JOHANNES KEPLER
UNIVERSITÄT LINZ

Author / Eingereicht von
Oktay Akgül, BSc
K01647998

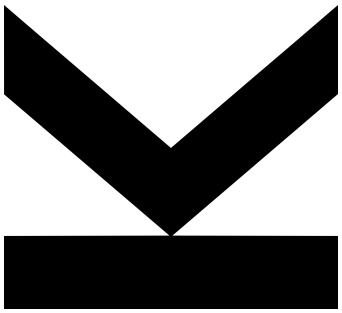
Submission / Angefertigt am
Institut für Systemsoftware

Thesis Supervisor / First Supervisor / BeurteilerIn / ErstbeurteilerIn / ErstbetreuerIn
Prof. Dr. Dr. h.c.
Hanspeter Mössenböck

September 2022

Exam Analytics

Visualisierung von Prüfungsergebnissen



Masterarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Masterstudium

Computer Science

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

JOHANNES KEPLER
UNIVERSITÄT LINZ

Eidesstattliche Erklärung

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

Linz, September 2022

JOHANNES KEPLER
UNIVERSITÄT LINZ

Kurzfassung

Die Johannes Kepler Universität versucht laufend, den Studierenden die maximale Qualität der Lehre anzubieten. Hierfür dienen in erster Linie die Prüfungsergebnisse, aus denen analytische Kennzahlen gewonnen und auf deren Basis Entscheidungen getroffen werden können. Aktuell gibt es keine Möglichkeit, die Prüfungsergebnisse in einfacher Form zu analysieren. Deswegen wurde ein webbasiertes Visualisierungstool erschaffen, welches das Ziel verfolgt, analytische Kennzahlen, wie z.B. den Notendurchschnitt, die Durchfallrate, den Notenspiegel, etc. optimal zu visualisieren. Zusätzlich wird die Möglichkeit geboten, die Prüfungsergebnisse der einzelnen Lehrveranstaltungen nach bestimmten Kriterien, wie z.B. Studienkennzahl, Prüfungsdatum oder Jahrgang eines Studierenden zu gliedern und visualisieren. Das Tool soll die Interpretation der Ergebnisse für die Vorsitzenden der Studienkommission, die Lehrenden, die Vizerektorin/den Vizerektor für Lehre und das Auslandsbüro erheblich erleichtern. Dadurch kann die Qualität der Lehre verbessert und die Curricula effizienter optimiert werden.

JOHANNES KEPLER
UNIVERSITÄT LINZ

Abstract

The Johannes Kepler University aims to provide the maximal quality of studies for the students. The decisions, which lead to optimal study quality, may be based on key figures, gained from the exam evaluations. Currently, there is a lack of support for analyzing exam evaluations, for which a web-based visualization tool was developed. Basically, the tool is designed for visualizing key figures of a course, such as the average grade, failure rate, overview of grades, etc., for the heads of the study commissions, the Vice-Rector of Academic Affairs, the lecturers and the international office. In addition, there is the opportunity, for those user groups, to group and visualize the data from examinations by different attributes, such as the study, date of examination or begin year of a student. With the help of the visualized data, it becomes more efficient to analyze the courses. Hence, it is easier to increase the quality of teaching and to optimize the existing curricula.

JOHANNES KEPLER
UNIVERSITÄT LINZ

Inhaltsverzeichnis

JOHANNES KEPLER
UNIVERSITÄT LINZ

Abbildungsverzeichnis

JOHANNES KEPLER
UNIVERSITÄT LINZ

Tabellenverzeichnis

JOHANNES KEPLER
UNIVERSITÄT LINZ

Auflistungsverzeichnis

JOHANNES KEPLER
UNIVERSITÄT LINZ

Abkürzungsverzeichnis

AJAX Asynchronous JavaScript and XML
API Application Programming Interface
CGI Common Gateway Interface
CSS Cascading Stylesheets
CSV Comma-separated values
DB Datenbank
DOM Document Object Model
ER Entity-Relation
EU Europäische Union
GUI Graphical User Interface
HTML Hyper Text Markup Language
HTTP Hypertext Transfer Protocol
IdP Identity Provider
IM Informationsmanagement
JKU Johannes Kepler Universität
JS JavaScript
JSON JavaScript Object Notation
LVA Lehrveranstaltung
ÖH Österreichische Hochschülerschaft
SAML Simple Assertion Markup Language
SP Service Provider
SQL Structured Query Language
SSO Single-Sign-On
Stuko Studienkommission
URI Uniform Resource Identifier
URL Uniform Resource Locator
VR Vizerektorin/Vizerektor für Lehre
XML Extensible Markup Language

JOHANNES KEPLER
UNIVERSITÄT LINZ

1 Einleitung

Die Johannes Kepler Universität (JKU) arbeitet ständig daran, die Qualität der Lehre nachhaltig zu verbessern, damit den Studierenden die bestmögliche Bildung angeboten wird. Es werden laufend Curricula optimiert, Feedback von Lehrveranstaltungen (LVAs) evaluiert und Vorschläge von der ÖH zur Verbesserung der Lehre berücksichtigt. Für eine Verbesserung braucht es Daten, und wichtige Daten zu einer LVA stellen die Prüfungsergebnisse dar, aus denen objektive Kennzahlen, wie z.B. der Notendurchschnitt, die Durchfallrate oder die Anzahl der nicht beurteilten Studierenden, gewonnen werden können. Fallen Prüfungsergebnisse für LVAs eines Studiums negativer als erwartet aus, muss die Studienkommission (Stuko) diese Fälle besonders genau analysieren, um der Ursache auf den Grund zu gehen und rechtzeitig entsprechende Maßnahmen zu setzen. Darüber hinaus können Lehrende bestimmte Kennzahlen der Prüfungsergebnisse nutzen, um ihre Lehre zu verbessern und dadurch die Qualität zu steigern. Zusätzlich ist es für die Vizerektorin/den Vizerektor für Lehre (VR) durchaus interessant zu wissen, wie die LVAs in den einzelnen Studien im Durchschnitt abschneiden, um als höhere Instanz ebenfalls Entscheidungen zu treffen, die die Qualität der Lehre verbessern. Eine andere Instanz, die ebenfalls Nutzen von den Kennzahlen der Prüfungsergebnisse ziehen kann, ist das Auslandsbüro. Durch eine neue Richtlinie der Europäischen Union (EU) müssen nun alle europäischen Universitäten eine Erfolgsbilanz eines Studienjahres der EU-Kommission übermitteln. Dafür sind vor allem Kennzahlen, wie der Notendurchschnitt, die Notenverteilung, die Anzahl der abgelegten Prüfungen, etc. für einzelne Studienrichtungen relevant, die für heimische und Austauschstudierende vorgelegt werden.

Damit die Stuko, die Lehrenden, die/der VR und das Auslandsbüro Prüfungsergebnisse besser interpretieren können, wurde ein webbasiertes Visualisierungstool, namens *xam-viz*, entwickelt. Unter einer Visualisierung versteht man, u.a. die Anzeige einer Notenverteilung, des Notendurchschnitts, der Durchfallrate und vielen weiteren Eckdaten.

Das Tool ermöglicht es, für bestimmte Benutzergruppen unterschiedliche Funktionalitäten anzubieten. Im Wesentlichen werden vier Benutzergruppen unterschieden: Lehrende/r, Stuko-Vorsitz, VR und Auslandsbüro. Für jede Benutzergruppe gibt es andere Visualisierungsmöglichkeiten bzw. andere Sichten. Es gibt auch die Möglichkeit, die Prüfungsergebnisse einzelner LVAs nach bestimmten Kriterien zu gliedern bzw. zu gruppieren, damit eine detaillierte Aufschlüsselung über die Ursache der Ergebnisse angeboten wird.

Die Prüfungsergebnisse von LVAs stellen sensible Daten dar, die nicht für die Öffentlichkeit vorgesehen sind. Der Grund dazu ist, dass negative Prüfungsergebnisse dazu verleiten können, die Lehrenden an den Pranger zu stellen. Deswegen spielt das Thema Datenschutz eine wichtige Rolle für dieses Tool. Als erste Maßnahme, die zum Datenschutz beiträgt, ist die anonymisierte Auswertung der Noten von Studierenden. Es werden nur solche Daten ausgewertet, die keine Rückschlüsse auf einzelne Studierende ermöglichen. Eine weitere Maßnahme stellt das Berechtigungssystem dar. Eine Benutzergruppe muss explizit für deren Sicht berechtigt werden, damit das Tool verwendet werden kann. Dadurch kann eine Benutzergruppe zu keinem Zeitpunkt andere Sichten benutzen und dadurch unerlaubt andere Daten einsehen, die nicht für diese bestimmt sind. Die letzte Maßnahme dient zum Schutz der Lehrenden, die es erlaubt, der Veröffentlichung der Prüfungsergebnisse gegenüber dem Stuko-Vorsitz zu widersprechen oder die Ergebnisse zu kommentieren. Durch einen Widerspruch kann der Stuko-Vorsitz diese Daten nicht mehr einsehen bzw. durch einen Kommentar die Ergebnisse besser beurteilen.

