgaben, bei welchen der Text in der Spalte "Kurs" und/oder "Kommentar" (bzw. innerhalb eines Kurses nur "Kommentar") übereinstimmt. In Abbildung 3.17 wird als Beispiel der Text "Punkte" ins Suchfeld eingetragen. Nebst dem kann auch nach Seriennnummer und Status gefiltert werden. Es ist ebenfalls möglich, die Aufgaben nach einem Zeitraum zu filtern. Entfernt werden können die Filter durch Klick auf den durchgestrichenen Trichter.

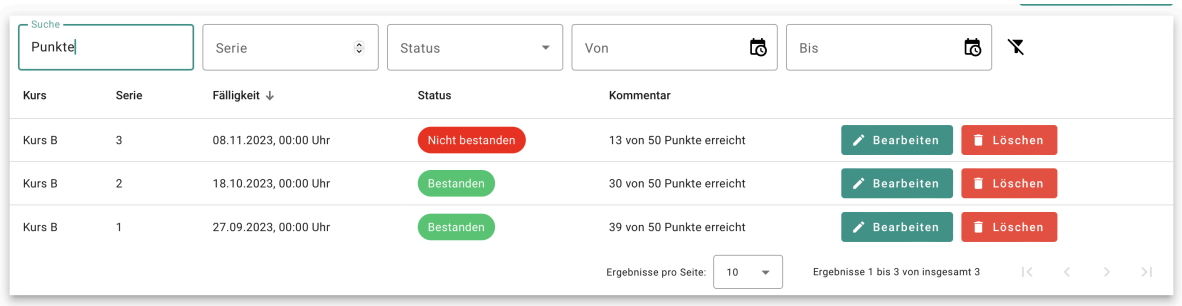

Abbildung 3.17.: Aufgabenliste, gefiltert nach dem Suchbegriff "Punkte".

Will die Person eine Aufgabe löschen, kann dies durch Klick auf die entsprechende "Löschen"-Schaltfläche getan werden. Wie bei den Kursen wird der Nutzer hier gebeten, den Löschvorgang zu bestätigen.

3.4.2. Erstellen und bearbeiten einer Aufgabe

Um eine Aufgabe zu erstellen, muss auf die Schaltfläche “Aufgabe erstellen” geklickt werden (siehe Abbildung 3.14). Die Seite zur Erstellung einer Aufgabe wird geöffnet und zeigt ein Formular an, in welchem der Nutzer angeben muss, welchem Kurs die Aufgabe angehört, welche Nummer die Serie hat, bis wann die Aufgabe abgegeben werden muss und welchen Status sie hat, wie in Abbildung 3.18 gezeigt. Der Nutzer kann auch einen eigenen Kommentar verfassen. Hat der Nutzer zu diesem Zeitpunkt noch keinen Kurs erstellt, kann er auch keinen Kurs auswählen und wird gebeten, als Erstes einen Kurs zu erstellen. Falls der Nutzer innerhalb eines Kurses auf den “Aufgabe erstellen”-Knopf klickt, wird im Feld “Kurs” automatisch der entsprechende Kurs ausgewählt.

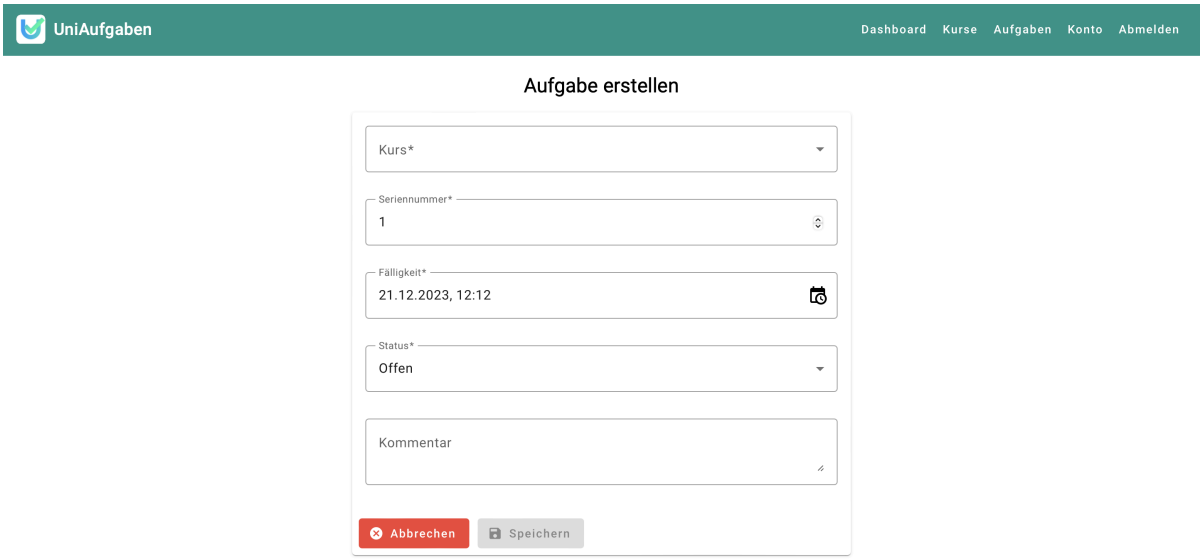

Abbildung 3.18.: Seite zur Erstellung einer Aufgabe

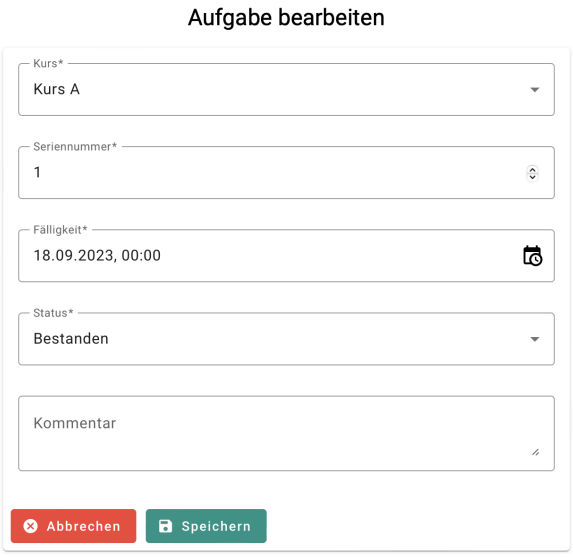

Abbildung 3.19.: Seite zur Bearbeitung einer Aufgabe

Wird eine Aufgabe zur Bearbeitung gewählt, wird das gleiche Formular wie bei der Erstellung einer Aufgabe geöffnet und mit den Informationen der gewählten Serie versehen, wie in Abbildung 3.19 dargestellt. Auf dieser Weise kann bspw. der Status der Serie verändert werden. Auch der Kurs kann gewechselt werden, falls nötig. Die Bewertungen werden dann entsprechend angepasst.

3.5. Verwalten des Kontos

Das eigene Konto kann auf der Kontoseite verwaltet werden. Die Seite kann durch Klick auf "Konto" in der Navigationsleiste aufgerufen werden. Auf dieser Seite, wie Abbildung 3.20 zeigt, kann das eigene Passwort geändert werden, sowie das Konto geschlossen und gelöscht werden.

Um das Passwort ändern zu können, ist es nötig, dass der Nutzer sein aktuelles Passwort eingibt. Das neue Passwort muss die gleichen Bedingungen erfüllen, wie bei der Registrierung. Ebenfalls wie bei der Registrierung, muss das Passwort dann im nächsten Eingabefeld bestätigt werden.

