JOHANNES KEPLER
UNIVERSITÄT LINZ

Eingereicht von
Alexander Stummer

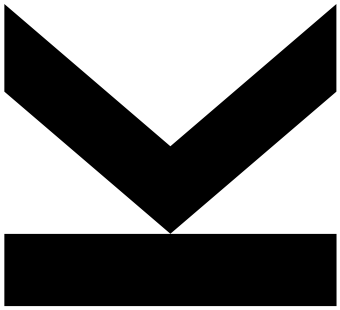
Angefertigt am
Institut für Systemsoftware

Beurteiler / Beurteilerin
Prof. Dr. Dr. h.c. Hanspeter Mössenböck

Mitbetreuung

Juli 2021
(Zur Info: Monat der Abgabe im Prüfungs- und Anerkennungsservice)

Interaktive Visualisierung von Prüfungsdaten



Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Science

im Bachelorstudium

Informatik

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

Inhaltsverzeichnis

1. Einleitung

Im Rahmen dieser Arbeit wurde ein Programm implementiert, das Daten über Lehrveranstaltungsprüfungen aus einer Datenbank ausliest, diese analysiert und Kennzahlen, wie den Notendurchschnitt oder die Durchfallrate einer Lehrveranstaltung berechnet. Diese Daten werden anschließend übersichtlich in Tabellenform angezeigt und die Notenverteilung und die Anzahl nicht bewerteter Studenten anhand von Grafiken dargestellt. Hierzu gibt es verschiedene Ansichten: Einmal die Übersichtsansicht, bei der je nach gewünschtem Studiengang und Semester die Lehrveranstaltungen mit oben angeführten Daten aufgelistet sind. Hier ist es möglich, die angezeigten Daten nach verschiedenen Kriterien zu sortieren (nach Namen, Beurteilungen, Notendurchschnitt, Durchfallrate, Zuteilungen).
Zu jeder Lehrveranstaltung gibt es dann auch eine Detailansicht, bei der Benutzer die Prüfungsdaten nach verschiedenen Kriterien gliedern können, wie Prüfungsdatum, Studienkennzahl und Jahr des Studienbeginns. Diese Kriterien sind beliebig kombinierbar. Dadurch soll sichtbar werden, wie sich die Noten einer Prüfung/Lehrveranstaltung nach diesen aufteilen.

2. Grundlagen

2.1. Grundlegende technische Umsetzung

Für die Umsetzung des Programms wurde ein Ansatz mit Backend und Frontend gewählt, wobei für die Kommunikation zwischen diesen beiden Ebenen eine REST-Schnittstelle [3] implementiert wurde, über welche die benötigten Daten im JSON-Format gesendet werden. Für das Backend wurde Java und das Spring-Framework als Basis benutzt, im Frontend JSP-Seiten und HTML, CSS und JavaScript inklusive dem Vue.js- Framework zum Verwalten der Daten. Genaueres zu diesen Technologien folgt im nächsten Unterkapitel.

2.2. Verwendete Technologien und Frameworks

2.2.1. Spring

Das Open-Source Java-Framework Spring [1] ist heute eines der am meisten genutzten Frameworks für Web-Anwendungen. Deshalb passt es auch sehr gut zu diesem Projekt, da auch hier eine Web-Applikation entwickelt wurde. Das Framework basiert auf dem Prinzip der Dependency Injection. Dies bedeutet, dass die Abhängigkeiten, die ein Objekt benötigt, zentral hinterlegt sind und zur Laufzeit automatisch zugewiesen werden. Grundsätzlich ist ein Spring-Projekt über das Projektverwaltungswerkzeug Maven [4] auch um benötigte Java-Bibliotheken beliebig erweiterbar. Wichtige Bibliotheken, die in diesem Projekt verwendet werden, sind die Java Persistence API (kurz JPA) [5] für Datenbankverwaltung, eine Bibliothek zur Verbindung mit dem MySQL-Datenbankserver [6] und eine Apache-Bibliothek, die einen Tomcat-Server [7] beinhaltet.

2.2.2. Vue.js

Vue.js [2] ist ein JavaScript-Framework, das dabei hilft, Benutzeroberflächen zu bauen. Kern von Vue.js ist es, mit einfacher Syntax Daten deklarativ in das DOM (=Document Object Model) einzubauen. Durch dieses Konzept sind die definierten Daten nun reaktiv und die Werte der deklarierten Felder sind jederzeit dynamisch veränderbar. Werden sie verändert, ändert sich die aktuelle Sicht automatisch. Des Weiteren kann man diese Daten sehr einfach in den HTML-Code einer Seite integrieren, da ein Vue-Objekt sich einfach an einen mit einer ID versehenen HTML-Tag anhängen und diesen vollständig kontrollieren kann. Um Daten in die Ansicht einzufügen, muss das entsprechende Feld zwischen doppelten geschwungenen Klammern stehen. Ein Beispiel wäre:

```
<div id="app">
      {{ lva.name }}
</div>
```

Schreibt man so einen Ausdruck in den HTML-Code, wird {{ lva.name }} zur Laufzeit automatisch durch den Wert, der im Feld “name“ des LVA-Objektes gespeichert ist, ersetzt. Das bedeutet, dass in diesem div-Element immer der aktuelle Wert des Feldes angezeigt wird. Dies vereinfacht das korrekte Darstellen von zu einem gewissen Zeitpunkt geladenen Daten sehr und hilft dabei, Benutzeroberflächen interaktiver zu gestalten.

2.3. Beispieldaten

Um die implementierten Funktionen und Elemente testen zu können, wurden vom Informationsmanagement der JKU anonymisierte Daten über LVA-Klassen, LVA-Abhaltungen und LVA-Noten aus zwei Semestern des Bachelorstudiums Informatik zur Verfügung gestellt.

3. Benutzung

Im folgenden Abschnitt soll ein normaler Programmdurchlauf Schritt für Schritt beschrieben werden. Startet man das Programm und ruft die Seite overview.jsp auf, gelangt man zu einer Startseite, die in Abbildung 1 zu sehen ist.

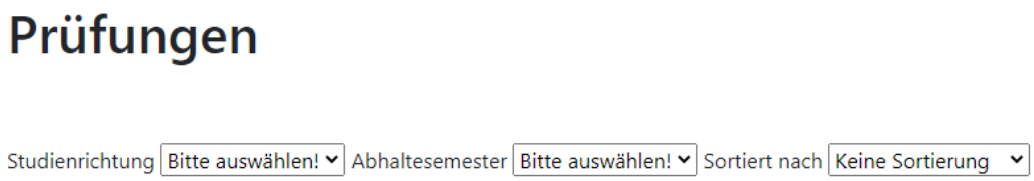


Abbildung 1: Startseite zu Beginn

Wie man sieht, zeigt die Startseite zunächst nur ein Titel und drei Auswahlboxen zu sehen. Aus der Box Studienrichtung kann der Benutzer einen in der Datenbank vorhandenen Studiengang auswählen, von welchem er die Noten sehen möchte, und aus der Auswahl der Semesterbox das gewünschte Semester. Das „Sortiert Nach“-Feld bietet verschiedene Sortierungskriterien für die Lehrveranstaltungen, auf welche später genauer eingegangen wird. Hat der Benutzer nun Studienrichtung und Abhaltesemester ausgewählt, werden die entsprechenden Daten automatisch geladen (Abbildung 2).