2 Anforderungen

Für die Applikation, *xam-viz*, gibt es eine Reihe von funktionalen bzw. nicht-funktionalen Anforderungen, die in diesem Kapitel näher beschrieben werden. Die funktionalen Anforderungen werden in allgemeine und benutzergruppenorientierte Anforderungen unterteilt.

2.1 Funktionale Anforderungen

Allgemein

1. Die Applikation muss alle Benutzer/-innen über Shibboleth (das an der JKU verwendete Authentifizierungssystem) authentifizieren.
2. Die Applikation muss verschiedene Sichten für die definierten Benutzergruppen bereitstellen.
3. Die Applikation muss die Prüfungsergebnisse einer LVA gliedern/gruppieren können.
4. Die Applikation muss an die Echtdaten der JKU angebunden werden.

Benutzergruppe Lehrende/r

1. Die Applikation muss die Prüfungsergebnisse der LVAs pro Semester visualisieren.
2. Die Applikation muss die Möglichkeit bieten, der Weiterleitung von Prüfungsergebnissen der LVAs an den Stuko-Vorsitz zu widersprechen.
3. Die Applikation muss die Möglichkeit bieten, die Prüfungsergebnisse der LVAs zu kommentieren.

Benutzergruppe Stuko-Vorsitz

1. Die Applikation muss die Prüfungsergebnisse der LVAs pro Semester und Studiengang visualisieren.
2. Die Applikation muss Filter -und Sortiermöglichkeiten anbieten.
3. Die Applikation darf die widersprochenen LVAs nicht anzeigen.
4. Die Applikation muss den Kommentar einer LVA anzeigen.

Benutzergruppe VR

1. Die Applikation muss die Prüfungsergebnisse der LVAs pro Semester und Studiengang visualisieren.
2. Die Applikation muss Filter- und Sortiermöglichkeiten anbieten.
3. Die Applikation muss die widersprochenen LVAs anzeigen.
4. Die Applikation muss den Kommentar einer LVA anzeigen.

Benutzergruppe Auslandsbüro

1. Die Applikation muss die Prüfungsergebnisse der LVAs von Austauschstudierenden bzw. JKU-Studierenden für bestimmte Semesterintervalle visualisieren.
2. Die Applikation muss Filter- und Sortiermöglichkeiten anbieten.

2.2 Nicht-funktionale Anforderungen

Benutzbarkeit

1. Die Applikation muss intuitiv bedienbar und leicht zu navigieren sein.

Performance

1. Die Applikation muss die Daten innerhalb von 30 Sekunden den Lehrenden, den Stuko-Vorsitzenden und der/dem VR anzeigen.
2. Die Applikation muss die Daten innerhalb von zwei Minuten dem Auslandsbüro anzeigen.

Wartbarkeit

1. Die Applikation sowie der Source-Code müssen ausführlich dokumentiert sein.
2. Die Stuko-Vorsitzenden und die von ihnen betreuten Studien müssen aktualisiert werden können.

3 Architektur

Die Architektur von *xam-viz* besteht aus dem Zusammenspiel von mehreren Komponenten. Abbildung 3.1 stellt eine grobe Übersicht der Architektur dar, wo die Pfeile die Richtung des Datenflusses anzeigen.

Wie im Architekturdiagramm ersichtlich, werden alle Anfragen über den Browser zuerst durch Shibboleth authentifiziert. Je nachdem, ob bereits eine Shibboleth-Session besteht oder nicht, werden die Benutzer/-innen entweder um die Eingabe der Benutzerdaten in der Shibboleth-Maske gebeten oder Shibboleth leitet die Anfrage an die Web-Applikation *xam-viz* weiter, die in Express.js implementiert ist. Allerdings gibt es Anfragen, für die keine Authentifizierung erforderlich ist, wie etwa für die Rollenauswahl oder für die Initiierung des Logins. Die Web-Applikation bezieht die Daten über Prüfungsergebnisse aus einer Oracle Datenbank (DB) und generiert einen HTML-Code mit der Hilfe von PUG-Templates. Dieser HTML-Code wird schlussendlich in den Browser der Benutzer/-innen geladen.

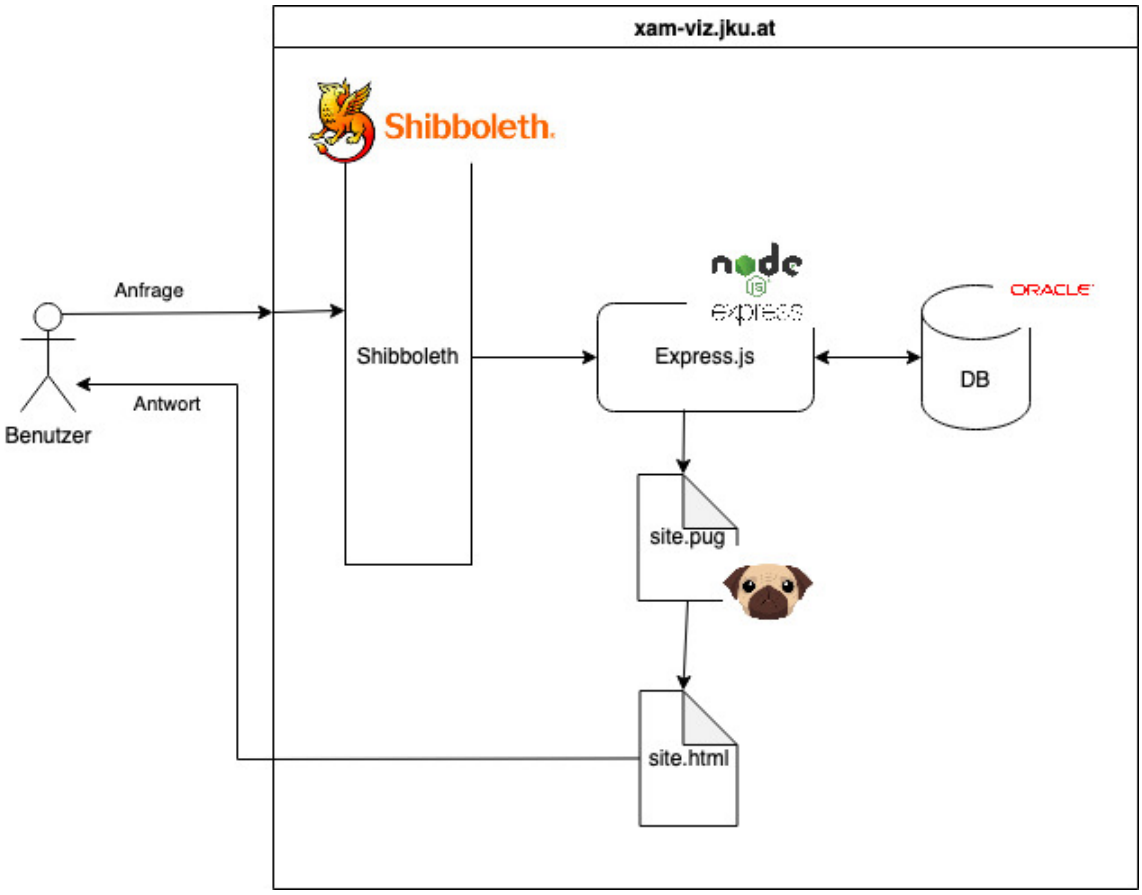


Abbildung 3.1: Architekturdiagramm

4 Datenmodell

Die Daten, mit denen *xam-viz* operiert, werden aus der JKU-internen Oracle-DB bezogen. Für diesen Zweck werden fünf Tabellenfunktionen, eine View und eine Tabelle vom Informationsmanagement (IM) zur Verfügung gestellt. Abbildung 4.1 zeigt ein ER-Modell der Daten.

Im ER-Modell sind die Tabellenfunktionen mit den Input-Parametern gekennzeichnet. Die fünf Tabellenfunktionen gliedern sich wie folgt auf:

- **LVANoten bzw. LVANotenMWL**
  Die Funktion liefert die Noten einer bestimmten LVA-Abhaltung eines Studiums (LVANoten) oder eines Lehrenden in einem Semester (LVANotenMWL).
- **LVAAbhaltung bzw. LVAAbhaltungMWL**
  Die Funktion liefert die Abhaltungen von LVAs für ein Studium (LVAAbhaltung) oder für Lehrende in einem Semester (LVANotenMWL).
- **LVAMobility**
  Liefert die Prüfungsergebnisse für ein Studium in einem Semesterintervall und inkludiert bei Bedarf die Auslandsstudierenden mit.