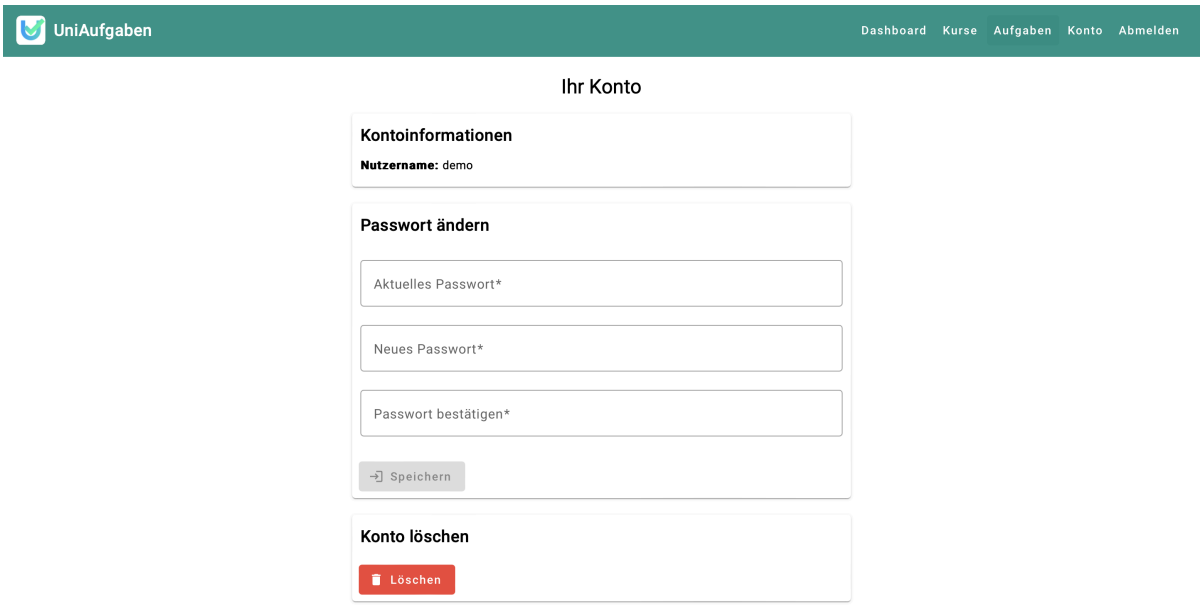

Abbildung 3.20.: Kontoseite

Braucht der Nutzer sein Konto nicht mehr, kann er auf die Schaltfläche "Löschen" klicken. Abbildung 3.21 zeigt das Dialogfenster, welches daraufhin geöffnet wird. Es ist notwendig, dass der Nutzer sein Passwort eingibt, um den Löschvorgang bestätigen zu können. Wird dieser bestätigt, bleibt das Fenster offen, bis alle Aufgaben, alle Kurse sowie das Konto selbst aus der Datenbank entfernt wurden. Schliesslich wird der Nutzer abgemeldet und zur Anmeldeseite weitergeleitet. Das Konto ist somit geschlossen.

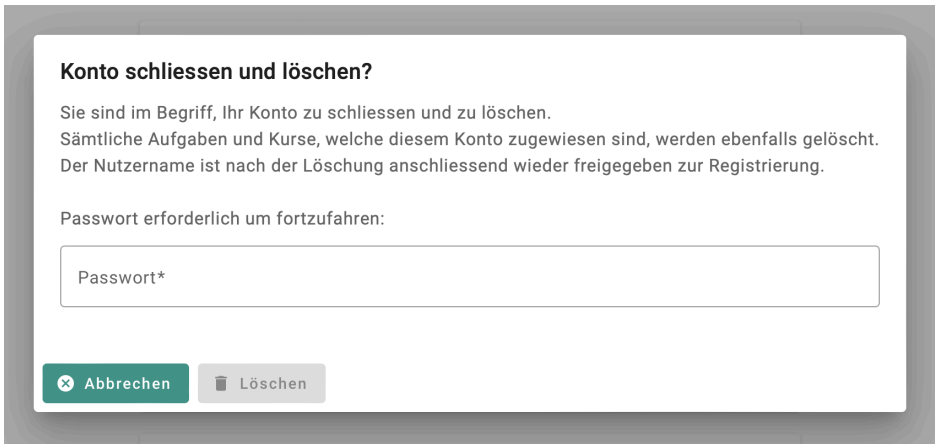

Abbildung 3.21.: Dialogfensters zur Bestätigung der Kontoschliessung

4
Implementation

Dieses Kapitel stellt die konkrete Implementation des Programms vor, welches aus Server und Client besteht. Das Programms basiert auf das MEAN-Stapel Tutorial von JavaTpoint, welches eine Schritt-für-Schritt-Anleitung beinhaltet zur Erstellung einer Webapplikation, bei welcher Nutzer Beiträge erstellen können [15]. Für viele Probleme wurden Lösungen genutzt, welche auf Stack Overflow [34] gefunden wurden.

4.1. Verwendete Technologien

Um die in Kapitel 2 vorgestellte Applikation zu entwickeln, wurde der MEAN-Technologiestapel verwendet, welches sich aus Folgendem zusammensetzt [15]:

- **MongoDB** [20]: Eine NoSQL-Datenbank, welche die Nutzer, Kurse und Aufgaben speichert.
- **Express.js** [11]: Ein Framework für Node.js (siehe letzten Punkt dieser Liste), mit welchem die Anfragen zwischen Client und Server durchgeführt werden.
- **Angular** [2]: Ein Framework, welches verwendet wird, um die Benutzeroberfläche, bzw. den Client aufzubauen.

– **Node.js** [25]: Wird genutzt, um das JavaScript-Engine auszuführen, welches für den Server benötigt wird.

Ausserdem wurden verschiedene mit npm (ein Paketmanager für Node.js) installierte Erweiterungen genutzt, unter anderem *Mongoose* [21], welche verwendet wurde, um die Implementation der Datenbankanfragen zu vereinfachen, *mongoose-unique-validator* [22], um sicherzustellen, dass jeder Nutzername nur einmal in der Datenbank vorkommen kann, *Bcrypt* [7], mit welcher die Passwörter verschlüsselt und überprüft werden können, *jsonwebtoken* (JWT) [16], mit welcher die Anmeldedaten gespeichert, bzw. überprüft werden können und *dotenv* [10], mit welcher wichtige Variablen in einer `.env`-Datei ausserhalb des eigentlichen Programms definiert werden können.

4.2. Server

Mit dem Server, bzw. Backend, werden Anfragen zwischen Client und Datenbank durchgeführt. Erstellt der Nutzer beispielsweise einen Kurs, werden die im Client eingegebenen Daten an den Server weitergeleitet, welcher dann die Daten an die Datenbank sendet und bei erfolgreichem Hinzufügen des Eintrags dem Client die entsprechende Bestätigung von der Datenbank zurücksendet.

Die Kommunikation zwischen Client und Server geschieht mit einer API gemäss REST [15]. Services vom Frontend senden GET (Abfrage eines Dokuments), POST (Erstellen eines neuen Dokuments), PATCH (Bearbeiten eines bestehenden Dokuments) und DELETE (Löschen eines Dokuments) Anfragen an die dafür vorgesehenen Endpunkte (auf Englisch "Endpoints" bezeichnet) vom Server, welcher dann die entsprechenden Anfragen an die Datenbank durchführt. Die Endpunkte vom Server werden in der Untersektion 4.2.1 vorgestellt.

4.2.1. Endpunkte

Endpunkte sind Punkte auf Seite des Servers, bei welchen die API-Anfragen erfüllt werden [8]. Der Client sendet Anfragen an die jeweilige URL, an welcher der Endpunkt auf dem Server vorhanden ist und dieser führt dann die benötigte Funktionen aus. Der Aufbau einer solchen URL kann am Beispiel

`https://localhost:3000/api/v1/aufgaben?dashboard=true` [26]

gezeigt werden [32]:

- **Protokoll**: Das Protokoll, welches verwendet wird, um auf den Endpunkt zuzugreifen. Im Fall der Applikation ist es `http` (HTTP: Hypertext Transfer Protocol).
- **Hostname**: Besteht aus einer Domain (hier: `localhost`) und gegebenenfalls einem Port (hier: `3000`).
- **Pfad**: Weist auf eine Datei oder einen Ort. In Fall von der Applikation wird mit dem Pfad der korrekte Endpunkt gewählt. Im Beispiel ist `aufgaben` der Pfad.
- **Parameter und Parameterwert**: Am Ende des Pfades können Parameter angehängt werden. Das Fragezeichen deklariert den Anfang der Parameter. Im Beispiel ist `dashboard` ein Parameter und `true` der dazugehörige Parameterwert.

Die Tabellen 4.1, 4.2 und 4.3 listen alle Endpunkte auf, welche in der Applikation verwendet werden. Jeder Punkt zeigt die verwendete Methode, das Ende des Pfades der verwendeten URL, da diese immer mit `http://localhost:3000/api/v1/` beginnt (bzw. die Domain des Servers anstelle von `localhost`, falls zutreffend), gefolgt von einer kurzen Beschreibung.