Abbildung 2: Übersichtsseite mit geladenen Daten

Diese Sicht zeigt nun alle in der Beispieldatenbank vorhandenen Lehrveranstaltungsdaten des Bachelorstudiums Informatik im Wintersemester 2019 sortiert nach dem Namen an.

Durch Scrollen kann man sich die verschiedensten Lehrveranstaltungen und deren Statistiken ansehen. Durch Klicken auf den „Ausblenden“-Knopf ganz rechts ist es auch möglich, einzelne, eventuell nicht relevante Zeilen auszublenden. Möchte man wieder alle Einträge sehen, gibt es oberhalb der Tabelle den Knopf „Alle einblenden!“.
Ist in einem Teil des Notenverteilungsbalkens keine Anzahl zu sehen, bekommt man diese durch Hovern des Mauszeigers über diesen Balkenteil. Genauso erfolgt die Anzeige des LVA-Titel in voller Länge, da es durchaus sein kann, dass dieser zu lang ist.
Es gibt auch die Möglichkeit, die vorhandenen Zeilen nach verschiedenen Kriterien zu sortieren. In Abbildung 2 sind die Daten nach Namen angeordnet, außerdem kann noch absteigend nach der Anzahl der Zuteilungen, Anzahl der Beurteilungen, nach Durchschnittsnote und Durchfallrate sortiert werden. Möchte man nun genauere Informationen zur Notenverteilung einer spezifischen Lehrveranstaltung erhalten, wird man durch einen Klick auf den dementsprechenden Titel auf eine Detailansichtsseite weitergeleitet (Abbildung 3).

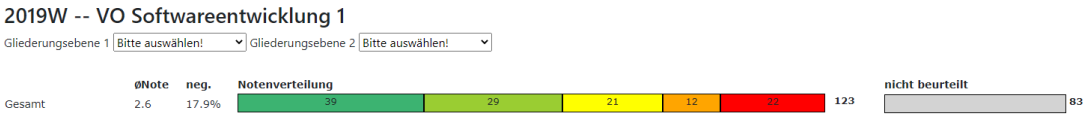


Abbildung 3: Detailseite VO Softwareentwicklung 1

Abbildung 3 zeigt die Seite, auf die ein Benutzer kommt, wenn er in Abbildung 2 auf “VO Softwareentwicklung 1“ klickt. Auf dieser Detailseite sind anfangs nur die Gesamtstatistiken der jeweiligen Lehrveranstaltung zu sehen. Nun kann man sich die Verteilung dieser Noten in verschiedenen Kategorien anzeigen lassen, indem man in den Feldern Gliederungsebene 1 und Gliederungsebene 2 aus je drei Möglichkeiten auswählt. Durch Auswahl in Gliederungsebene 1 kann man sich die Verteilung der Noten auf die Prüfungstermine, auf die verschiedenen Studienrichtungen der Studierenden oder auf das Jahr, in dem ein Studierender sein Studium begann, anzeigen lassen, und diese mit den anderen beiden Kategorien durch Auswahl in Gliederungsebene 2 kombinieren.

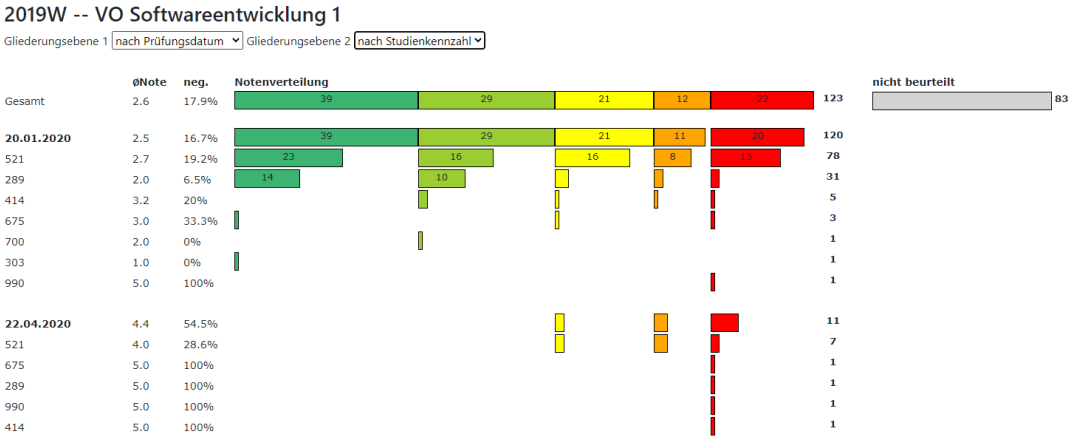


Abbildung 4: Detailseite mit Gliederung

Abbildung 4 zeigt nun die Noten der Lehrveranstaltung VO Softwareentwicklung 1 gegliedert nach Prüfungsdatum und Studienkennzahl. Dies bedeutet, dass angezeigt wird, wie sich die Noten der LVA

auf die einzelnen Prüfungstermine aufteilen und weiters innerhalb der einzelnen Prüfungstermine auf die einzelnen Studienrichtungen. Durch Hovern über die Studienkennzahl ist es möglich, den Namen der zugehörigen Studienrichtung zu sehen. Möchte der Benutzer zurück zur Startseite, kann er dies einfach durch Drücken des Zurück-Buttons im Browser tun und er gelangt so wieder zur vorigen Seite samt aller Einträge in der Tabelle wie in Abbildung 2.

4. Programmstruktur

4.1. Datenbankstruktur

Im Zuge des Projekts wurde auf Basis der im Abschnitt 2.3 beschriebenen Beispieldaten ein MySQL-Datenbankschema aufgesetzt, das aus drei Tabellen besteht: `LVAKlasse`, `LVAAbhaltung` und `LVANote`. Die Felder der zur Verfügung gestellten Textdateien entsprechen den Feldern der einzelnen Datenbanktabellen.