Zusätzlich gibt es die View **VEXAMDVCURR1**, die eine Auflistung der existierenden Curricula liefert und die Tabelle **LVAAbhaltungDV**, in die die widersprochenen LVAs gespeichert werden können.

Die Abhaltungen und Noten für die Sicht der Stuko-Vorsitzenden bzw. des/der VR werden für einen bestimmten Studiumschlüssel (SPKKEYS) und ein bestimmtes Semester (SEM) aus **LVAAbhaltung** bzw. **LVANoten** in diesem Studium und Semester abgefragt. Danach werden die Abhaltungen und die Noten mithilfe dem Semester (LVSEM), der Nummer (LVANR) und dem Klassencode (KLAID) miteinander verknüpft. Zusätzlich

werden die Kommentare und Widersprüche mit der Nummer (LVANR) und dem Semester (LVSEM) einer Abhaltung aus **LVAAbhaltungDV** geladen. Die Prozedur ist für die Sicht der Lehrenden identisch. Nur werden die Abhaltungen und Noten aus **LVAAbhaltungMWL** bzw. **LVANotenMWL** mit der AK-Nummer (VLOGIN) und dem Semester (SEM) abgefragt. Für die Sicht des Auslandsbüros werden die Daten ausschließlich aus **LVAMobility** für ein bestimmtes Semesterintervall (VSEMVON und VSEMBIS) geladen.

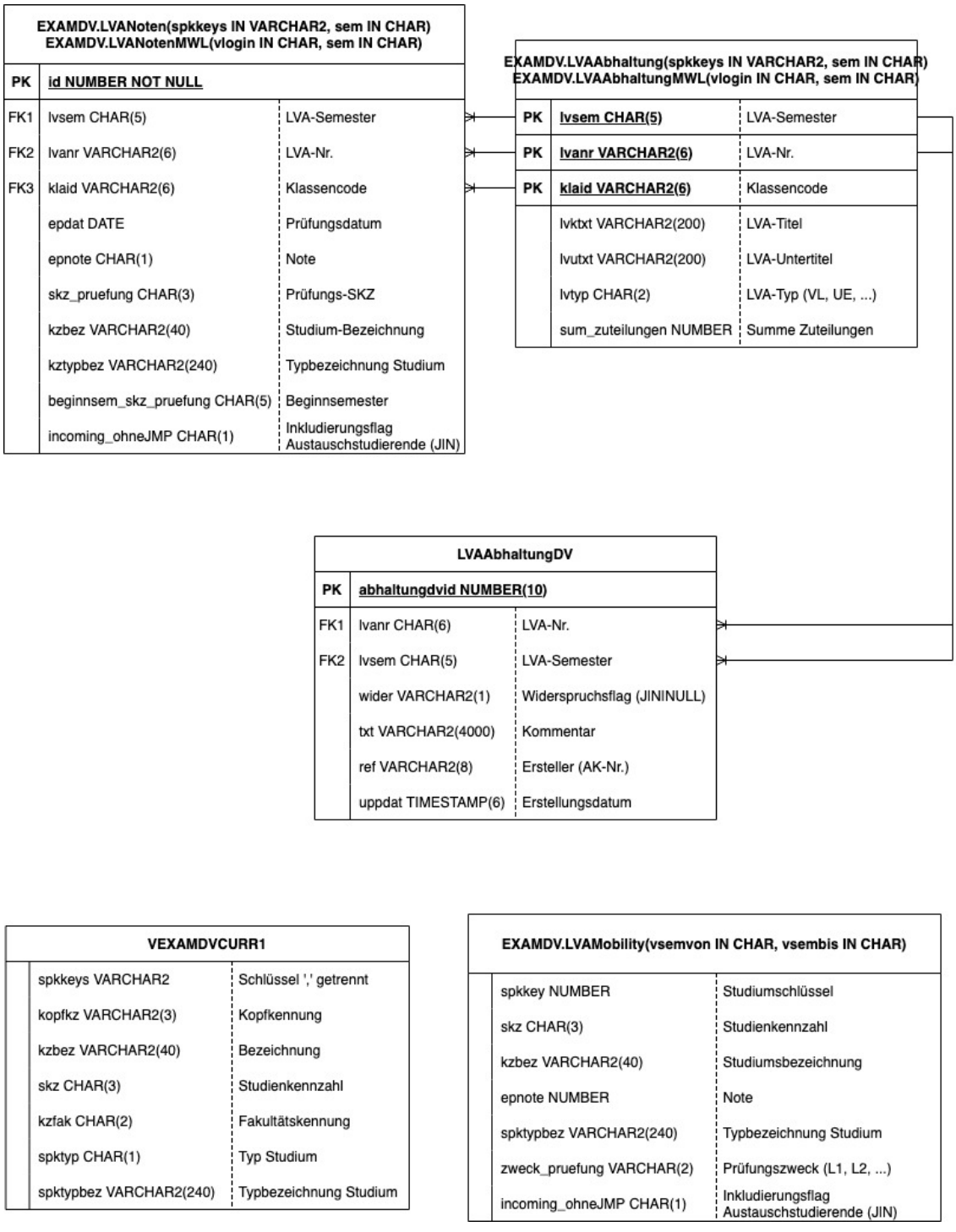


Abbildung 4.1: ER-Modell

5 Authentifizierung & Autorisierung

Die *xam-viz*-Applikation operiert mit vertraulichen Daten, weshalb die Authentifizierung und Autorisierung der Benutzer/-innen eine wichtige Rolle spielt. Bevor die Web-Applikation verwendet werden kann, erfolgt eine Authentifizierung der Benutzer/-innen mittels Shibboleth. Sobald die Identität gewährleistet ist, muss zunächst der Zugriff auf eine bestimmte Sicht in der Applikation autorisiert werden. In anderen Worten muss z.B. der Zugriff der Lehrenden auf die VR-Sicht verhindert werden. Dafür wird ein Rollensystem angewandt, wobei Benutzer/-innen auch mehrere Rollen besitzen können. Für *xam-viz* sind folgende Rollen definiert:

- **teacher**
  Diese Rolle autorisiert den Zugriff auf die Sicht der Lehrenden. Die Zuordnung der Lehrenden zu dieser Rolle erfolgt durch Shibboleth.
- **stuko**
  Diese Rolle autorisiert den Zugriff auf die Sicht der Stuko-Vorsitzenden. Die Zuordnung erfolgt hier über eine eigene Zugriffsliste.
- **vr**
  Diese Rolle autorisiert den Zugriff auf die Sicht der/des VR. Die Zuordnung erfolgt hier über eine eigene Zugriffsliste.
- **international-office**
  Diese Rolle autorisiert den Zugriff auf die Sicht des Auslandsbüros. Die Zuordnung erfolgt hier über eine eigene Zugriffsliste.

5.1 Shibboleth

Shibboleth[1] ist das Authentifizierungssystem, welches an der JKU verwendet wird. Bei dieser Technologie handelt es sich um ein sogenanntes Single-Sign-On (SSO) System. Die Idee eines SSO-Systems ist es, eine Identität für mehrere Systeme benutzen zu können.

Für SSO-Systeme gibt es zwei führende Protokolle, nämlich OAuth2 und Simple Assertion Markup Language (SAML), wobei letzteres von Shibboleth verwendet wird. Das SAML-Protokoll[2] definiert die Kommunikation von Sicherheitsinformationen zwischen zwei Entitäten im XML Format über das HTTP-Protokoll. Die zwei Entitäten werden als Service Provider (SP) und Identity Provider (IdP) bezeichnet (siehe Abbildung 5.1). Die Nachrichten, die zwischen dem SP und IdP ausgetauscht werden, werden im Fachjargon *Assertions* genannt. Solche *Assertions* beinhalten die Sicherheitsinformationen, die die Benutzer/-innen eindeutig identifizieren und zusätzliche Attribute beinhalten können. Ein Beispiel für *Assertions* wird in der Auflistung 5.1 gezeigt.