Methode	Pfad	Beschreibung
GET	`kurse/`	Ruft alle Kurse des Nutzers ab.
GET	`kurse/:id`	Ruft einen spezifischen Kurs ab. `:id` ist eine Variabel, welche die ID des aufzurufenden Kurses beinhaltet.
POST	`kurse/`	Erstellt einen neuen Kurs.
PATCH	`kurse/:id`	Bearbeitet den Kurs, welcher die in der Variabel `:id` angegebene ID trägt. Befindet sich der Parameter `?checkBewertung=true` in der URL, werden die Bewertung der Serien des Kurses gezählt und entsprechend in der Datenbank aktualisiert.
DELETE	`kurse/`	Löscht alle Kurse eines Nutzers.
DELETE	`kurse/:id`	Löscht jenen Kurs, welcher die ID besitzt, welche in `:id` angegeben ist.

Tabelle 4.1.: Endpunkte für Kurse

Methode	Pfad	Beschreibung
GET	`aufgaben/`	Ruft alle Aufgaben eines Nutzers ab.
GET	`aufgaben/stats`	Ruft die Anzahl offener Aufgaben ab (Anzahl insgesamt und Anzahl fällig in den nächsten 14 Tagen).
GET	`aufgaben/:id`	Ruft die Aufgabe, welche die in der Variable `:id` angegebene ID trägt.
POST	`aufgaben/`	Erstellt eine neue Aufgabe.
PATCH	`aufgaben/:id`	Bearbeitet die Aufgabe, welche die in der Variabel `:id` angegebene ID besitzt.
DELETE	`aufgaben/`	Löscht alle Aufgaben eines Nutzers, oder falls die ID eines Kurses angegeben wurde (Parameter `?kursID=...`), alle Aufgaben des gewählten Kurses.
DELETE	`aufgaben/:id`	Löscht jene Aufgabe, welche die ID besitzt, welche in `:id` angegeben ist.

Tabelle 4.2.: Endpunkte für Aufgaben

Methode	Pfad	Beschreibung
POST	`nutzer/signup`	Erstellt ein neues Benutzerkonto.
POST	`nutzer/login`	Meldet einen Nutzer an.
PATCH	`nutzer/:id`	Bearbeitet einen Nutzer, welcher die ID trägt, welche in `:id` angegeben ist.
DELETE	`nutzer/`	Löscht das angemeldete Benutzerkonto.

Tabelle 4.3.: Endpunkte für Nutzer

Der in Tabelle 4.3 aufgelistete Endpunkt zur Anmeldung hat die Methode POST, obwohl beim Aufruf kein neues Konto erstellt wird. Dies liegt daran, dass die Anmeldedaten nicht in der URL selbst enthalten sind, sondern im Body der Anfrage. Obwohl es nicht verboten ist, einer GET-Anfrage eine Payload zu geben, wird davon abgeraten, da die Semantik hierfür nicht definiert ist [18].

Mit einem Tool wie Postman [27] ist es möglich, manuell Anfragen an den Server zu erstellen. Abbildung 4.1 zeigt als Beispiel eine GET-Anfrage, welche den Kurs A aus Kapitel 2 abruft.

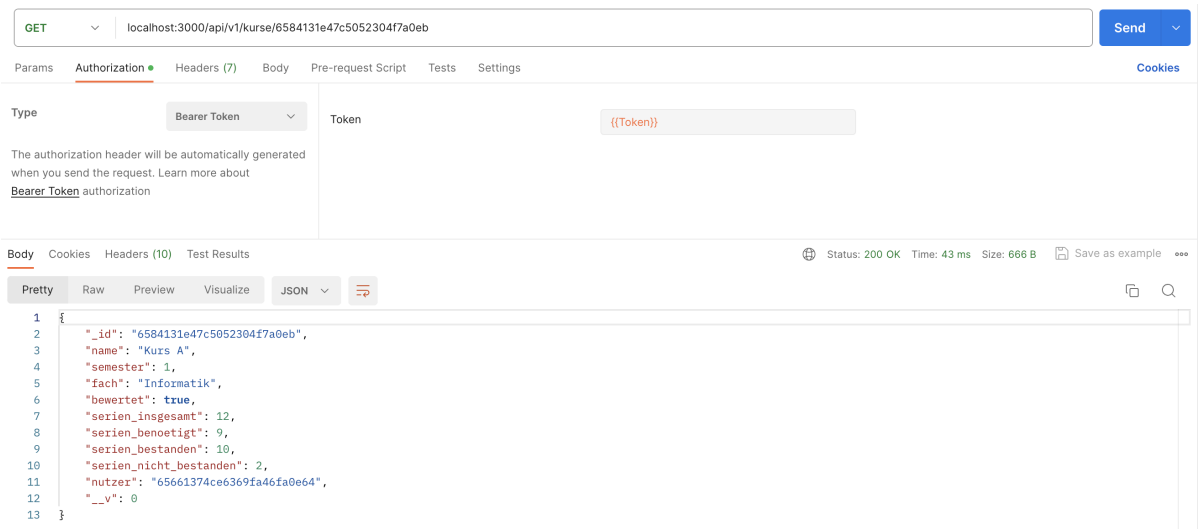

Abbildung 4.1.: GET-Anfrage für den Kurs A

Bei einer GET-Anfrage wie jene aus Abbildung 4.1 wird eine Payload zurückgeschickt, falls die GET-Anfrage erfolgreich war (Status 200, bzw. OK). Im Fall von der GET-Anfrage für Kurs A, werden die Informationen zu Kurs A als JSON-Objekt gesendet. Das JSON-Objekt beinhaltet alle in Attribute der Kurs-Entität (Sektion 2.4), sowie die ID des Kurses (`_id`). `__v` ist die Version des Eintrags, wird jedoch in dieser Arbeit nicht aktiv verwendet.

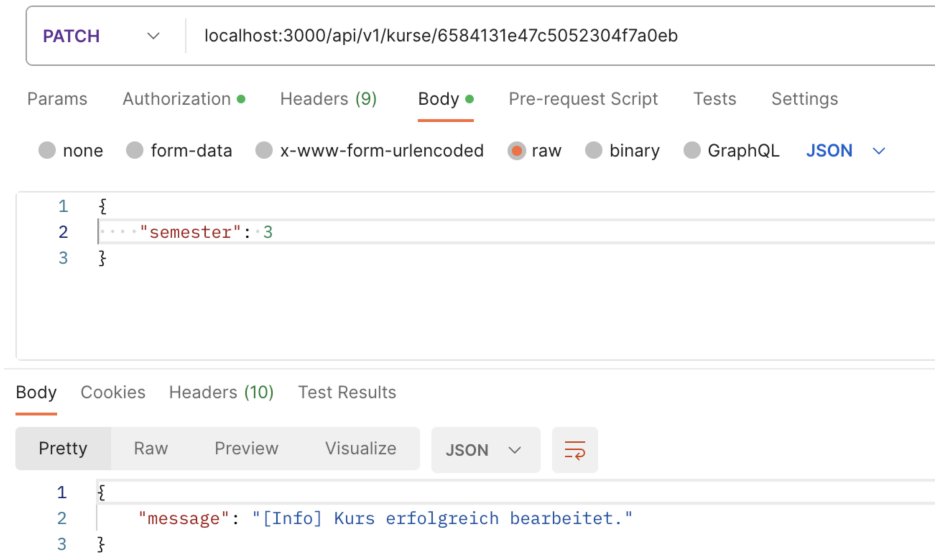

Abbildung 4.2.: PATCH-Anfrage für den Kurs A, um das Semester zu ändern

Bei POST-, oder auch PATCH-Anfragen wie jene in Abbildung 4.2, kann der Anfrage eine Payload angehängt werden. Im Beispiel aus der Abbildung wurde das Semester von Kurs A geändert. Hierfür wurde ein JSON-Objekt angegeben, welches in diesem Fall nur das Semester ändern soll, und deswegen nur aus `"semester": 3` besteht. Sobald diese Aktion ausgeführt wurde, antwortet der Server entsprechend mit einer Nachricht und dem HTTP Status Code 200.