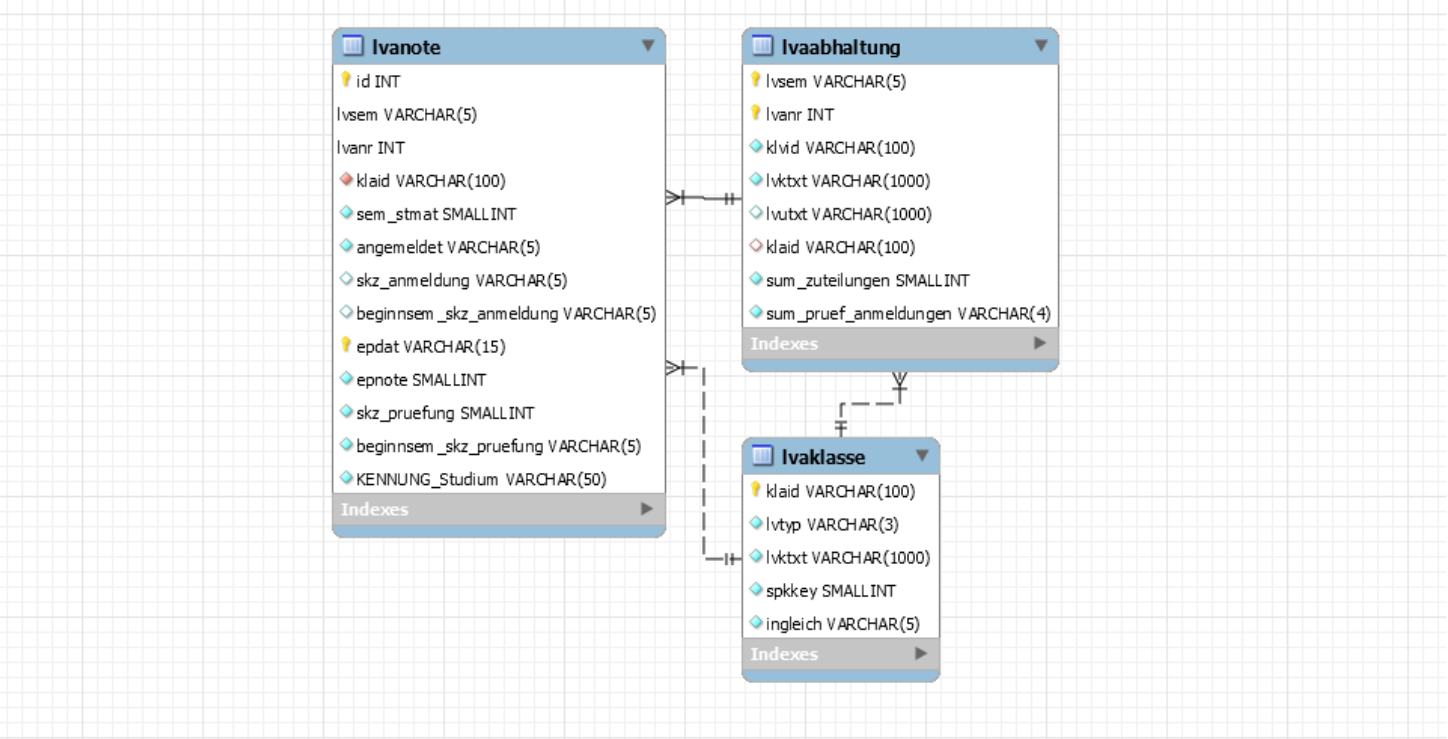


Abbildung 5: Datenbankschema

Wie in Abbildung 5 zu sehen ist, hat `LVAAbhaltung` einen Fremdschlüssel von `LVAKlasse,` und `LVANote` hat von den beiden anderen Tabellen einen Fremdschlüssel. Nach dem Bilden des Schemas wurden die Beispieldateien in CSV-Dateien konvertiert und mithilfe des Tools MySQL-Workbench automatisiert in die Datenbank eingespielt.

Um auf diese Daten aus Java zugreifen zu können, wurden in der application.properties-Datei unter spring.datasource die URL der Datenbank sowie Benutzername und Passwort hinterlegt, um auf diese Zugriff zu erhalten. Die einzelnen Tabellen werden im Java-Code mithilfe der JPA durch Objektrelationales Mapping direkt auf Java-Klassen abgebildet, die so modelliert sind, dass sie die Felder und Beziehungen zwischen den Tabellen widerspiegeln. Realisiert wird es mit Annotationen wie `@Entity` für Klassen, `@Id` für den Schlüssel und `@ManyToOne`, um die Verbindungen zu schaffen. Damit die Felder korrekt zugeordnet werden, müssen diese in Java den gleichen Namen haben wie in SQL. Sollte ein zusammengesetzter Schlüssel vorhanden sein, gibt es zusätzlich eine Klasse, die die

Schlüsselobjekte beinhaltet. Diese wird mit `@IdClass` hinterlegt. Mit der Annotation `@Table`, dem als Attribut der Name der SQL-Tabelle übergeben wird, kann die Tabelle auf die Klasse abgebildet werden wie in Abbildung 6 sichtbar. Genaueres zu den verwendeten Datenklassen folgt im nächsten Unterkapitel und in Kapitel 5.

```java
package jku.examvisualization.models;

import ...

@Entity
@Table(name = "LVAAbhaltung")
@IdClass(LVAAbhaltungID.class)
public class LVAAbhaltung implements Serializable {

    @Id
    private String lvsem;

    @Id
    private int lvanr;

    private String klvid;

    private String lvktxt;

    private String lvutxt;

    @ManyToOne
    @JoinColumn(name = "klaid")
    private LVAClass lvaClass;

    private int sum_zuteilungen;

    private int sum_pruef_anmeldungen;

```

Abbildung 6: Datenklasse mit Mapping-Annotationen

4.2. Java-Klassenaufbau

Das Backend der Applikation wurde in Java implemetiert. In diesem Abschnitt soll auf diese Programmstruktur genauer eingegangen werden.

Die Implementierung besteht aus 12 Java-Klassen und 4 Java-Interfaces. Drei dieser vier Interfaces – nämlich `GradesRepository`, `LVAAbhaltungRepository` und `LVAClassRepository` sind Interfaces, die das Interface CrudRepository erweitern und die Abfragen enthalten, die aus den entsprechenden Datenbanktabellen die benötigten Daten liefern. Die auszuführende SQL-Abfrage wird hier einer `@Query`-Annotation als String übergeben und bei Aufruf der Methode ausgeführt.

Damit das Frontend die benötigten Daten aus dem Backend abfragen kann, ist eine REST-Schnittstelle nötig. In diesem Projekt gibt es sogar zwei solche Schnittstellen. Beide sind mit `@Controller` annotiert: Einmal die Klasse `IndexController`, in der die Namen der verschiedenen JSP-Seiten des Frontend abgefragt und zurückgegeben werden, damit dem Benutzer die richtige Ansicht gezeigt werden kann. Hier gibt es nur zwei Methoden, eine für die Startseite und eine für die Detailansicht. Die zweite Schnittstelle ist die Klasse `ExamController`. Diese Klasse beinhaltet alle prüfungsdatenbezogenen REST-Methoden. Da in dieser Applikation nur Informationen dargestellt werden und nicht veränderbar sein sollen, handelt es sich rein um GET-Methoden. Diese stellen die akkumulierten Statistiken der Noten eines Semesters nach Studienkennzahl, einer einzelnen

Lehrveranstaltung, die unterschiedlichen Aufteilungen der Noten nach Prüfungsterminen, Beginnjahren und Studiengängen, alle vorhandenen Curricula oder die aktuellsten 6 Semester, zu denen es Einträge gibt, zur Verfügung. Die notwendigen Parameter wie Semester, Studienkennzahlen et cetera werden als Parameter der REST-Abfrage im Pfad übergeben.