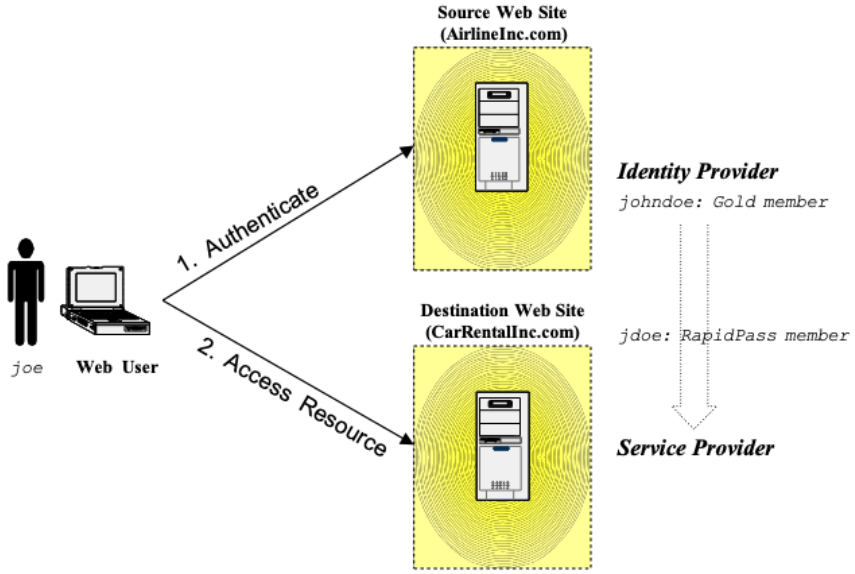


Abbildung 5.1: SSO-System Illustration[2]

Wichtige Informationen für *xam-viz* stellen die AK-Nummer und die Rollen einer Identität dar. Diese Informationen sind als zusätzliche Attribute in jeder *Assertion* eingebettet (siehe Auflistung 5.1). Die Rollen werden im Format **https://xam-viz.jku.at/roles/[role]** überliefert. Der Ausdruck in den eckigen Klammern gibt Auskunft darüber, um welche Rolle

es sich nun tatsächlich handelt. Weitere Rollen werden im selben Format mit Beistrichen getrennt codiert. Als Beispiel wird das Rollen-Attribut *https://xam-viz.jku.at/roles/teacher*, *https://xam-viz.jku.at/roles/stuko* zu den folgenden Rollen zugeordnet: *teacher* und *stuko*.

Nach einer Authentifizierung mit Shibboleth werden im Hintergrund bestimmte Servervariablen im Webserver gesetzt[3]. Die obengenannten Attribute aus den *Assertions* werden ebenfalls zu Servervariablen zugewiesen. Damit es einfacher wird diese Servervariablen auszulesen, werden die notwendigen Variablen mittels Common Gateway Interface (CGI) Skripts in die HTTP-Header des Webservers injiziert. Die Applikation muss demnach keine *Assertions* interpretieren, sondern nur die HTTP-Header lesen können. Eine Aufschlüsselung der HTML-Header zu den Attributen von *Assertions* sind in der Tabelle 5.1 vorgegeben. Dadurch, dass die HTTP-Header die Servervariablen wiedergeben und diese Variablen im Sicherheitskontext des Webservers betrieben werden, können diese Header nicht von außen manipuliert werden[3].

HTTP-Header	Attribut	Bedeutung
X-Forwarded-User	cn	AK-Nummer
X-Forwarded-Entitlements	entitlements	Rollen

Tabelle 5.1: Zuordnung HTTP-Header zu Assertions

Die Rollen von Shibboleth werden benötigt, um eine Benutzergruppe für bestimmte Aktionen zu autorisieren. So haben, z.B. die Benutzer/-innen mit der Rolle *teacher* keinen Zugriff auf die Sichten von den anderen Benutzergruppen. Nichtsdestotrotz, werden nicht alle Rollen über Shibboleth geliefert. Für die Benutzergruppen *vr, stuko* und *international-office* erfolgt die Autorisierung über die Zugriffsliste (siehe Kapitel 5.2), da die Wartung dieser Liste einfacher und schneller erfolgen kann.

```
<?xml version="1.0" ?>
<saml2:Assertion ID="..." IssueInstant="..." Version="2.0">
  <saml2:Issuer>https://shibboleth.im.jku.at/idp/shibboleth
  </saml2:Issuer>
  <saml2:Subject>...</saml2:Subject>
  <saml2:Conditions>...</saml2:Conditions>
  <saml2:AuthnStatement>...</saml2:AuthnStatement>
  <saml2:AttributeStatement>
    ...
    <saml2:Attribute FriendlyName="cn">
      <saml2:AttributeValue>AK194487</saml2:AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute FriendlyName="eduPersonEntitlement">
      <saml2:AttributeValue>https://xam-viz.jku.at/roles/teacher
      </saml2:AttributeValue>
    </saml2:Attribute>
    ...
  </saml2:AttributeStatement>
</saml2:Assertion>
```

Auflistung 5.1: Beispiel SAML-Assertions

5.2 Zugriffsliste

Die Zugriffsliste verwaltet alle Zugänge für die aktiven Stuko-Vorsitzenden und zusätzlich den Zugang für die/den VR und das Auslandsbüro. Hierbei handelt es sich um eine einfache JSON-Datei, die eine Liste von Entitäten mit verschiedensten Informationen beinhaltet. Die Zugriffsliste befindet sich lokal im Wurzelverzeichnis der Applikation und muss gegebenenfalls aktualisiert werden, wenn sich, z.B. die Stuko-Vorsitzenden ändern. Die Aktualisierung erfolgt durch die Administratoren der Applikation und erfordert aktuell das Starten eines Deployment-Jobs über Jenkins.

Die Struktur der Zugriffsliste wird in der Auflistung 5.2 näher erläutert.

```
[
  ...
    // Stuko-Vorsitz
    {
      "stuko": "Informatik",
      "role": "stuko",
      "firstname": "Max",
      "lastname": "Mustermann",
      "ak": "AKxxxxx",
      "skz": ["033/521", "066/921"]
    },
    // VR
    {
      "stuko": "",
      "role": "vr",
      "firstname": "Stefan",
      "lastname": "Koch",
      "ak": "AKxxxxxx",
      "skz": []
    },
    // Auslandsbüro
    {
      "stuko": "",
      "role": "international-office",
      "firstname": "Christine",
      "lastname": "Hinterleitner",
      "ak": "AKxxxxxx",
      "skz": []
    }
  ...
]
```

Auflistung 5.2: Zugriffsliste Struktur

Jedes Mal, wenn die Benutzer/-innen die Rolle *vr, stuko* oder *international-office* einnehmen möchten, wird diese Zugriffsliste gelesen und mithilfe der AK-Nummer aus Shibboleth die Entität eindeutig erforscht. Danach wird die angegebene Rolle mit der tatsächlich hinterlegten Rolle verglichen und je nach Ergebnis werden die Benutzer/-innen entweder zu der entsprechenden Sicht oder zu einer Fehlerseite weitergeleitet. Zusätzlich sind bei Stuko-Vorsitzenden eine Liste an Studien hinterlegt, für die sie zuständig sind. Diese werden bei der Rollenauswahl in die Session mitgespeichert, um die entsprechenden Auswahlmöglichkeiten vorzubefüllen.

6 Backend

Als Backend[4] wird ein System bezeichnet, welches sich mit der Datenverarbeitung und der Business-Logik beschäftigt und die Aufgabe hat, die verarbeiteten Daten mit einer API zur Verfügung zu stellen. Heutzutage existieren viele Frameworks und Programmiersprachen, um ein Backend-System effizient zu entwickeln. In *xam-viz* kommt Node.js zum Einsatz.

6.1 Node.js

Als Backend-Framework wird Node.js[5], oder auch kurz Node genannt, verwendet. Während die gängigsten Browser eine Laufzeitumgebung für JavaScript (JS) inkludiert haben, braucht es im Backend das Node-Framework für das Ausführen von JS Anwendungen. Node ist eine Open-Source Laufzeitumgebung für JS im Backend-Bereich und benutzt die V8-Engine von Google. Die Besonderheit von Node ist das asynchrone I/O Event-Modell, welches hohe Performance und niedrige Speicherauslastung erlaubt. Während klassiche Backend-Systeme in anderen Programmiersprachen, wie z.B. Java, Multithreading verwenden, um simultane Aufgaben zu erledigen, werden in Node diese Aufgaben als asynchrone Events in einem Single-Thread in der Applikationsebene behandelt. Aus diesem Grund wird auch JS in Node verwendet, weil diese Sprache es erlaubt asynchrone Events relativ leicht mittels *Callbacks* oder *Promises* abzubilden.