Mit HTTP Status Codes kann angegeben werden, ob eine HTTP-Anfrage erfolgreich erfüllt werden konnte, oder ob diese gescheitert ist. Relevant sind in dieser Arbeit die Status Codes der Kategorien 200-299, 400-499 und 500-599. Codes im Bereich 200 bis 299 bestätigen den erfolgreichen Abschluss der Anfrage. Zum Beispiel bedeutet 200 "OK" und 201 "erstellt". Codes im Bereich 400 bis 499 geben an, dass ein Problem vorhanden ist, welches vom Client verursacht wurde. Zum Beispiel bedeutet 401, dass der Client nicht authentifiziert ist, oder 404, dass die Ressource nicht gefunden wurde. Codes im Bereich 500 bis 599 melden ein Problem auf Seite des Servers. Code 500 bedeutet zum Beispiel, dass der Server auf eine Situation gestossen ist, bei welcher er nicht weiss, was zu tun ist [33].

Es sollte angemerkt werden, dass bei allen Endpunkten eine Authentifizierung erforderlich ist, mit Ausnahme der beiden POST-Endpunkte für Nutzer aus Tabelle 4.3. Dies liegt daran, dass eine Middleware stets den Authentifizierungstoken abruft, bevor die eigentliche Anfrage erfüllt wird, damit garantiert wird, dass eine Person ausschliesslich Änderungen an ihren Kursen, ihren Aufgaben und ihr Konto durchführen kann, bzw. nicht die Kurse und Aufgaben anderer Personen abrufen und verändern kann. In Postman kann der Authentifizierungstoken, wie Abbildung 4.1 zeigt, im Tab "Authorization" eingetragen werden. Dieser Token kann etwa aus dem lokalen Speicher des Browsers eines eingeloggten Nutzers kopiert werden. Weiteres zur Authentifizierung wird in Untersektion 4.2.4 erklärt.

4.2.2. Konfiguration von Mongoose und Express.js

Damit sich der Server mit der Datenbank verbinden kann, ist es notwendig, Mongoose und Express.js zu konfigurieren.

Mongoose führt die Anfragen zwischen Server und Datenbank durch. Hierfür muss bei MongoDB eine URL generiert werden, welche Mongoose verwenden kann, um sich zu authentifizieren und auf die Datenbank zugreifen zu können. Fehlt die Authentifizierung, verweigert MongoDB jegliche Anfrage. Listing 4.1 zeigt einen Ausschnitt aus dem Code, in welchem Mongoose die Verbindung zur Datenbank herstellt. Zeile 2 beinhaltet die URL, mit welcher die Authentifizierung durchgeführt wird. Diese beinhaltet die Variablen `DB_AUTH_USERNAME` (Nutzername), `DB_AUTH_PASS` (Passwort), `DB_AUTH_HOST` (IP oder URL des Hosts) und `DB_AUTH_COLLECTION` (Sammlung in der Datenbank), welche nicht im Programm selbst gespeichert sind, sondern in einer externen ".env"-Datei [19]. Schliesslich wird die Verbindung aufgebaut durch Aufruf der Funktion `mongoose.connect(dbURL)` in Zeile 5.

```
1  // Definiert URL, mit welcher zur MongoDB Datenbank verbunden wird.
2  const dbURL = "mongodb+srv://"+process.env.DB_AUTH_USERNAME+":"+process.env.
       DB_AUTH_PASS+"@"+process.env.DB_AUTH_HOST+'/'+process.env.DB_AUTH_COLLECTION+'?';
3
4  // Datenbank Verbindung aufbauen.
5  mongoose.connect(dbURL)
6    .then(function () {
7      console.log("[Info] Verbindung zur Datenbank hergestellt.");
8    })
9    .catch(function () {
10     console.log("[Fehler] Verbindung zur Datenbank fehlgeschlagen.");
11   });
```

Listing 4.1: Konfiguration von Mongoose zur Verbindung zur Datenbank

Da der Client und Server nicht auf demselben Port laufen, ist es nötigt, Access-Control-Headers zu setzten, da ansonsten CORS-Fehler (Cross-Origin Resource Sharing) auftreten können, wenn der Nutzer jegliche Aktion ausführt, bei welcher die Kommunikation zwischen Client und Server benötigt wird. Listing 4.2 zeigt die hierfür gesetzten Headers; `Access-Control-Allow-Origin`, welches bestimmt, welcher Dienst Zugriff hat, `Access-Control-Allow-Headers`, welches beschränkt, welchen Header die Anfragen haben können, sowie `Access-Control-Allow-Methods`, welches beschränkt, welche Methoden verwendet werden dürfen [15].

```
1  // Access-Control einstellen.
2  app.use((req,res,next) => {
3    res.setHeader("Access-Control-Allow-Origin","*");
4    res.setHeader("Access-Control-Allow-Headers","Origin, X-Requested-With, Content-Type
         , Accept, Authorization");
5    res.setHeader("Access-Control-Allow-Methods", "GET, POST, PATCH, PUT, DELETE,
         OPTIONS");
6    next();
7  });
```

Listing 4.2: Konfiguration der Access-Control-Header

Die Endpunkte sind aufgeteilt in drei Dateien, jeweils eine für Nutzer-Anfragen, Kurse-Anfragen und Aufgaben-Anfragen. Diese werden in die Hauptdatei des Servers einbezogen durch Nutzung von `app.use`, wie Listing 4.3 zeigt. Die Endpunkte aus den Dateien werden im Fall der Applikation als "Routen" bezeichnet.

```
1 const apiString = "/api/v1"
2 app.use((apiString + "/kurse"), kursRouten);
3 app.use((apiString + "/aufgaben"), aufgabenRouten);
4 app.use((apiString + "/nutzer"), nutzerRouten);
```

Listing 4.3: Einbezug der Routen

4.2.3. Erfüllung von Anfragen

Für jeden Endpunkt existiert im Programm eine Router-Funktion, welche von Express.js gehandhabt wird. Listing 4.4 zeigt zum Beispiel die Funktion, welche verwendet wird, um einen Kurs abzurufen. Die oberste Zeile bestimmt die HTTP-Methode (`get`), den Pfad (`/:id`), sowie die auszuführende Funktion, welche ausgeführt wird, wenn dieser Endpunkt angesprochen wird. Diese Funktion erhält die Variablen `req` (Request, bzw. Anfrage) und `res` (Response, bzw. Antwort). `req` beinhaltet die Daten, welche verwendet werden, um die Anfrage zu erfüllen. Im Fall des Beispiels sind dies die ID des Nutzers (`req.userData.nutzerID` in Zeile 2) und die ID des Kurses (`req.params.id` in Zeile 5).

```
1  router.get('/:id', checkAuth, (req, res) => {
2    const nutzerID = req.userData.nutzerID;
3
4    // Sucht den Kurs anhand der ID.
5    kursModell.findOne({_id: req.params.id, nutzer: nutzerID})
6      .then((result) => {
7        if(result){
8          res.status(200).json(result);
9        } else {
10         res.status(404).json({
11           message: "Kurs nicht gefunden."
12         });
13       }
14     })
15     .catch(function () {
16       res.status(500).json({
17         message: "Abrufen des Kurses fehlgeschlagen."
18       });
19     });
20 });
```

Listing 4.4: Abrufen eines Kurses

In Zeile 5 von Listing 4.4 wird die Funktion `kursModell.findOne` verwendet. `kursModell` ist ein Mongoose-Modell, welches die in Sektion 2.3 beschriebenen Attribute von der Entität "Kurs" in einem Mongoose-Schema beinhaltet, welches in Listing 4.5 gezeigt wird. Es existiert jeweils ein Mongoose-Modell für jede im ER-Diagramm gezeigten Entität. Mit Mongoose-Modellen werden die Anfragen zur Datenbank durchgeführt, in diesem

Fall wird ein Kurs anhand der ID des Kurses und der ID des Nutzers gesucht.

```
const kursSchema = mongoose.Schema({
  // Name vom Kurs.
  name: {type: String, required: true},

  // Semester, in welchem der Kurs stattfindet.
  semester: {type: Number, required: true},

  // Fach, welchem der Kurs zugehört.
  fach: {type: String, required: true},

  // Bestimmt, ob der Kurs bewertet wird, oder nicht.
  // Falls TRUE wird ein Fortschrittsbalken angezeigt.
  bewertet: {type: Boolean, required: true},

  // Gesamte Anzahl an Serien für den Kurs.
  serien_insgesamt: {type: Number},

  // Anzahl Serien, welche bestanden werden müssen.
  serien_benoetigt: {type: Number},

  // Anzahl Serien, welche aktuell bestanden sind.
  serien_bestanden: {type: Number},

  // Anzahl Serien, welche aktuell nicht bestanden sind.
  serien_nicht_bestanden: {type: Number},

  // Zugehöriger Nutzer.
  nutzer: {type: String, required: true}
});
```

Listing 4.5: Mongoose-Schema für Kurse

Ist die Anfrage erfolgreich, wird eine Antwort mit `res` gesendet. Diese Antwort besteht aus einem HTTP Status Code (in diesem Fall 200) und einem JSON-Objekt, welches hier das JSON-Object des gefundenen Kurses ist. Kann der Kurs nicht gefunden werden, wird eine Fehlernachricht mit dem Status Code 404, oder bei einem Fehler auf Seite des Servers wird eine Nachricht mit Status Code 500 zurückgesendet.