Als Datenklasse wird neben den in 4.1 beschriebenen Datenbankmodellklassen noch die Klasse `ExamResult` benötigt. Diese hat Felder für die wesentlichen Kennzahlen der Notenverteilung einer Lehrveranstaltung, die in der Klasse ExamResultCalculator berechnet werden.

Die zentrale Klasse des gesamten Programms ist `ExamServiceImpl`. Sie erweitert das vierte Interface (`ExamService`), in dem die wesentlichen Methoden und deren Datentypen definiert sind. `ExamServiceImpl` wird von `ExamController` aufgerufen, wenn hier eine Anfrage eingeht und führt dementsprechend die richtige Methode aus. Sie ist die Verbindung zwischen Controller und Modell/Datenbank. Um die richtigen Daten zu bekommen, hat die Klasse jedes der Repositories als Felder. Diese sind mit `@Autowired` annotiert und werden daher zur Laufzeit mittels Dependency Injection zugewiesen. Außer diesen hat die Klasse noch eine Instanz der Klasse `ExamResultCalculator` und eine Map, die Studienkennzahlen auf die zugehörigen Studiengangsnamen mappt. Grundsätzlich laufen die Methoden alle nach dem selben Schema ab: Zuerst werden die benötigten Daten durch Aufruf einer Querymethode abgefragt. Handelt es sich dabei um Noten, werden diese anschließend dem `ExamResultCalculator` übergeben. Ansonsten kommt es zu einer erneuten Abfrage. Sind diese Prozesse abgeschlossen, wird das Ergebnis an den `ExamController` zurückgegeben, der die Daten anschließend an das Frontend schickt. Die Klasse `ExamServiceImpl` fragt also Daten aus der Datenbank ab, bereitet diese auf und schickt sie an den Controller.

`CoursesOfStudyReader` ist eine weitere wichtige Klasse. Diese liest zum Start der Applikation die Textdatei „courses-of-study.txt“, welches im Resources-Ordner unter static hinterlegt ist. In dieser Datei stehen alle Studienkennzahlen und deren Namen wie in einer .csv-Datei. Die Klasse liest Zeile für Zeile und baut daraus die oben erwähnte Map auf. In Abbildung 7 sind alle beschriebenen Klassen überblicksmäßig zu sehen.

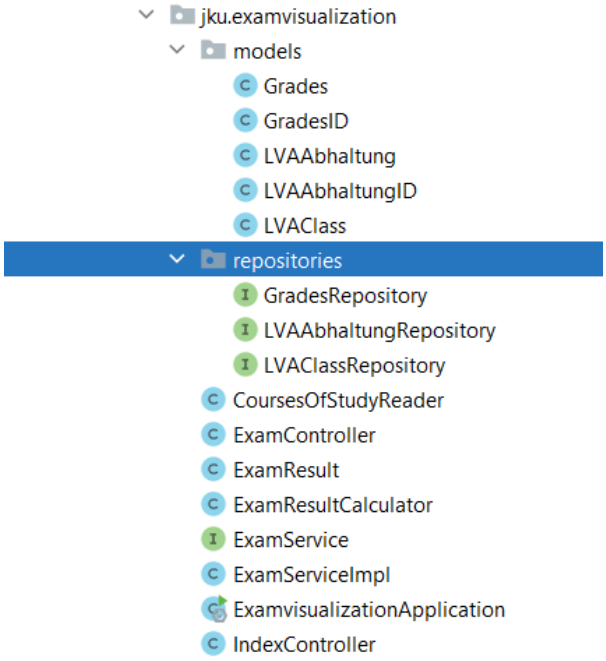


Abbildung 7: Klassenaufbau

4.3. Frontend-Struktur

Das Frontend oder die Benutzeroberfläche der Applikation ist mit Jakarta Server Pages (JSP) realisiert. Die Elemente der Seite sind HTML-Bausteine, deren Layout mit CSS gestaltet ist. Die Interaktivität der Seite wird mit JavaScript implementiert. Hierzu wird das Framework Vue.js verwendet.

Die Benutzeroberfläche besteht aus zwei JSP-Dateien:
Overview.jsp definiert die Startseite, die eine Überschrift, verschiedene Auswahlboxen und eine Tabelle für Lehrveranstaltungen definiert. Auf dieser Seite können mit den in main.js implementierten Funktionen des Vue-Objekts die verschiedenen Lehrveranstaltungen einer Studienrichtung angezeigt und nach verschiedenen Kriterien sortiert werden. Der Benutzer kann auch zur Detailseite einer Veranstaltung gehen.
Tut man dies, gelangt man zur zweiten JSP-Seite, detailsView.jsp. Diesen Übergang regelt die JavaScript-Datei changePage.js. Die Detailseite zeigt die Notenverteilung einer Lehrveranstaltung nach verschiedene Kategorien, die der Benutzer auswählen kann. Informationen, welche Lehrveranstaltung auf der Detailseite angezeigt werden muss, werden vor dem Wechsel in der Session gespeichert, was ebenso ermöglicht, dass beim Zurückkehren zur Startseite die zuvor abgerufenen Daten erneut sichtbar sind. Das Design gewisser Elemente ist in style.css deklariert, zum Teil wurde auch Bootstrap [8] verwendet, welches für viele HTML-Tags zahlreiche Layouts vordefiniert. Für Details zum Layout der Seiten siehe Kapitel 3.

5. Implementierung

5.1. Backend

5.1.1. Java-Datenstrukturen

Im Backend der Applikation gibt es sechs Datenklassen. Drei davon entsprechen dem Schema der Datenbanktabellen des Projekts – die Klassen `Grade`, `LVAClass` und `LVAAbhaltung`. Ergänzend dazu gibt es noch die beiden Klassen `GradesID` und `LVAAbhaltungID`, die die zusammengesetzten Schlüssel der Datenbanktabellen in Java korrekt darstellen und nur die Schlüsselobjekte der Klassen umfassen.