Das Node-Framework baut intern auf Modulen auf, die unterschiedliche Funktionalitäten bereitstellen. Aus dem HTTP-Modul kann aus einer Node-Instanz ein einfacher Webserver ohne viel Boilerplate-Code implementiert werden. Ein Beispiel für ein minimalistisches Node-Programm könnte wie folgt aussehen:

```
1 /* import HTTP-Modul */
2 const http = require("http");
3
4 const requestListener = function (req, res) {};
5
6 const server = http.createServer(requestListener);
7 server.listen(8000, 'localhost', () => {
8     console.log('Server is running on http://localhost:8000');
9 });
```

Auflistung 6.1: Node-Beispiel

In der Zeile 6 sieht man die Übergabe der Callback-Funktion *requestListener*, die in der Zeile 4 definiert ist. Im *requestListener* kann mithilfe der zwei Objekte *req* (eingehende Anfrage) und *res* (ausgehende Antwort) eine Web-Applikation entwickelt werden.

Natürlich müssen in einer Web-Applikation weitere Themen wie Schnittstellen, Session-Management, Security, etc. berücksichtigt werden. Über die Zeit wurden zahlreiche Frameworks aufbauend auf Node entwickelt, die das Entwickeln von Web-Applikationen noch einfacher gestalten sollen. Ein bekanntes Framework stellt *Express.js* dar.

6.2 Express.js

Express.js[6], oder auch kurz Express genannt, ist ein Web-Framework, welches auf Node aufbaut und zahlreiche Features anbietet, die das Entwickeln von Web-Applikationen vereinfachen. Dazu gehören u.a. Session-Management, Schnittstellen, Caching, APIs, Middleware und Security. Express stellt ein Modul in Node dar und arbeitet im Kern mit sogenannten Middleware-Funktionen. Bei einer Middleware handelt es sich um eine Funktion, die Zugriff auf das Anforderungsobjekt (req), das Antwortobjekt (res) und die nächste Middleware (next) im Anforderung/Antwort-Zyklus der Anwendung hat. Im Grunde besteht eine Express-Applikation aus einem Middleware-Stack von Funktionen, die nacheinander ausgeführt werden und bestimmte Funktionalitäten erfüllen. Eine Middleware hat im Wesentlichen folgende Aufgaben:

- Ausführen von beliebigem Code

- Vornehmen von Änderungen an den Anforderungs- und Antwortobjekten
- Beenden des Anforderung/Antwort-Zyklus
- Aufrufen der nächsten Middleware im Stack

Der Einsatz von Middleware-Funktionen erlaubt es, zahlreiche Funktionalitäten, wie etwa die Authentifizierung, das Logging, die Fehlerbehandlung, etc. zentral zu entwickeln und damit automatisch eines der wichtigsten Konzepte der Softwareentwicklung zu verfolgen: *DRY - Don't Repeat Yourself*.

Um das Konzept von Express näherzubringen, soll ein minimalistisches Beispiel dazu dienen:

```
1  /* Import Express-Modul */
2  const express = require('express')
3  const app = express()
4
5  /* Middleware 1 */
6  app.use((req, res, next) => {
7    console.log(req.url);
8    next();
9  })
10
11 /* Middleware 2 */
12 app.get('/', (req, res) => {
13   res.send('Hello World!')
14 })
15
16 app.listen(3000, () => {
17   console.log('Example app listening on port 3000')
18 })
```

Auflistung 6.2: Express-Beispiel

In Zeilen 6-9 bzw. 12-14 werden zwei verschiedene Arten von Middleware-Funktionen demonstriert, nämlich applikationsbezogene und schnittstellenbezogene Middleware. Eine applikationsbezogene Middleware wird immer ausgeführt, wenn eine Anfrage den Webserver erreicht. In dem Beispiel 6.2 wird die URL bei jeder Anfrage auf die Konsole ausgegeben und die nächste Middleware im Stack aufgerufen. Im Gegensatz dazu wird eine schnittstellenbezogene Middleware erst dann ausgeführt, wenn die Anfrage den

Webserver auf die definierte Schnittstelle mit der definierten HTTP-Methode erreicht. In Zeilen 12-14 ist die Middleware auf die Wurzel-URL (’/’) mit der GET HTTP-Methode definiert.

Die Reihenfolge der Middleware-Funktionen im Stack entspricht der Reihenfolge ihrer Definition. Im Beispiel 6.2 würde der Aufruf von `http://localhost:3000/` zuerst die Middleware 1 und danach die Middleware 2 ausführen, wobei eine Anfrage an eine andere Schnittstelle nur die Middleware 1 triggern würde.

6.3 Oracle-DB

Die Daten über Lehrveranstaltungen und Prüfungen werden in einer Oracle-DB zur Verfügung gestellt. Wie im Kapitel 4 bereits erläutert, kommen Tabellenfunktionen[7] zum Einsatz. Eine Tabellenfunktion ist eine herkömmliche Funktion in Oracle SQL, die innerhalb einer FROM-Anweisung eines SELECT-Statements aufgerufen werden kann. Die Funktionen geben eine Sammlung an Daten zurück, die mit dem TABLE-Schlüsselwort zu einem Datensatz mit Zeilen und Spalten transformiert werden können. Das Verwenden von Tabellenfunktionen bietet sich an, wenn:

- mehrere Daten an einem Ort zusammengefügt werden sollen oder
- eine View mit Parametern benötigt wird

Eine SQL Abfrage, welche alle Abhaltungen eines bestimmten Lehrenden im Sommersemester 2019 zurückliefert, kann wie folgt aussehen:

```
SELECT * FROM TABLE (EXAMDV.LVANotenMWL('AKxxxxxx', '2019S'));
```

Der Zugriff auf die Oracle-DB erfolgt über ein Express-Modul (*node-oracledb[8]*). Dadurch können Connection-Pooling, Prepared-Statements und zahlreiche andere Features verwendet werden. Allerdings können die Input-Parameter von Tabellenfunktionen nicht als Prepared-Statements angegeben werden. Das bedeutet, dass diese Parameter im Vorhinein durch die Applikation selber validiert werden müssen.

7 Frontend

Als Frontend[4] werden die Teile einer Applikation bezeichnet, die näher an dem/der Endbenutzer/-in angesiedelt sind. Im Grunde ist das Frontend verantwortlich für das *Look&Feel* der Applikation, also die Darstellung der Daten aus dem Backend und die Interaktion der Benutzer/-innen mit der Applikation.

Für das Frontend wird hauptsächlich HTML, CSS und JS verwendet. Es gibt verschiedene Arten, das Frontend aufzubauen. Eine Möglichkeit ist, das HTML serverseitig zu erzeugen und dem Browser als Antwort einer Anfrage zu übermitteln. In der Regel kommen hier Template-Engines zum Einsatz, die als Teil eines Web-Frameworks existieren, wie Thymeleaf für SpringBoot, Blade für Laravel und PUG für Express. Die zweite Möglichkeit ist, JS-Frontend-Frameworks wie React, Angular oder Vue zu verwenden, die die Daten aus den Schnittstellen des Backends laden und das HTML selbst erzeugen.

Die erste Methode hat den Vorteil, dass die ganze Logik im Backend angesiedelt ist, also ein Ort, wo die ganze Applikation entwickelt wird. Zusätzlich wird mit Templates das Erstellen und Generieren von HTML-Seiten erleichtert und durch die Verwendung von Bedingungen, Iterationen, usw. besteht die Möglichkeit, Logik in die HTML-Erstellung zu integrieren. Der wesentliche Nachteil ist, dass HTML-Strukturen zu komplex und Templates dadurch unübersichtlich werden können. Zusätzlich kann dazu tendiert werden, mehr Logik als nötig in Templates zu integrieren, was die Komplexität und Performance noch weiter beeinträchtigt. Außerdem muss beachtet werden, dass je größer das HTML wird, umso länger die Übermittlung der Antwort vom Backend zum Frontend dauert. Dies kann allerdings mit Caching effizient umgegangen werden.[9]

Frontend-Frameworks beheben die zuvor genannten Nachteile, da hier in der Regel JSON-Strukturen in HTML umgewandelt werden. Das heißt, die Performance zwischen Backend und Frontend ist besser und da solche Frameworks näher am Browser angesiedelt sind, ist die Darstellung des HTMLs auch um einiges schneller. Die HTML-Strukturen werden

in der Regel auch in Komponenten geteilt, um die Komplexität zu vermindern. Allerdings sind solche Frameworks nicht trivial zu erlernen und es braucht viel Zeit, um so ein Framework voll und ganz zu beherrschen.[9]

Aufgrund der Tatsache, dass *xam-viz* keine komplexen Bedienungen enthält und die Interaktion auf zwei Seiten begrenzt ist, wurde auf ein Frontend-Framework verzichtet. Zusätzlich werden die Templates auch in Komponenten geteilt, damit ein Template nicht zu komplex und unübersichtlich wird. Für Express gibt es die Template-Engine PUG.