4.2.4. Authentifizierung

Einer der Parameter in der in Listing 4.4 verwendeten Funktion `router.get` ist `checkAuth`. `checkAuth` ist eine Middleware, welche bei fast allen Endpunkten aufgerufen wird, um sicherzustellen, dass eine Person nur Zugriff auf die korrekten, bzw. ihre eigenen Ressourcen hat.

```
module.exports = (req,res,next) => {
  try {
    const token = req.headers.authorization.split(" ")[1];
    const decodedToken = jwt.verify(token, tokenKey);
    req.userData = {nutzername: decodedToken.nutzername, nutzerID: decodedToken.
        nutzerID};
    next();
```

```
7   } catch(error) {
8     res.status(401).json({message: "Authentifikation fehlgeschlagen."});
9   }
10 };
```

Listing 4.6: Middleware für die Authentifizierung (`checkAuth`)

Wie in Listing 4.6 zeigt, versucht die Middleware den Authentifizierungstoken abzurufen (Zeile 3) und zu überprüfen (Zeile 4). `tokenKey` ist eine Zeichenfolge, welche vom Administrator der Software in der `.env`-Datei definiert werden muss und als Schlüssel dient bei der Verschlüsselung des Tokens. Konnte der Token erfolgreich überprüft werden, werden Nutzername und ID des Nutzers der Anfrage `req` angefügt, damit diese für die Datenbankanfragen verwendet werden können, wie in Zeile 5 gesehen werden kann. Anschliessend wird der Rest der Anfrage beim Endpunkt erfüllt, aufgerufen durch `next()` in Zeile 6. Im Fall, dass der Token nicht überprüft werden kann, nicht mehr gültig oder nicht korrekt ist, wird die Anfrage abgelehnt und der Status Code 401 ("nicht authentifiziert") wird mit einer Fehlermeldung zurückgesendet.

```
1     bcrypt.compare(req.body.passwort, nutzer.passwort, (err,result) => {
2
3       // Falls Passwort falsch
4       if(!result){
5         return res.status(401).json({message: "Anmeldung fehlgeschlagen."});
6       }
7
8       // Erstellt Token.
9       const token = jwt.sign({nutzername: nutzer.nutzername, nutzerID: nutzer._id},
            tokenKey,{expiresIn: '24h'});
10      return res.status(200).json({
11        token: token,
12        expiresIn: 86400,
13        nutzerID: nutzer._id
14      });
15    })
```

Listing 4.7: Überprüfung des Passworts und Erstellung des Tokens

Authentifizierungstokens werden bei der Anmeldung erstellt. Wie Listing 4.7 zeigt, wird zuerst das eingegebene Passwort mit dem in der Datenbank vorhandenen Passwort verglichen. Angenommen das Passwort wurde korrekt eingegeben, wird anschliessend der Token erstellt, wie in den Zeilen 9 bis 14 ersichtlich ist. Der Token beinhaltet unter anderem den Nutzernamen, die ID des Nutzers, sowie die Gültigkeit des Tokens, und wird mit demselben Schlüssel verschlüsselt, welcher bei der Überprüfung verwendet wird, um den Token zu entschlüsseln.

Zeile 1 von Listing 4.7 nutzt die Funktion `bcrypt.compare`, welche das vom Nutzer bei der Anmeldung eingegebene Passwort mit dem in der Datenbank gespeicherte Passwort vergleicht. Bcrypt ist verantwortlich für die Verschlüsselung der Passwörter, denn die Passwörter sollten nicht in reiner Textform in der Datenbank gespeichert werden, um zu vermeiden, dass die Passwörter ohne Weiteres verwendet werden können, falls ein mutmasslicher Angriff auf die Datenbank stattfindet. Bcrypt wandelt das bei der Registrierung eingegebene Passwort in einen Hash um, welches dann in der Datenbank gespeichert wird

und bei der Anmeldung zum Vergleich mit dem eingegebenen Passwort verwendet werden kann.

4.3. Client

Der Client, bzw. das Frontend, ist die mit Angular erstellte Benutzeroberfläche, welche dem Nutzer angezeigt wird, wenn dieser das Programm verwendet. Bei Angular werden "Components" (zu Deutsch "Komponenten") verwendet, welche zusammen die Benutzeroberfläche bilden. Im für diese Arbeit erstellen Programm bestehen die meisten Komponenten aus:

- **TypeScript-Datei**: Definiert die Komponente als Klasse, welche exportiert wird, damit diese in anderen Komponenten verwendet werden kann. Beinhaltet unter anderem auch die Funktionen, welche die Komponente verwendet.
- **HTML-Datei**: Definiert, welche HTML-Elemente für die Komponente verwendet, bzw. angezeigt werden sollten. Die HTML-Datei kann auch andere Komponenten beinhalten.
- **SCSS-Datei**: SCSS ist eine Erweiterung von CSS mit Sass (Syntactically Awesome Style Sheets). Beide definieren das Aussehen der HTML-Elemente, wobei ersteres mehr Features anbietet wie z.B. Erstellung und Verwendung von Variablen, welches CSS nicht unterstützt [30].

Die Hauptkomponente der Applikation ist die `app`-Komponente, gezeigt in Listing 4.8. Die HTML-Datei dieser Komponente besteht lediglich aus der Navigationsleiste und einem `<div>`-Element, welches das Router Outlet von Angular beinhaltet. Das Router Outlet von Angular ermöglicht abhängig von der URL, welche im Browser des Nutzers angegeben ist, die korrekten Komponenten anzuzeigen.

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

Listing 4.8: HTML-Datei von der `app`-Komponente

Die URLs, welche beim Client verwendet werden, weichen von den URLs des Backend ab. Beispielsweise kann die URL wie folgt aussehen, wenn die Kursliste aufgerufen wird:

`http://localhost:4200/kurse/`

Der Angular Router ladet je nach verwendetem Pfad die jeweilige Komponente, welche in Listing 4.9 angegeben ist. Alle Pfade, ausser bei `login` und `signup` sind geschützt durch `canActivate:[AuthGuard]`, d.h. der Nutzer muss angemeldet sein, um diese Seiten öffnen zu können, ansonsten wird er zur Anmeldeseite weitergeleitet, wo er sich anmelden muss. `AuthGuard` ist eine Funktion welche, im Grunde genommen, beim Authentifizierungsservice nachfragt, ob der Nutzer aktuell authentifiziert ist. Der Authentifizierungsservice wird in der Untersektion 4.3.4 vorgestellt.

```
const routes: Routes = [
  {path: '', component: DashboardComponent, canActivate:[AuthGuard]},
  {path: 'kurse', component: KursListeComponent, canActivate:[AuthGuard]},
  {path: 'kurs/neu', component: KursErstellenComponent, canActivate:[AuthGuard]},
  {path: 'kurs/bearbeiten/:id', component: KursErstellenComponent, canActivate:[
      AuthGuard]},
  {path: 'aufgaben', component: AufgabenListeComponent, canActivate:[AuthGuard]},
  {path: 'aufgaben/:id', component: AufgabenListeComponent, canActivate:[AuthGuard]},
  {path: 'aufgabe/neu', component: AufgabeErstellenComponent, canActivate:[AuthGuard
      ]},
  {path: 'aufgabe/bearbeiten/:id', component: AufgabeErstellenComponent, canActivate:[
      AuthGuard]},
  {path: 'konto', component: AccountComponent, canActivate:[AuthGuard]},
  {path: 'login', component: LoginComponent},
  {path: 'signup', component: SignupComponent}
];
```