Die letzte und wohl zentralste Datenstruktur ist die Klasse `ExamResult`. Abbildung 8 zeigt die Felder und den Konstruktor der Klasse.

```
1   package jku.examvisualization;
2
3   public class ExamResult {
4
5       private String tag;
6
7       private double averageGrade;
8
9       private double failureRate;
10
11      private int no0f1, no0f2, no0f3, no0f4, no0f5;
12
13      private int totalGrades;
14
15      private int numberNotGraded;
16
17      public ExamResult(String tag) { this.tag = tag; }
20
```

Abbildung 8: Felder der Klasse ExamResult

Das Feld `tag` ist als String möglichst allgemein gehalten, damit diese Klasse sowohl für Kenngrößen der Lehrveranstaltungen als auch für die unterschiedlichen Gliederungen einheitlich verwendet werden kann. Hier werden die Bezeichnungen gesetzt, die später in den Tabellen zu sehen sein sollen – wie Prüfungsdatum, Lehrveranstaltungsname, Studienkennzahl oder auch Jahrgang. Im Konstruktor der Klasse muss immer ein Tag mitgegeben werden.
Die restlichen Felder speichern die benötigten Kenngrößen für einzelne Lehrveranstaltungen, die in der Klasse ExamResultCalculator berechnet werden: Durchschnittsnote, Durchfallrate, Anzahl der einzelnen Noten für den Verteilungsgraphen, Anzahl der Benotungen und Nicht-Benoteten Studenten. In dieser Form werden die Daten auch an das Frontend geschickt.

5.1.2. Wesentliche Algorithmen

Im folgenden Abschnitt werden die wichtigen Algorithmen des Backend im Detail beschrieben.

5.1.2.1. Notenakkumulierung und Kennzahlberechnung

Ein zentraler Algorithmus der Applikation ist in der Methode `calculateResults` der Klasse `ExamResultCalculator` implementiert. In dieser Methode findet die Berechnung aller Kennzahlen einer Lehrveranstaltung statt.
Als Eingabeparameter benötigt dieser Algorithmus:

einen String, dessen Wert das Prüfungsdatum, der Lehrveranstaltungsname, die Studienkennzahl oder auch der Jahrgang ist, welcher im Feld `tag` gespeichert werden soll,
eine Liste, die die Menge aller Noten enthält, auf Basis derer die Kennzahlen berechnet werden, und die Anzahl aller zur Lehrveranstaltung angemeldeten Studierenden als Integer.

Zu Beginn des Algorithmus wird ein neues `ExamResult`-Objekt mit dem Stringparameter erstellt, ein Integer-Array der Größe 5, um die Anzahl der einzelnen Noten zu zählen und eine Variable `sumOfGrades`, die die Summe aller Noten für das Berechnen der Durchschnittsnote speichern soll. Anschließend wird über die Liste der Noten iteriert und jede Note auf `sumOfGrades` aufaddiert und das Array der einzelnen Noten an der entsprechenden Stelle um eins erhöht. Hat die Schleife alle Noten durchlaufen, werden die entsprechenden Felder für die Anzahl an Noten gesamt, Sehr Gut, Gut, Befriedigend, Genügend und Nicht Genügend gesetzt. Die Durchschnittsnote wird ebenso wie die Durchfallrate und die Anzahl nicht-bewerteter Studierender berechnet und im `ExamResult`-Objekt ergänzt. Dieses wird zum Abschluss von der Methode als Rückgabewert an die aufrufende Methode zurückgegeben.

```
public ExamResult calculateResults(String tag, List<Grades> grades, int totalNumberAssigned){
    ExamResult examResult = new ExamResult(tag);
    int[] numberOfGrades = new int[5];
    int sumOfGrades = 0;

    for(Grades grade : grades){
        sumOfGrades += grade.getEpnote();
        numberOfGrades[grade.getEpnote() - 1]++;
    }

    examResult.setNoOf1(numberOfGrades[0]);
    examResult.setNoOf2(numberOfGrades[1]);
    examResult.setNoOf3(numberOfGrades[2]);
    examResult.setNoOf4(numberOfGrades[3]);
    examResult.setNoOf5(numberOfGrades[4]);

    double averageGrade = (double)sumOfGrades/grades.size();
    examResult.setAverageGrade(roundOneDecimal(averageGrade));

    double failureRate = (double) examResult.getNoOf5()/grades.size();
    failureRate = failureRate*100;
    examResult.setFailureRate(roundOneDecimal(failureRate));

    examResult.setTotalGrades(grades.size());

    examResult.setNumberNotGraded(totalNumberAssigned - grades.size());

    return examResult;
}
```

Abbildung 9: Algorithmus zur Berechnung der Kennzahlen

5.1.2.2. Auslesen der LVAs eines Semesters und einer SKZ

Ein weiterer wesentlicher Algorithmus ist in der Methode `getSemesterGrades` in der Klasse `ExamServiceImpl` implementiert. Dieser berechnet alle Kennzahlen aller Lehrveranstaltungen eines Studiengangs in einem Semester und gibt diese zurück.

Als Inputparameter werden ein String mit dem Semester und ein `int`-Wert mit der Studienkennzahl übergeben. Rückgabewert ist eine Liste von `ExamResult`-Objekten. Mithilfe der beiden Eingangsparameter werden durch einen Aufruf der Methode `getAllClassesBySemester` des `LVAClassRepository` alle `LVAClass`-Objekte abgefragt, welche Kurse im geforderten Semester abhalten. Über diese wird anschließend iteriert und alle diese Abhaltungen werden geladen. Hierbei muss zwischen Lehrveranstaltungen mit und ohne Untertitel unterschieden werden, da ansonsten die Noten der

Kurse, die mit unterschiedlichen Untertiteln gehalten werden, zu einem Kurs zusammengefasst werden würden. Kurse wie Übungen hingegen, bei denen mehrere Abhaltungen der identen Veranstaltungen stattfinden, sollen zusammengefasst werden. Im nächsten Schritt werden die Noten dieser Abhaltungen geladen und im Falle von Übungen zusammengefasst. Darüber hinaus erfolgt ein Aufruf der Methode `filterMultipleTries`, die Einträge von Studierenden herausfiltern, die mehrfach vorkommen. Dies ist dann der Fall, wenn eine Person zum Beispiel bei Haupt- und Nebentermin einer Lehrveranstaltung antritt. Die Note des früheren Datums soll in dieser Menge nicht aufscheinen. Diese Filterung erfolgt nur für die Gesamtnotenverteilung, in der Aufteilung auf Prüfungstermine werden beide Noten angezeigt. Zum Abschluss wird der in 5.1.2.1 beschriebene Algorithmus aufgerufen, um aus den erhaltenen Noten die Kennzahlen zu berechnen und das `ExamResult`-Objekt zu erzeugen. Dieses wird zur Rückgabeliste hinzugefügt. Wurden alle Klassen durchlaufen, wird die gesamte Liste zurückgegeben.