7.1 PUG

PUG[10], ehemals Jade, ist eine Template-Engine für JS, welche PUG-Syntax in HTML-Code umwandeln kann. Das Schreiben von HTML bringt viel Redundanz mit sich. Durch den Einsatz von Templates wird einerseits die Redundanz eliminiert und viel Boilerplate-Code gespart und andererseits besteht die Möglichkeit, Bedingungen, String-Interpolation, Iterationen, Vererbung, usw. zu verwenden. Eine positive Eigenschaft von PUG ist nicht nur die Verwendung von JS, sondern auch die nahtlose Integration in Express. So kann durch ein Modul in Express, als Antwort auf eine Anfrage ein PUG-kompiliertes HTML als Antwort zurückgegeben werden. Zusätzlich besteht auch die Möglichkeit, Parameter in ein PUG-Template zu füttern. Durch diese Kombination können ziemlich leicht dynamische HTML-Seiten generiert werden. Das Beispiel für ein PUG-Template (siehe Abbildung 7.1) und der resultierende HTML-Code (siehe Abbilung 7.2) sollen ein paar der Eigenschaften von PUG aufzeigen. Wie man erkennen kann, ist das PUG-Template wesentlich kürzer und kompakter als das generierte HTML.

```
doctype html
html(lang='en')
 head
   title Hello, World!
 body
   h1 Hello, World!
   div.remark
     p Pug rocks!
```