Listing 4.9: Definierte Routen, welche jeweils eine Komponente laden

4.3.1. Aufruf und Auflistung von Einträgen

Der Datenaustausch erfolgt immer durch einen Service. In der Applikation existieren Kursservice, Aufgabenservice und Authentifizierungsservice. Die durch die Services bereitgestellten Funktionen, bei welchen ein Datentransfer zwischen Client und Server stattfindet, sind entweder observierbare (Observable) oder abonnierbare (Subscription) Objekte. Beide Varianten werden verwendet, um auf die Daten vom Server warten zu können. Observables können abonniert werden, damit wenn Daten ankommen, diese dann dem Programm mitteilen, damit es die Daten nun verarbeiten kann. Bei Subscriptions muss zwischen Ausgangs- und Zielort ein `Subject` genutzt werden, damit das Ziel benachrichtigt werden kann.

```
  getKurse(){
    return this.httpClient.get<{message: String, kurse: any}>(this.baseURL+'kurse/')
      .pipe(
        map(function (kursData) {
          // Umgestaltung der Daten, damit '_id' zu 'id' umbenannt werden kann.
          return {
            kurse: kursData.kurse.map(function (kurs: {
              name: any;
              semester: any;
              fach: any;
              bewertet: any;
              serien_insgesamt: any;
              serien_benoetigt: any;
              serien_bestanden: any;
              serien_nicht_bestanden: any;
              nutzer: any;
              _id: any
            }) {
              return {
                name: kurs['name'],
                semester: kurs.semester,
                fach: kurs.fach,
                bewertet: kurs.bewertet,
```

```
24              serien_insgesamt: kurs.serien_insgesamt,
25              serien_benoetigt: kurs.serien_benoetigt,
26              serien_bestanden: kurs.serien_bestanden,
27              serien_nicht_bestanden: kurs.serien_nicht_bestanden,
28              nutzer: kurs.nutzer,
29              id: kurs._id
30            }
31          })
32        }
33      })
34    )
35    ...
36    .subscribe((transformedKurs) => {
37      this.kurse = transformedKurs.kurse;
38      this.kursSubject.next({
39        kurse: [...this.kurse]
40      })
41    });
42  }
```

Listing 4.10: Funktion zum Abrufen der Kurse

Listing 4.10 zeigt einen Ausschnitt aus dem Kursservice. Mit der Funktion `httpClient.get` werden die Kurse abgerufen, welche dann auf der Kursseite aufgelistet werden, wie bei der Vorstellung in Untersektion 3.3.1 gezeigt wurde. MongoDB speichert die IDs der Einträge ab mit dem Namen `_id`, welches in `id` umgeändert wird, unter anderem, damit vermieden wird, dass diese ID aus Versehen verändert wird. Um dies zu erreichen, müssen die Kurse durch eine `map()`-Funktion gehen, bei welcher alle Attribute in ein neues Objekt übernommen werden, wobei die ID dann mit `id` statt `_id` gespeichert wird. Nach Abschluss dieser Transformation wird das Ziel mit einem Subject benachrichtigt, welches sich in der angehängten `subscribe`-Funktion befindet (Zeilen 36 bis 40).

Für die eigentliche Auflistung in der Tabelle ist der in Listing 4.11 gezeigte Codeausschnitt verantwortlich. Dies ist eine Tabelle von der Angular Material Bibliothek [3]. Wird die in Listing 4.10 aufgeführte Funktion `getKurse()` aufgerufen, wird die GET-Anfrage an den Server gesendet. Erhält der Service die Daten vom Server, wird die Komponente, welche die Kurse auslisten möchte, durch ein Subject benachrichtigt, und die Daten werden in die Tabelle eingespeist als `MatTableDataSource`.

```
1  <table mat-table matSort [dataSource]="kursDataSource" multiTemplateDataRows (
       matSortChange)="onSort()" *ngIf="!loading">
2    <ng-container matColumnDef="name">
3      <th mat-header-cell *matHeaderCellDef mat-sort-header>Kurs</th>
4      <td mat-cell *matCellDef="let kurs">{{kurs['name']}}</td>
5    </ng-container>
6    <ng-container matColumnDef="semester">...</ng-container>
7    <ng-container matColumnDef="fach">...</ng-container>
8    <ng-container matColumnDef="optionen">...</ng-container>
9    ...
10 </table>
```

Listing 4.11: HTML-Code der Kurstabelle

Listing 4.11 zeigt einen Ausschnitt aus dem HTML-Code, welcher die Tabelle bildet. Mit `[dataSource]=''kursDataSource''` (Zeile 1) werden die erhaltenen Daten in die Tabelle eingespeist. Für jede Spalte wird dann ein Container definiert, welcher aus Header (`th`) und Zeilen (`td`) besteht. `th` stellt die oberste Zeile in der Tabelle dar, welche den Namen der Spalte anzeigt. Im Fall von `th` in Zeile 3 ist es "Kurs". Mit `td` werden dann in jeder Zeile die Daten aus dem Datensatz für die entsprechende Spalte eingefügt. Jeder Kurs hat eine eigene Zeile, und `td` fügt die Daten des jeweiligen Kurses in die Spalte ein. `*matCellDef=''let kurs''` ist vergleichbar mit einer "`for kurs in kursDataSource`"-Schleife.

4.3.2. Erstellung und Bearbeitung von Einträgen

Wo immer der Nutzer Einträge erstellen oder bearbeiten kann, wird ihm in der angezeigten Komponente ein Formular bereitgestellt. Formulare bestehen aus einem oder mehreren Eingabefeldern, welche durch eine FormGroup gehandhabt werden, damit die in ihnen eingegebenen Informationen überprüft und verwendet werden können, oder falls ein bestehender Eintrag bearbeitet werden soll, werden die Eingabefeldern entsprechend mit den existierenden Daten geladen. Bei der Applikation werden FormGroups jeweils im Constructor initialisiert, wobei für jedes Eingabefeld ein FormControl definiert wird, wie Listing 4.12 zeigt.

```
1   this.kursFormular = new FormGroup({
2     nameFC: new FormControl("", [Validators.required]),
3     semesterFC: new FormControl(1,[Validators.required]),
4     fachFC: new FormControl("",[Validators.required]),
5     bewertetFC: new FormControl(false,[Validators.required]),
6     insgesamtFC: new FormControl(1, []),
7     benoetigtFC: new FormControl(1, [])
8   })
```

Listing 4.12: Initialisierung des Formulars für einen Kurs

Für jedes FormControl wird der Standardwert des Feldes, sowie die Liste der Validators, welche die Eingabe überprüfen, definiert. Mit `Validators.required` wird zum Beispiel sichergestellt, dass das Eingabefeld nicht leer ist.

```
1  minSerienValidator():ValidatorFn {
2    return ():ValidationErrors | null => {
3      const insgesamt = this.kursFormular.get('insgesamtFC')!.value;
4      const benoetigt = this.kursFormular.get('benoetigtFC')!.value;
5
6      if (insgesamt < benoetigt) {
7        return {largerThanError: true};
8      } else {
9        return null;
10     }
11   };
12 }
```

Listing 4.13: Im Kursformular verwendete Validatorfunktion

Es ist ebenfalls möglich, einen benutzerdefinierten Validator zu erstellen. Im Formular für Kurse muss beispielsweise sichergestellt werden, dass die Anzahl an zu bestehende Serien nicht grösser sein kann als die gesamte Anzahl an Serien. Hierfür wurde die in Listing 4.13 gezeigte Validatorfunktion definiert, welche `null` (d.h. keinen Fehler) liefert, wenn die Eingabe korrekt ist, oder ansonsten den Fehler `largerThanError` zurückgibt, welches im HTML-Teil der Komponente verwendet werden kann, um eine Fehlernachricht anzuzeigen. Validatorfunktionen können der Validatorliste im FormControl angefügt werden, im Fall vom `minSerienValidator` wird dieser nicht immer benötigt, weswegen eine andere Funktion genutzt wird, welche die Validatorfunktion bei Bedarf hinzufügt oder entfernt.