5.1.2.3. Notengliederung nach Prüfungsdatum, SKZ und Jahr

Der letzte, wesentliche und der umfangreichste Algorithmus ist jener, der die Noten einer spezifischen Lehrveranstaltung nach verschiedenen Kriterien gliedern kann. So sollen die Noten auf die einzelnen Prüfungsdaten in einem Semester, auf die Studienkennzahlen der angetretenen Studierenden oder deren Jahr des Studienbeginns aufgeteilt werden und auch jede beliebige Kombination zweier dieser soll möglich sein, wie zum Beispiel alle Noten aufgeteilt nach Prüfungsdatum und für jedes Datum nach Studienkennzahlen gegliedert.

Dieser Algorithmus ist auf zwei Methoden der Klasse `ExamServiceImpl` aufgeteilt: `getLVAGradesFiltered`, die die erste Gliederungsebene herstellt und die gesamte Gliederungshierarchie zurückgibt und die private Methode `getGradesBySecondFilter`, die die zweite Ebene regelt.

Die Inputparameter von `getLVAGradesFiltered` sind der Name, der Typbezeichner (z.B. VO, UE, etc.), das Semester und gegebenenfalls der Untertitel der Lehrveranstaltung sowie zwei Strings, die die gewünschten Kriterien für die Gliederungsebenen zugewiesen haben. Rückgabewert ist eine Map, die als Schlüsselobjekte die `ExamResult`-Objekte hat, welche die Kennzahlen nach dem Kriterium der ersten Ebene gliedern, und als zugehörige Werte eine Liste an `ExamResult`-Objekten, die den Werten der zweiten Ebene zu dem jeweiligen Schlüssel entsprechen.

Nachdem die Abhaltungen der Lehrveranstaltung abgefragt wurden, zu welchen die Gliederung gewünscht ist, wird mit einer `switch`-Konstruktion über den String mit dem Gliederungskriterium der ersten Ebene die weitere Vorgehensweise bestimmt. Dieser kann drei Werte annehmen: „date“, „skz“ oder „year“. Handelt es sich um „date“, werden zunächst alle Prüfungsdaten der Lehrveranstaltung in eine Liste geladen, über welche anschließend iteriert wird und für jedes Datum werden die zugehörigen Noten gespeichert und das `ExamResult`-Objekt erstellt. Für „year“ und „skz“ ist die Funktionsweise ähnlich, nur statt der Prüfungsdaten werden bei „year“ die Beginnsemester abgefragt und anschließend zu den Jahren zusammengefasst und bei „skz“ alle Studienkennzahlen. Danach werden ebenso die `ExamResult`s berechnet. Nach diesem Vorgang wird überprüft, ob eine zweite Gliederungsebene angegeben wurde. Wenn ja, wird `getGradesBySecondFilter` aufgerufen. Zum Abschluss wird das Ergebnis in die Rückgabemap eingefügt.

`getGradesBySecondFilter` bekommt als Parameter beide Gliederungskriterien als Strings, einen weiteren String, der den konkreten Wert des Datums oder Jahres enthält, sollte eines dieser das erste Kriterium sein, ein Integer, welcher die aktuell in der ersten Ebene analysierte Studienkennzahl speichert, wenn diese das erste Kriterium ist, und eine Liste, welche alle zur Notenabfrage notwendigen

`LVAAbhaltung`-Objekte enthält. Zurückgegeben wird eine einfache Liste aller berechneten `Exam-Results`.

Der restliche Ablauf dieser Methode ist sehr ähnlich zur oben beschriebenen. Auch hier wird mittels `switch`, über das zweite Kriterium, die Vorgehensweise unterschieden. Es werden wieder benötigte Prüfungsdaten, Studienkennzahlen oder Jahre geladen, allerdings mit anderen Methoden des `Grades-Repository`, da der aktuelle Wert der ersten Ebene in Betracht gezogen werden muss. Anschließend werden die Noten abgefragt und im `ExamResultCalculator` akkumuliert und zur Rückgabeliste hinzugefügt. Diese wird zum Abschluss zurückgegeben.

5.2. Frontend

5.2.1. Vue.js-Objekte und Datenfelder

Das statische Layout des Frontend ist mit HTML und CSS implementiert. Die interaktive Funktionalität des User Interface ist in JavaScript geschrieben. Um die gewünschten Daten, die dynamisch aus dem Backend geladen werden, mit wenig Aufwand anzeigen und verwalten zu können, wird das JavaScript-Framework Vue.js verwendet. Beide JSP-Dateien haben einen HTML-Tag div, der alle anderen Tags im body-Bereich umfasst und als ID „app“ hat. Wird eine Seite geladen, übernimmt ein Vue-Objekt dieses Element und alle Kinderelemente und Funktionen dieses Objekts können für verschiedene Szenarien implementiert und verwendet werden. Diese Objekte enthalten mehrere Felder (Abbildung 10):

```
let overView = new Vue({
    el: '#app',
    data: {lvas: undefined...},
    methods: {...},
    mounted: function () {...}
});
```

Abbildung 10: Vue-Objekt und dessen Felder

- **el**: Diesem Feld wird die ID des HTML-Elements zugewiesen, welches übernommen wird. In diesem Fall ist es für beide ‘#app‘.
- **data**: Dieses Feld ist wiederum ein Objekt mit mehreren Feldern, die alle Daten enthalten. Auf der Startseite sind dies zwei Variablen, „lvas“, welches die verschiedenen Lehrveranstaltungen und deren Kennzahlen als JSON-Array speichert, und „mostGrades“, dem das Maximum der Anzahl an Beurteilten aus allen Kursen zugewiesen wird. Beide Felder sind zu Beginn undefiniert, da diese Daten erst geladen werden, wenn der Benutzer ein Semester und einen Studiengang gewählt hat. Das Objekt der Detailseite benötigt mehr Felder. Alle diese werden zu Beginn mit Nullwerten initialisiert. Der Titel der Detailseite wird in „title“ gespeichert, in „type“der Typ, in „name“ der Name und in „subtitle“ der Untertitel der Lehrveranstaltung. Alle Kennzahlen der Gesamtstatistik des Kurses sind im Objekt „overviewData“, „filteredData“ werden die nach Kriterien gegliederten Noten in einem JSON-Array und „barwidth1-4“ werden die prozentualen Breiten der Abschnitte der Noten 1-4 des Verteilungsgraphen der Gesamtstatistik zugewiesen.
- **methods:** In diesem Abschnitt können Methoden, die zur Aufbereitung oder Beschaffung der Daten gebraucht werden, definiert werden. Beide Objekte bieten Funktionen, welche selektive,

vom Benutzer gewählte Prüfungsdaten in die Seite laden oder aus diesen die Verteilungsgraphen erzeugen und einfügen. Eine detaillierte Beschreibung wichtiger Funktionen folgt in Abschnitt 5.2.2.