Abbildung 7.1: PUG-Template

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Hello, World!</title>
  </head>
  <body>
    <h1>Hello, World!</h1>
    <div class="remark">
      <p>Pug rocks!!</p>
    </div>
  </body>
</html>
```

Abbildung 7.2: PUG-kompiliertes HTML

7.2 JQuery

Damit die Benutzer/-innen mit der Applikation interagieren können, wird JS im Browser verwendet. Dadurch können z.B. dynamische Inhalte mit AJAX nachgeladen werden. Für diesen Zweck kommt das JQuery[11] Framework zum Einsatz. Kurz zusammengefasst, ist JQuery eine JS-Bibliothek, die zahlreiche Features, wie DOM Manipulation, Event-Handling, AJAX-Requests, etc. bereitstellt. Zusätzlich handelt es sich bei JQuery um ein weit verbreitetes und gut dokumentiertes Framework.

7.3 Bootstrap

Die Optik einer Webanwendung wird mittels CSS gestaltet. Allerdings spielt CSS auch bei der Platzierung der einzelnen HTML-Elemente eine wesentliche Rolle. Bei Bootstrap[12] handelt es sich um ein CSS-Framework, dessen Kernaufgabe es ist, das Entwickeln von responsiven Webanwendungen zu erleichtern. Zusätzlich bietet Bootstrap eine Vielzahl an Design-Templates für Buttons, Typography, Formulare, Navigation, etc. an. In Bootstrap kommt ein sogenanntes *Grid-System* zum Einsatz, wodurch die Elemente einer Webanwendungsseite in einer tabellarischen Struktur dargestellt werden können.

8 Benutzung

In diesem Kapitel wird die Handhabung von *xam-viz* beschrieben. Um *xam-viz* verwenden zu können, muss man sich im JKU-Netz befinden bzw. eine aktive VPN-Verbindung ins JKU-Netz haben. Der Aufruf erfolgt mittels `https://xam-viz.jku.at`.

8.1 Login

Beim Aufruf erscheint eine Login-Maske (Abbildung 8.3), auf der man sich mit seiner AK-Nummer und seinem AK-Passwort anmelden muss. Falls man bereits auf einem anderen JKU-Dienst eingeloggt ist, wird die Login-Maske nicht mehr angezeigt, sondern man wird sofort weitergeleitet.

Nach der Anmeldung erscheint eine Maske, auf der man die gewünschte Rolle auswählen muss (siehe Punkt 1 in Abbildung 8.1). Folgende Rollen stehen zur Auswahl:

- Lehrende/r
- Stuko-Vorsitz
- VR-Lehre
- Auslandsbüro

Die Auswahl der Rolle entscheidet, auf welche Daten die Benutzer/-innen zugreifen dürfen und welche Ansicht ihnen präsentiert wird. Nach Auswahl der Rolle klickt man auf den Login-Button (Punkt 2 in Abbildung 8.1). Dadurch wird man auf die jeweilige Sicht der ausgewählten Rolle weitergeleitet. Wählt man eine Rolle aus, für die man keine Berechtigung hat, wird eine entsprechende Fehlermeldung angezeigt (siehe Abbildung 8.2).

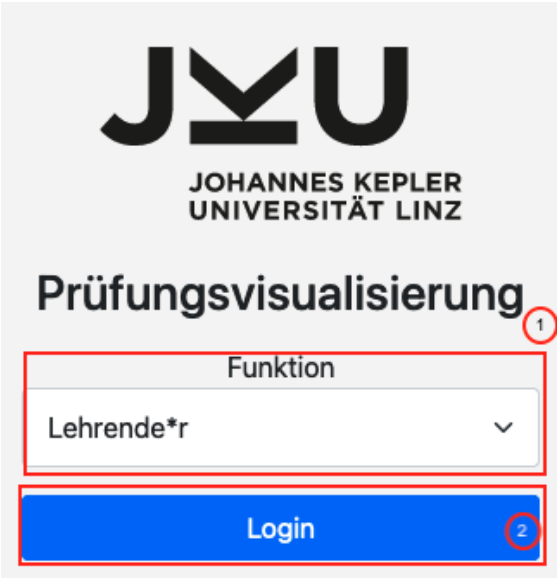


Abbildung 8.1: Rollenauswahl

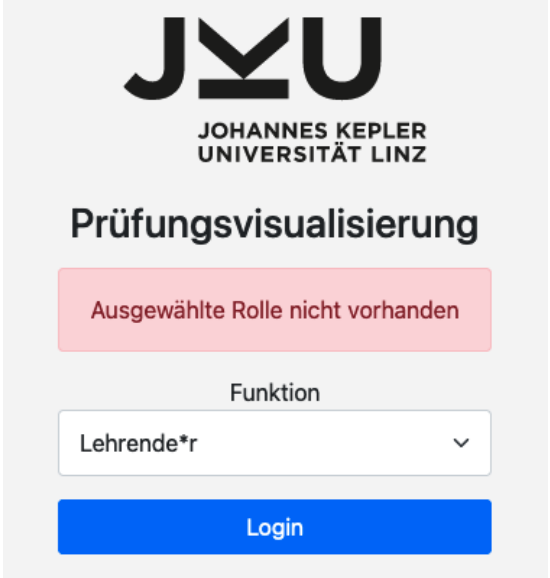


Abbildung 8.2: Rolle nicht vorhanden

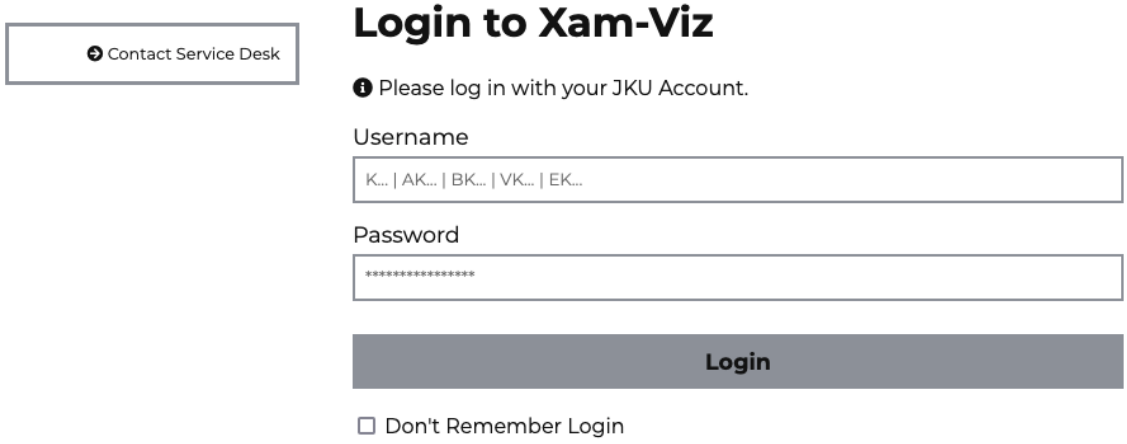


Abbildung 8.3: Shibboleth-Login-Maske

8.2 Ansicht für Lehrende

Lehrende können sich die Prüfungsergebnisse ihrer eigenen LVAs pro Semester anzeigen lassen, sie auf Wunsch kommentieren und gegebenenfalls der Weiterleitung an den Stuko-Vorsitz widersprechen. Der Aufbau der Benutzeroberfläche ist wie folgt (siehe nummerierte Punkte in Abbildung 8.4):

1. *Semesterauswahl*
   Hier werden die letzten sechs Semester zur Auswahl angeboten, wobei das aktuelle Semester vorselektiert wird. Über welchen Zeitraum das aktuelle Semester geht, wird in Abschnitt 8.7 erläutert.

2. *Rolle wechseln*
   Die ausgewählte Rolle kann jederzeit gewechselt werden (sofern man die Berechtigung dafür hat). Dazu wird man wieder zur Rollenauswahl-Maske weitergeleitet.

3. *LVA-Titel*
   Name der LVA. Falls ein Untertitel vorhanden ist, wird dieser ebenfalls angezeigt.

4. *Durchschnittsnote*
   Die Durchschnittsnote wird pro LVA sowie aggregiert für alle LVAs angezeigt.

5. *Durchfallrate in %*
   Die Durchfallrate wird pro LVA sowie aggregiert für alle LVAs angezeigt.

6. *Notenverteilung*
   Siehe Abschnitt 8.8.

7. *Anzahl nicht beurteilt*
   Differenz zwischen der Anzahl der Anmeldungen der LVA und der Anzahl der beurteilten Studierenden.

8. *Widersprochene LVA und Kommentar*
   Falls Lehrende der Weiterleitung von Prüfungsergebnissen an den Stuko-Vorsitz widersprochen haben (siehe Detailansicht), wird das hier farblich hervorgehoben. Falls ein Kommentar angegeben wurde, wird dieser als Pop-up angezeigt, sobald man mit der Maus darüberfährt.

Bei all diesen Daten ist zu beachten, dass pro Prüfling nur die letzte Prüfung berücksichtigt wird, d.h. bei Prüfungswiederholungen fließt nur die letzte Prüfung in die Berechnungen ein.

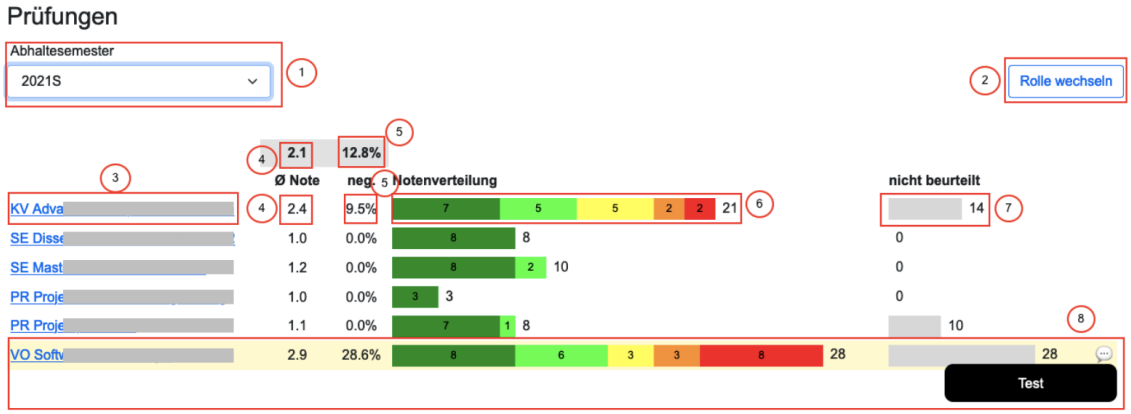


Abbildung 8.4: Übersicht für Lehrende

Detailansicht einer LVA für Lehrende