Sind alle Bedingungen vom Formular erfüllt, ist es dem Nutzer möglich, den Eintrag zu speichern durch Klick auf die Schaltfläche "Speichern". Beim Klick wird dann die Funktion `onAddPost` ausgeführt, welche nochmals überprüft, ob das Formular gültig ist, anschliessend ein Objekt erstellt, welches die Daten aus dem Formular sammelt und schliesslich entweder einen neuen Eintrag erstellt (Zeile 21), oder einen bestehenden Eintrag abändert (Zeile 24), wie in Listing 4.14 gezeigt wird.

```
1  onAddPost(){
2    // Falls Formular ungültig: Speichern abbrechen.
3    if(this.kursFormular.invalid) return;
4
5    this.loading = true;
6
7    const kurs:Kurs = {
8      bewertet: this.kursFormular.value.bewertetFC,
9      fach: this.kursFormular.value.fachFC,
10     id: "",
11     name: this.kursFormular.value.nameFC,
12     nutzer: this.nutzerID,
13     semester: this.kursFormular.value.semesterFC,
14     serien_benoetigt: this.kursFormular.value.benoetigtFC ? this.kursFormular.value.
           benoetigtFC : 0,
15     serien_bestanden: this.kurs?.serien_bestanden ? this.kurs.serien_bestanden : 0,
16     serien_nicht_bestanden: this.kurs?.serien_nicht_bestanden ? this.kurs.
           serien_nicht_bestanden : 0,
17     serien_insgesamt: this.kursFormular.value.insgesamtFC ? this.kursFormular.value.
           insgesamtFC : 0
18   }
19
20   if(this.mode === 'create'){
21     this.kursService.addKurs(kurs);
22   } else {
23     kurs.id = this.kursID;
24     this.kursService.editKurs(kurs);
25   }
26 }
```

Listing 4.14: Die `onAddPost`-Funktion zum Speichern von Kursen

Anzumerken ist, dass `id` im Kurs-Objekt in Listing 4.14 (Zeile 10) leer ist, weil dieses bei der Erstellung des Eintrags nicht verwendet wird. Verantwortlich für die ID ist die Datenbank und nicht der Client, weshalb beim Client eine leere Zeichenfolge angegeben wird. Bei der Änderung eines Eintrags wird die bereits vorhandene ID des Eintrags in

das Feld `id` eingetragen, wie in Zeile 23 ersichtlich ist.

Bei Aufgaben besteht zusätzlich das Problem, dass einige Attribute der Kurs-Entitäten Zahlen sind, welche vom Stand der Bewertung der Serien abhängen. Ändert sich zum Beispiel der Stand der Bewertung bei einer Serie in Kurs A, muss sowohl die Aufgabe wie auch der Kurs selbst aktualisiert werden, da der Kurs die Anzahl an bestandenen und nicht bestandenen Serien speichert.

```
1       this.aufgabenService.editAufgabe(aufgabe)
2           .pipe(
3               concatMap(() => {return this.kursService.updateBewertung(aufgabe.kurs)})
```

Listing 4.15: Teil der `onAddPost`-Funktion zum Speichern von Aufgaben

Dabei muss sichergestellt werden, dass diese Zahlen erst dann aktualisiert werden, wenn die Änderungen an der Aufgabe in der Datenbank gespeichert wurden. Dies wird erreicht durch Nutzung von `concatMap`, wie in Listing 4.15 gezeigt. Wird eine Aufgabe von einem Kurs zu einem anderen verschoben, müssen die Bewertungen der Serien in beiden Kursen neu gezählt werden. Dies wird erreicht durch Hinzufügen einer weiteren `concatMap`, wie jene aus Zeile 3, jedoch mit der ID des alten Kurses.

4.3.3. Löschen von Einträgen

Immer wenn eine Person einen Kurs, eine Aufgabe oder gar ihr Konto löschen möchte, wird ein Dialogfenster geöffnet, welches um Bestätigung bittet. Bei der Implementation muss dabei darauf geachtet werden, welche Auswirkung die Löschung haben kann, bzw. soll. Löscht die Person zum Beispiel eine Aufgabe, muss dann auch die Bewertung des Kurses aktualisiert werden. Löscht sie hingegen einen Kurs, müssen alle darin enthaltenen Aufgaben ebenfalls aus der Datenbank gelöscht werden. Wird das Konto gelöscht, müssen auch sämtliche Aufgaben und Kurse, welche von der Person erstellt wurden, gelöscht werden.

```
1     forkJoin([this.aufgabenService.fullDeleteAll(), this.kursService.fullDeleteAll(),
          this.authService.delete()])
2       .subscribe(() => {
3         this.router.navigate(['/login']);
4         this.authService.logout();
5         this.snackBar.open("Kontoschliessung erfolgreich.", "OK",{
6           duration: 10000,
7           horizontalPosition: "center",
8           verticalPosition: "top"
9         });
10        this.dialogRef.close();
11      });
12    });
```

Listing 4.16: Code für die Löschung des Kontos samt Kursen und Aufgaben

Listing 4.16 enthält als Beispiel einen Ausschnitt aus dem Code, welcher genutzt wird, um ein Konto zu löschen. Obwohl die ID des Nutzers in den Kursen und Aufgaben enthalten

ist, hat das Löschen des Nutzers nicht automatisch zur Folge, dass dessen Aufgaben und Kurse ebenfalls aus der Datenbank entfernt werden. Der Client muss die Kurse und Aufgaben separat löschen. Um sicherzustellen, dass alles gelöscht wurde, bevor der Nutzer zur Anmeldeseite weitergeleitet wird, werden die Aufrufe an die Services in diesem Fall in einem `forkJoin` durchgeführt. Mit dem `forkJoin` erhält `subscribe` erst eine Antwort, wenn alle Services geantwortet haben. Bei der Löschung von Aufgaben wird `concatMap` verwendet, da die Applikation erst die Aufgabe löschen muss, um dann die Bewertung aktualisieren zu können.

4.3.4. Authentifizierung

Wie bereits am Anfang der Sektion erwähnt, sind die meisten Routen durch die AuthGuard-Funktion geschützt, d.h. der Client überprüft, ob der Nutzer angemeldet ist, ansonsten wird der Nutzer zur Anmeldeseite weitergeleitet. Der Client führt nicht die Überprüfung des Authentifizierungstoken durch, denn dies geschieht auf Seite des Servers, beschrieben in Untersektion 4.2.4. Der Client ist lediglich verantwortlich für die Speicherung und Abruf der Anmeldedaten im lokalen Speicher, welcher vom Browser des Nutzers verwendet wird.

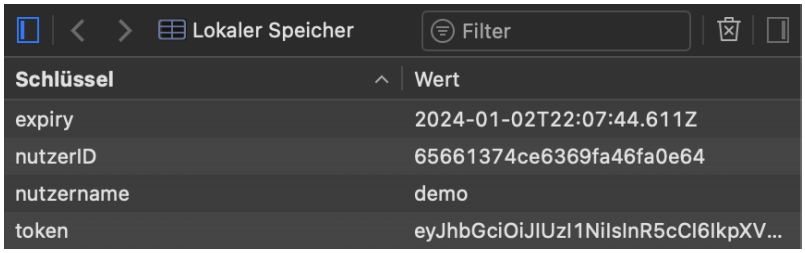

Abbildung 4.3.: Beispiel der im lokalen Speicher gespeicherten Daten

Wie Abbildung 4.3 veranschaulicht, werden die ID des Nutzers, Nutzername, Token und Ablaufdatum (`expiry`) im lokalen Speicher abgelegt, sobald die Authentifizierung auf Seite des Servers erfolgreich war. Wie Listing 4.17 zeigt, wird hierfür die Funktion `localStorage.setItem` genutzt, bei welcher der erste Parameter den Schlüssel und der zweite den entsprechenden Wert setzt.

```
1 private saveAuthData(token:string, expiryDate:Date, nutzerID:string, nutzername:
      string){
2   localStorage.setItem('token',token);
3   localStorage.setItem('expiry',expiryDate.toISOString());
4   localStorage.setItem('nutzerID',nutzerID);
5   localStorage.setItem('nutzername',nutzername);
6 }
```

Listing 4.17: Funktion zum Speichern der Anmeldedaten in den lokalen Speicher

Das Abspeichern dieser Daten in den lokalen Speicher erlaubt es, dass wenn die Applikation aufgerufen wird, diese dann im lokalen Speicher nach den Daten sucht, diese an den Server sendet und schliesslich die Anmeldung automatisch durchführt, gegeben der Token ist gültig. Unabhängig davon, ob die Anmeldung durch das Formular oder automatisch durchgeführt wurde, wird ein Timer gesetzt, welcher zum Zeitpunkt vom Ablauf der

Gültigkeit des Tokens abläuft und dann die Abmeldung des Nutzers erzwingt. Bei der Abmeldung werden die gespeicherten Daten aus dem Browser gelöscht. Dies geschieht ähnlich wie bei der Anmeldung, ausser dass stattdessen die Funktion `localStorage.clearItem` genutzt wird, bei welcher der jeweilige Schlüssel angegeben werden muss. Danach wird der Nutzer zur Anmeldeseite weitergeleitet.