- ➔ **mounted:** Das letzte Feld der Objekte ist eine spezielle Art einer Methode. Jene wird automatisch ausgeführt, sobald das Objekt das HTML-Element übernommen hat, lädt diese auf der Startseite die Inhalte der Auswahlboxen für Semester und Studiengang und auf der Detailseite die Kennzahlen des gesamten Kurses.

Es gäbe noch mehr Felder, die für gewisse Applikationen vom Nutzen wären, für beschriebenes Projekt sind die oben beschriebenen aber definitiv ausreichend.

5.2.2. Funktionen

5.2.2.1. Erzeugen der Notenverteilungsbalken

Die wichtigsten Methoden der Vue-Objekte beider Ansichten sind jene, die die Verteilung der Noten und die Anzahl der nicht bewerteten Studierenden einer Lehrveranstaltung grafisch darstellen.

Die grafische Notenverteilung ist in den Methoden `createGradeDistributionGraph` und für die Detailseite zusätzlich in `createPartialGraph` implementiert. Beide bekommen als Inputparameter das aktuell verarbeitete JavaScript-Objekt, welches alle Kennzahlen einer Lehrveranstaltung, oder eines Gliederungskriteriums enthält.
In `createGradeDistributionGraph` wird nun zuerst ein div-Element erstellt, welches als Container für den Balken und die Beschriftungen verwendet wird und eine grundlegende Höhe und Weite innerhalb der Tabellenzelle definiert. In diesen Container kommt nun ein weiteres div-Element, dessen Weite die Breite des Balkens bestimmt und auf Basis der Anzahl der meisten Bewertungen unter allen Lehrveranstaltungen berechnet wird, damit die Proportionen zwischen allen Verteilungsdiagrammen korrekt sind. Anschließend wird in dieses div-Element für die Noten 1-5 jeweils ein weiteres eingefügt, dessen Breite durch den prozentuellen Anteil an der Anzahl aller Bewertungen dieser Lehrveranstaltung bestimmt wird und je nach Note wie folgt gefärbt ist:
Der Sehr-Gut-Abschnitt ist dunkelgrün, Gut ist hellgrün, Befriedigend gelb, Genügend orange und Nicht Genügend rot. Zusätzlich wird für jeden Balken anhand dessen Breite bestimmt, ob die Anzahl an Noten, die dieser repräsentiert, innerhalb dessen angezeigt wird, oder ob diese Information erst bei Hovern der Maus darüber gezeigt werden soll. Wurde dies für jede Note gemacht, wird rechts neben den Verteilungsgraphen noch die Gesamtanzahl an Bewertungen geschrieben und die Methode gibt anschließend das Container-div zurück und dieses wird in die Tabellenspalte eingefügt.

Die Methode `createPartialGraph` zeichnet den Balken nach demselben Prinzip, der Unterschied zu oben beschriebener besteht darin, dass hier die Notenverteilungen für die Gliederungsebenen erfolgen. Hier wird der Balken nicht automatisch durchgängig gezeichnet, sondern abhängig vom prozentuellen Anteil einer Note von der Anzahl der Gesamtstatistik dieser Note beispielsweise an einem Prüfungsdatum dieses Kurses. Damit der Startpunkt des jeweiligen Balkens mit dem des Gesamtverteilungsbalkens übereinstimmt, werden die Felder `barWidth1`, `barWitdth2`, `barWitdth3` und `barWidth4` des `data`-Segments des Vue-Objektes zu Hilfe genommen. Diese speichern die fixierten Breiten, an denen der Balken der jeweiligen Noten beginnt.

Die grafische Darstellung der Anzahl an nicht bewerteten Studierenden erfolgt in der Methode `createNotGradedGraph`. Auch diese bekommt das aktuell bearbeitete Objekt als Eingangsparameter. Das Grundprinzip des Erzeugens des Balkens ist dasselbe wie oben, die Breite des Balkens wird durch

das prozentuelle Verhältnis der Nicht-Bewerteten zu der Anzahl der Bewerteten bestimmt. Die Farbe dieses Balkens ist grau.

5.2.2.2. Befüllen der Tabelle der Gliederungsebenen

Einhergehend mit der zuvor beschriebenen Methode `createPartialGraph` ist die Methode `createDetailsListing`, welche auf der Detailseite dafür sorgt, dass die Gliederungen nach verschiedenen Kriterien korrekt angezeigt werden. Diese ist nur für die Detailseite relevant.

Zu Beginn der Methode werden alle aktuell angezeigten Tabelleneinträge mit Ausnahme der Gesamtstatistik der Lehrveranstaltung gelöscht. Anschließend wird über das `filteredData`-Objekt iteriert, welches bereits geladen wurde und eine Map ist. Für jedes Schlüsselobjekt der Map wird eine neue Reihe in die Tabelle eingefügt und in die Spalten werden die Kennzahlen und der Verteilungsbalken eingefügt. Handelt es sich bei dem Gliederungskriterium um die Studienkennzahl, wird zusätzlich nur die Zahl angezeigt. Der Titel ist zu sehen, wenn man mit der Maus über diese hovert.
Ist eine zweite Gliederungsebene ausgewählt, wird der Vorgang für jedes in der Map unter diesem Schlüssel eingetragene Wertobjekt wiederholt. Eine Leerzeile sorgt dafür, dass die Einträge eindeutig getrennt sind und die Ansicht übersichtlich bleibt.

5.2.2.3. Sortieren der Übersichtstabelle

Das Sortieren der Tabelle der Lehrveranstaltungen nach verschiedenen Sortierkriterien erfolgt in der Methode sortLVATable in main,js. Diese hat keine Eingangsparameter und keine Ausgabewerte. Sie sortiert einzig das Feld „lvas“ des data-Segments des Vue-Objekts. Über das Einbinden dessen in den HTML-Code wird die Ansicht automatisch mit der richtigen Sortierung angezeigt.

Die Methode wird immer aufgerufen, wenn der Wert der „Sortiert nach“-Auswahlbox geändert wird. Zuerst wird der aktuelle Wert dieser Box abgefragt und im Sessionspeicher hinterlegt, damit beim Zurückkehren der Seite von der Detailseite die letzte Sortierung erneut angezeigt wird. Danach wird zwischen 4 Fällen unterschieden:
Fall 1: „none“
In diesem Fall wird die Methode mit dem `return`-Statement sofort beendet, da keine Sortierung vorzunehmen ist.
Fall 2: „assigned“
In diesem Fall soll nach Anzahl zugeteilter Studierender absteigend sortiert werden. Um diese zu bekommen, muss die Anzahl aller Bewertungen und der Nicht-Bewerteten aufsummiert und anschließend verglichen werden.
Fall 3: „tag“
Hier soll nach der Bezeichnung der Lehrveranstaltung alphabetisch sortiert werden. Zusätzlich sollen Vorlesungen und Übungen der gleichen Lehrveranstaltung (zum Beispiel VO und UE Softwareentwicklung 1) hintereinander angezeigt werden. Deshalb wird ein Substring gebildet, der die Bezeichnungen abschneidet und anhand dessen wird verglichen.
Fall 4: Restliche Filter
Der generelle Fall, bei dem Einträge anhand der gewünschten Felder (Anzahl Beurteilungen, Durchschnittsnote, Durchfallrate) verglichen werden.