5 Schlussfolgerung

5.1. Rückblick und Resultat

Ziel dieser Arbeit war, ein Programm erstellen, welches Personen nutzen können, um die Aufgaben in ihrem Studium verwalten zu können. Das Resultat der Arbeit ist, dass die Software erfolgreich erstellt wurde und somit auch die Ziele erreicht werden konnten. Mithilfe des MEAN-Stapels wurde eine Webapplikation entwickelt, welche aus Server und Client besteht. Nutzer können ein Konto anlegen, in welchem sie dann Kurse erstellen können, welche die Serien beinhalten. Kurse und Aufgaben können nebst erstellt, auch bearbeitet, sortiert, gefiltert und gelöscht werden. Die Software behält einen Überblick des Standes der Bewertungen, berechnet die noch zu bestehende Anzahl an Serien und stellt es grafisch dar mit einem Fortschrittsbalken. Ausserdem kann jede Person das Kennwort von ihrem eigenen Konto ändern, oder falls dieses nicht mehr benötigt wird, auch löschen. Durch Verwendung von Observables und Subjects konnte erreicht werden, dass die Applikation auf Daten wartet, und erst bei Empfang der Daten fortfährt.

5.2. Verbesserungsmöglichkeiten

Obwohl die Ziele erreicht werden konnten, existieren einige Punkte, bei welchen die Software verbessert werden könnte. Zum Beispiel existiert keine einfache Möglichkeit für den Verwalter der Software, Nutzer bearbeiten oder löschen zu können. Andererseits gibt es im Fall des Verlustes des Kontopassworts auch keine Möglichkeit für den Nutzer, ein neues erstellen zu können. Dies könnte behoben werden durch Implementation eines Administratorkontos, bei welchem der Verwalter die Nutzer verwalten könnte, sowie der Entwicklung eines Passwort-Reset-Tools, bei welchem die Nutzer eine E-Mail-Adresse hinterlegen könnten, um dann eigenständig das Passwort abändern zu können. Für ein verbessertes Erlebnis auf Mobilgeräten, sollte in Erwägung gezogen werden, das Programm entsprechend mit einer mobilen Ansicht zu erweitern.

A
Lizenzierung

B

GitHub-Repository des Projekts

Die Dateien des Projekts sind verfügbar auf der folgenden GitHub-Seite:
`https://github.com/The-Skyless/UniAufgaben-Public`

Die Seite beinhaltet unter anderem den Quelltext des Projekts, sowie eine Anleitung zur Einrichtung des Programms.

Literaturverzeichnis

[1] Andreas Meier. *Introduction pratique aux bases de données relationnelles*. Springer Paris, 2. edition, 2006.

Webressourcen

[2] Introduction to the Angular docs. https://angular.io/docs/ (Letzter Aufruf: 4. Januar 2024).

[3] Angular Material. https://material.angular.io/ (Letzter Aufruf: 4. Januar 2024).

[4] Angular Custom Form Validators: Complete Guide. https://blog.angular-university.io/angular-custom-validators/ (Letzter Aufruf: 10. Oktober 2023).

[5] Angular Material Data Table: A Complete Example (Server Pagination, Filtering, Sorting). https://blog.angular-university.io/angular-material-data-table/ (Letzter Aufruf: 29. Dezember 2023).

[6] Angular ngIf: Complete Guide. https://blog.angular-university.io/angular-ngif/ (Letzter Aufruf: 14. Oktober 2023).

[7] bcrypt - npm. https://www.npmjs.com/package/bcrypt (Letzter Aufruf: 4. Januar 2024).

[8] Was ist ein API-Endpunkt? https://www.cloudflare.com/de-de/learning/security/api/what-is-api-endpoint/ (Letzter Aufruf: 28. Dezember 2023).

[9] Angular 10 Mat Select Deferred Loading Spinner. https://stackblitz.com/edit/angular-10-mat-select-deferred-loading-spinner/ (Letzter Aufruf: 14. Oktober 2023).

[10] dotenv. https://github.com/motdotla/dotenv/ (Letzter Aufruf: 4. Januar 2024).

[11] Express. https://expressjs.com (Letzter Aufruf: 4. Januar 2024).

[12] Free Documentation Licence (GNU FDL). http://www.gnu.org/licenses/fdl.txt (Letzter Aufruf: 6. Januar 2024).

[13] How to create dictionary and add key–value pairs dynamically? https://www.geeksforgeeks.org/how-to-create-dictionary-and-add-key-value-pairs-dynamically/ (Letzter Aufruf: 24. Oktober 2023).

[14] Material Symbols and Icons - Google Fonts. https://fonts.google.com/icons (Letzter Aufruf: 4. Januar 2024).

[15] Mean Stack Tutorial. https://www.javatpoint.com/mean-stack-tutorial/ (Letzter Aufruf: 27. Dezember 2023).

[16] node-jsonwebtoken. https://github.com/auth0/node-jsonwebtoken/ (Letzter Aufruf: 4. Januar 2024).

[17] Color. https://m1.material.io/style/color.html (Letzter Aufruf: 6. November 2023).

[18] GET - HTTP. https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET/ (Letzter Aufruf: 29. Dezember 2023).

[19] How to Use MEAN Stack: Build a Web Application from Scratch. https://www.mongodb.com/languages/mean-stack-tutorial/ (Letzter Aufruf: 4. Januar 2024).

[20] MongoDB. https://www.mongodb.com (Letzter Aufruf: 4. Januar 2024).

[21] Mongoose ODM v8.0.3. https://mongoosejs.com (Letzter Aufruf: 4. Januar 2024).

[22] mongoose-unique-validator. https://www.npmjs.com/package/mongoose-unique-validator (Letzter Aufruf: 4. Januar 2024).

[23] Ng-Matero Extensions. https://github.com/ng-matero/extensions (Letzter Aufruf: 6. November 2023).

[24] Angular - How to resolve CanActivate deprecated in Angular-15 Auth Guard. https://www.youtube.com/watch?v=1FgSlvsywXg (Letzter Aufruf: 5. Oktober 2023).

[25] Node.js. https://nodejs.org/ (Letzter Aufruf: 4. Januar 2024).

[26] API versioning. https://www.postman.com/api-platform/api-versioning/ (Letzter Aufruf: 4. Januar 2024).

[27] Postman API Platform. https://www.postman.com (Letzter Aufruf: 6. Januar 2024).

[28] Arvind Rai. Angular Pattern Validation Example. https://www.concretepage.com/angular-2/angular-2-4-pattern-validation-example (Letzter Aufruf: 4. Januar 2024).

[29] RxJS. https://rxjs.dev (Letzter Aufruf: 6. November 2023).

[30] Sass Guide. https://sass-lang.com/guide/ (Letzter Aufruf: 27. Dezember 2023).

[31] Boris Schäling. Der moderne Softwareentwicklungsprozess mit UML. http://www.highscore.de/uml/usecasediagramm.html (Letzter Aufruf: 26. Dezember 2023).

[32] Was ist der Unterschied zwischen URL, Domain, Subdomain & Hostnamen? https://www.sistrix.de/frag-sistrix/seo-grundlagen/was-ist-der-unterschied-zwischen-einer-url-domain-subdomain-hostnamen-usw (Letzter Aufruf: 28. Dezember 2023).

[33] Marinko Spasojevic. Angular Material Table, Filtering, Sorting, Paging. https://code-maze.com/angular-material-table/ (Letzter Aufruf: 24. Oktober 2023).

[34] Stack Overflow. https://stackoverflow.com/ (Letzter Aufruf: 4. Januar 2024).

[35] Angular HTTP Error Handling. https://www.tektutorialshub.com/angular/angular-http-error-handling/ (Letzter Aufruf: 6. November 2023).

[36] TypeScript Documentation. https://www.typescriptlang.org/docs/ (Letzter Aufruf: 4. Januar 2024).