Die Sortierung an sich erfolgt mit der in JavaScript vorhandenen Array.sort-Funktion, der die jeweiligen Komparatoren übergeben werden.

6. Installationsanleitung

6.1. Anschluss an Echtdaten

Aktuell läuft die Applikation auf Basis eines lokal eingerichteten MySQL-Datenbankservers, auf den die zur Verfügung gestellten Beispieldaten gespielt wurden. Um das Programm an Echtdaten anzuschließen, muss das Datenbankschema exakt jenem entsprechen, das in Abschnitt 4.1 beschrieben ist. Das bedeutet, dass die Tabellen, Feldnamen und Schlüsselobjekte ident mit der Struktur der Beispieldaten sein müssen, damit die Applikation damit arbeiten kann. Damit auf die Daten zugegriffen werden kann, müssen die Zugangsdaten zu der Datenbank in der Datei application.properties in den Feldern spring.datasource.username und spring.datasource.password hinterlegt werden und der Link zur Datenbank unter spring.datasource.url angegeben werden. Wenn kein MySQL-Server verwendet wird, muss zusätzlich unter spring.jpa.properties.hibernate.dialect ein anderer Dialekt angegeben werden, der für das Mapping zwischen Java-Code und SQL verantwortlich ist, und in pom.xml des Projekts muss für einen Konnektor des entsprechenden Datenbankservers eine Abhängigkeit hinzugefügt werden.

6.2. Installation der Applikation

Um die Applikation ausführen zu können, wird eine ausführbare Datei vom Typ .jar benötigt, die auf einem Server installiert werden kann. Damit das Programm funktioniert, müssen zuvor in den JavaScript-Dateien die Links der REST-Anfragen noch geändert werden. Aktuell beginnen diese mit „http://localhost:8080/exam“. Wird die Applikation installiert, muss der localhost:8080-Teil durch die Domain des Servers ersetzt werden.

Grundsätzlich ist die neueste Version der .jar-Datei im target-Ordner des Projektes hinterlegt. Da aber, bevor die Applikation live gehen kann, oben genannte Anpassungen vorgenommen werden müssen, ist ein neuerlicher Build erforderlich. Bei diesem Projekt handelt es sich um ein Maven-Projekt, daher geschieht dies durch Ausführen von folgendem sogenannten Maven-Goal: mvn clean install. Durch diesen Befehl wird ein neuer Build erstellt und der alte im target-Ordner durch den neuen ersetzt. Diese .jar-Datei kann anschließend auf einem Server hinterlegt werden und müsste lauffähig sein.

7. Technische Daten

In diesem Abschnitt werden einige technische und Leistungsdaten des implementierten Programms aufgezeichnet. Zunächst wird die Anzahl an Einträgen in der Datenbank in folgender Tabelle gezeigt:

Tabelle Einträge Datenbank

DB-Tabelle	Anzahl Einträge in Reihen
LVAKlasse	122 Reihen
LVAAbhaltung	302 Reihen
LVANote	10273 Reihen

Tabelle 1: Einträge Datenbank (in Reihen)

Als Nächstes werden die Laufzeiten zentraler Funktionen des Backend aufgelistet. Die Funktionen wurden ausgewählt, da sie die meisten Daten laden und verarbeiten müssen.

Tabelle Laufzeiten

Funktion	Laufzeit (in ms)
Alle Noten eines Semesters und einer Studienrichtung laden	530.7 ms
Laden zweier Gliederungsebenen Detailansicht	61.8 ms

Tabelle 2: Laufzeiten

Wie man sehen kann, verarbeiten die Algorithmen die Daten schnell und effizient.
In der letzten Tabelle wird die Anzahl der Codezeilen aufgeteilt nach den verwendeten Programmiersprachen gelistet.

Tabelle Codezeilen

Sprache	Codezeilen
Java	1354
HTML	136
JavaScript	676
CSS	27

Tabelle 2: Laufzeiten

Der Großteil des Programms wurde mit Java im Backend und JavaScript und HTML im Frontend implementiert. CSS wurde für das Layout verwendet.

8. Fazit

Alles in allem ist es gelungen, im Rahmen dieses Projektes eine Applikation zu implementieren, die den gesamten Anforderungen entspricht und die gewünschten Funktionen bietet. Bis auf kleinere Probleme bei einzelnen Implementierungsschritten verlief das Projekt reibungslos. Die Applikation könnte durch verschiedene Erweiterungen verbessert und je nach Bedarf an Anforderungen für einen Echtbetrieb angepasst werden. Zum Beispiel könnte ein Login-System hinzugefügt werden, damit jeder Benutzer auch nur die Studiengänge und Noten anschauen kann, welche er selbst besucht. Des Weiteren könnte man für Lehrveranstaltungsleiter eine Kommentar- und Freigabefunktion ergänzen, in der diese Kommentare zur Notenverteilung ihres Kurses posten können oder vor dem Anzeigen der Notenstatistiken eine Freigabe erteilen müssen.
Es gibt noch viele weitere Möglichkeiten, diese Applikation zu verbessern, jedoch stellt die im Rahmen dieses Projektes ausgearbeitete Version die wesentlichsten Funktionen zur Verfügung. Auch das Layout kann sicher noch besser gestaltet werden, nichtsdestotrotz reicht es aus, um die Funktionalität des Programms zu präsentieren. Zum Abschluss lässt sich sagen, dass das Projekt durchaus ein gelungenes war, bei dem viel dazugelernt und das Ziel erfüllt wurde.

9. Literaturverzeichnis

1. https://spring.io/, abgerufen am 15.6.2021

2. Introduction, What is Vue.js? , abgerufen am 15.6.2021 von https://vuejs.org/v2/guide/

3. Was ist eine REST-API ?, abgerufen am 12.7.2021 von https://www.cloudcomputing-insider.de/was-ist-eine-rest-api-a-611116/

4. https://maven.apache.org/#, abgerufen am 12.7.2021

5. Introduction to the Java Persistence API, abgerufen am 12.7.2021 von https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html

6. MySQL Connector/J 8.0 Developer Guide, abgerufen am 12.7.2021 von https://dev.mysql.com/doc/connector-j/8.0/en/

7. https://mvnrepository.com/artifact/org.apache.tomcat.embed/tomcat-embed-jasper , abgerufen am 12.7.2021

8. Introduction – Bootstrap v5.0, abgerufen am 12.7.2021 von https://getbootstrap.com/docs/5.0/getting-started/introduction/