Durch einen Klick auf den LVA-Titel gelangt man zur Detailseite dieser LVA (Abbildung 8.5). Hier besteht die Möglichkeit, Prüfungsergebnisse nach bestimmten Kriterien zu gliedern und dadurch detailliertere Analysen durchzuführen. Zusätzlich können Lehrende innerhalb einer bestimmten Frist (siehe Abschnitt 8.7) die Prüfungsergebnisse kommentieren und/oder der Weiterleitung an den Stuko-Vorsitz widersprechen. Der Aufbau der Detailseite ist wie folgt:

1. *LVA-Titel und Semester*
2. *Gliederungsmöglichkeiten* mit bis zu zwei Ebenen
   Gegliedert werden kann nach dem Prüfungsdatum, der Studienkennzahl und dem Jahrgang. In Abbildung 8.5 wurde z.B. zuerst nach dem Prüfungsdatum und innerhalb dessen nach der Studienkennzahl gegliedert.
3. *Gesamtvisualisierung* wie in der Hauptansicht
4. *Gliederungsebene 1*
   In der Abbildung 8.5 wurde z.B. nach Prüfungsdatum untergliedert.

5. *Gliederungsebene 2*
   In der Abbildung 8.5 wurde z.B. nach der Studienkennzahl untergliedert. Wenn man mit der Maus über eine Studienkennzahl fährt, erscheint der Name des Studiums als Pop-up.

6. *Widerspruchsmöglichkeit*
   Innerhalb der Frist (siehe Abschnitt 8.7) kann hier der Weiterleitung der Ergebnisse an den Stuko-Vorsitz widersprochen werden.

7. *Kommentarmöglichkeit*
   Innerhalb der Frist (siehe Abschnitt 8.7) kann hier ein Kommentar zu den Prüfungsergebnissen abgegeben werden, der für den Stuko-Vorsitz sichtbar ist, sofern der Weiterleitung nicht widersprochen wurde.

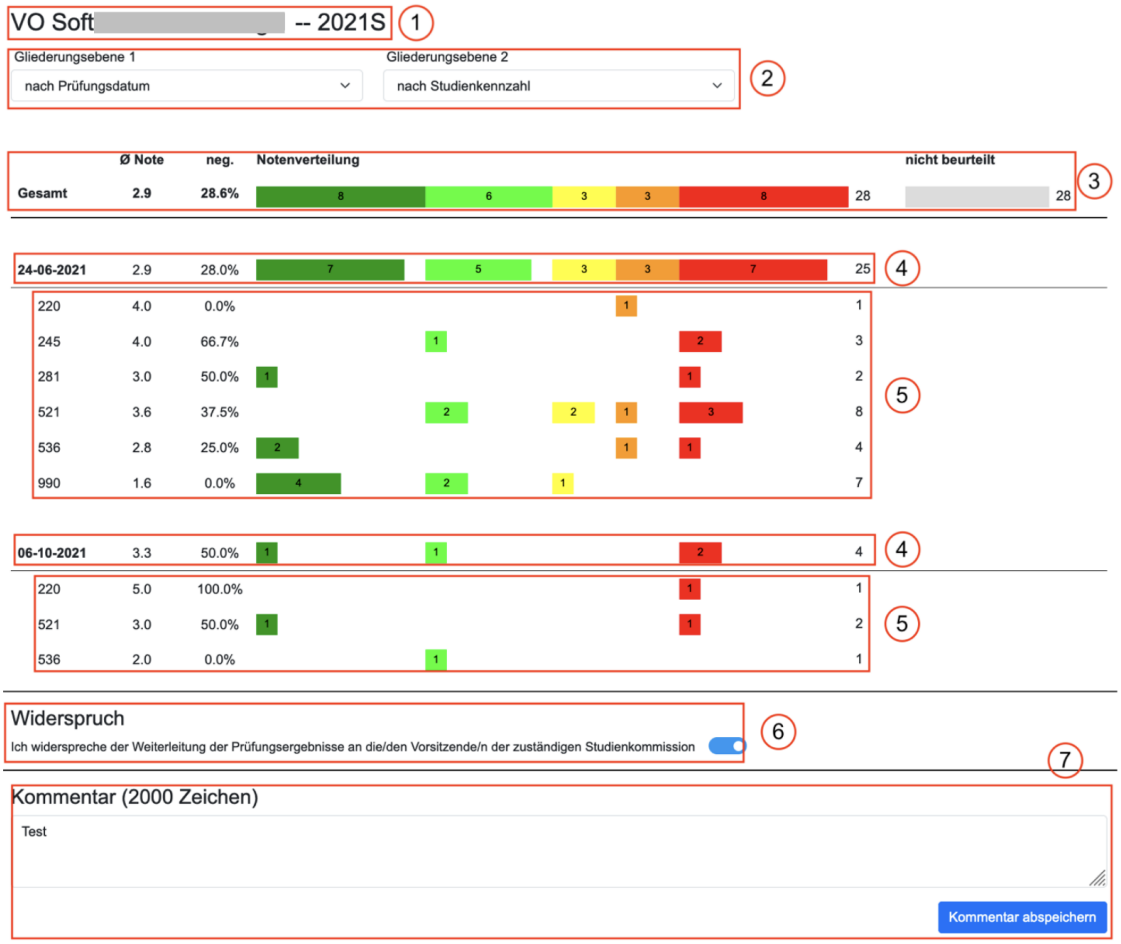


Abbildung 8.5: Detailansicht Lehrende

8.3 Ansicht für Stuko-Vorsitzende

Stuko-Vorsitzende sehen die Prüfungsergebnisse aller LVAs in den von ihnen betreuten Studien. Das gewünschte Studium sowie das gewünschte Semester können über Dropdown-Menüs ausgewählt werden. Zusätzlich gibt es Filter- und Sortiermöglichkeiten. Die Sicht für Stuko-Vorsitzende wird allerdings erst freigeschaltet, nachdem die Widerspruchsfrist (Abschnitt 8.7) abgelaufen ist. Der Aufbau der Benutzeroberfläche ist wie folgt (siehe Abbildung. 8.6):

1. *Auswahl der Studienrichtung*
   Hier kann über ein Drop-down-Menü eines der Studien ausgewählt werden, für das die Studienkommission zuständig ist.

2. *Semesterauswahl*
   Hier kann über ein Drop-down-Menü eines der letzten sechs Semester ausgewählt werden. Es werden dann alle LVAs des ausgewählten Studiums im ausgewählten Semester angezeigt.

3. *LVA-Art* (Filter)
   Die angezeigten LVAs können nach ihrer Art gefiltert werden: AG, IK/KS, VO/VL, KV/VU, UE, PR, SE, PS

4. *Sortiermöglichkeit*
   Die angezeigten LVAs können nach folgenden Kriterien sortiert werden: LVA-Name, Zuteilungen, Beurteilungen, Durchschnittsnote, Durchfallrate, Verzögerung.

5. *Ausblenden*
   Einzelne LVAs können (filterübergreifend) ausgeblendet werden, um die Ansicht auf einige wenige im Analysekontext relevante LVAs zu beschränken.

6. *Alle einblenden*
   Ausgeblendete LVAs können damit wieder eingeblendet werden.

7. *Rolle wechseln*
   Die ausgewählte Rolle kann jederzeit gewechselt werden, sofern man die Berechtigung dafür hat. Dazu wird man wieder zur Rollenauswahl-Maske weitergeleitet.

8. *LVA-Titel*
   Name der LVA. Falls ein Untertitel vorhanden ist, wird dieser ebenfalls angezeigt.

9. *Durchschnittsnote*
   Die Durchschnittsnote wird pro LVA sowie aggregiert für alle LVAs angezeigt.

10. *Durchfallrate in %*
    Die Durchfallrate wird pro LVA sowie aggregiert für alle LVAs angezeigt.

11. *Durchschnittliche Verzögerung in Semestern*
    Siehe Abschnitt 8.6.

12. *Notenverteilung*
    Siehe Abschnitt 8.8.

13. *Anzahl nicht beurteilt*
    Differenz zwischen der Anzahl der Anmeldungen der LVA und der Anzahl der beurteilten Studierenden.

Bei all diesen Daten ist zu beachten, dass bei Prüfungswiederholungen nur die letzte Prüfung berücksichtigt wird. Außerdem sehen Stuko-Vorsitzende nur jene LVAs, deren Weiterleitung an den Stuko-Vorsitz von Lehrenden nicht widersprochen wurde. Wurde der Weiterleitung nicht widersprochen, sondern lediglich ein Kommentar angebracht, wird dieser als Pop-up angezeigt, sobald man mit der Maus darüberfährt.

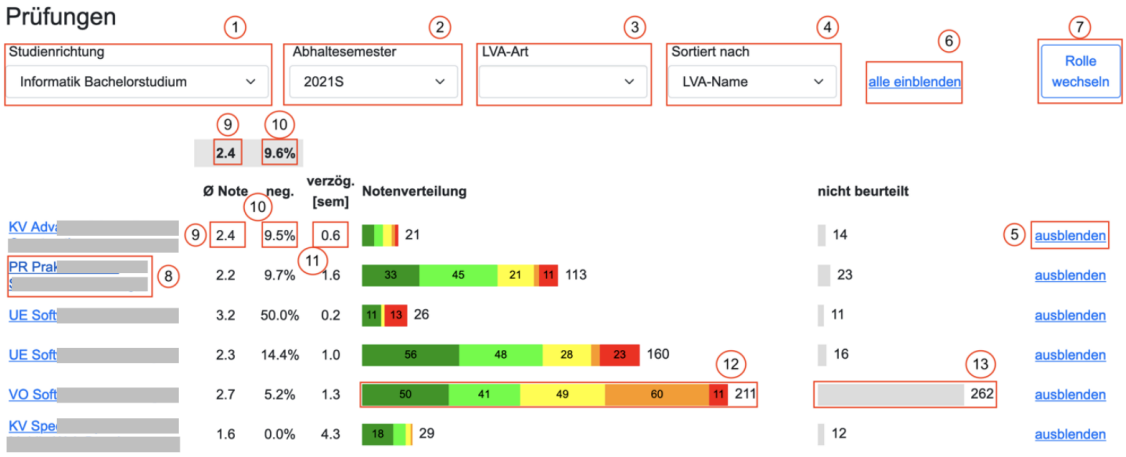


Abbildung 8.6: Übersicht für Stuko-Vorsitzende

Detailansicht einer LVA für Stuko-Vorsitzende

Durch einen Klick auf den LVA-Titel gelangt man zur Detailseite dieser LVA (Abbildung 8.7). Hier besteht die Möglichkeit Prüfungsergebnisse nach bestimmten Kriterien zu gliedern und dadurch detailliertere Analysen durchzuführen. Der Aufbau der Detailseite ist wie folgt:

1. *LVA-Titel und Semester*

2. *Gliederungsmöglichkeiten* mit bis zu zwei Ebenen
   Gegliedert werden kann nach dem Prüfungsdatum, der Studienkennzahl und dem Jahrgang.

3. *Gesamtvisualisierung* wie in der Hauptansicht
4. *Gliederungsebene 1*
   In Abbildung 8.7 wurde z.B. nach dem Prüfungsdatum untergliedert.
5. *Gliederungsebene 2*
   In Abbildung 8.7 wurde z.B. nach dem Jahrgang untergliedert.

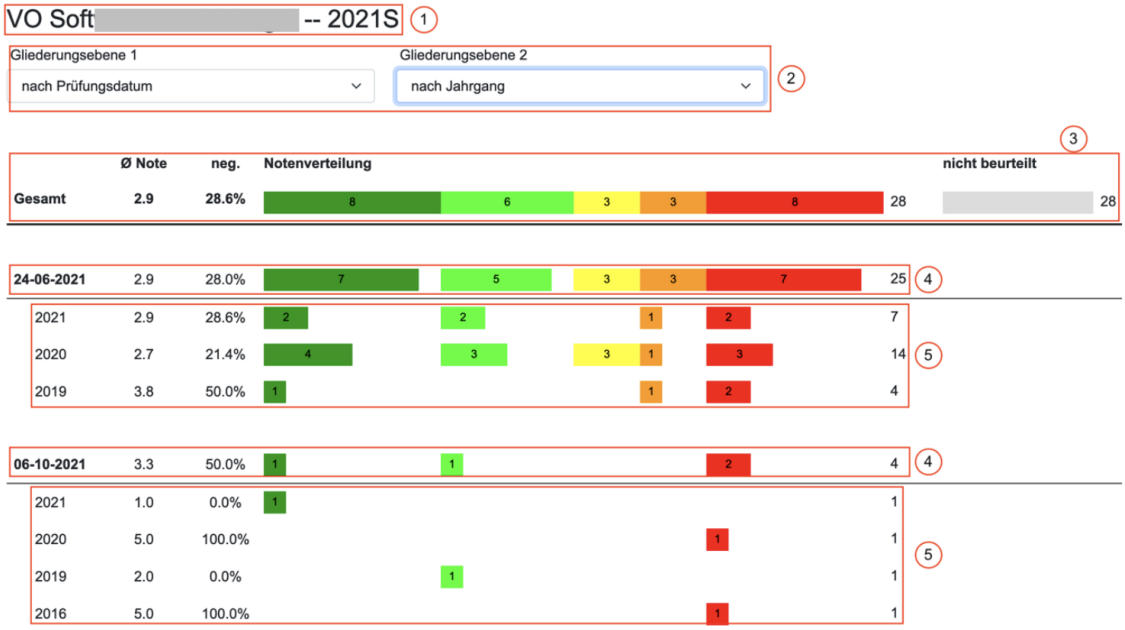


Abbildung 8.7: Detailansicht Stuko-Vorsitz

8.4 Ansicht für VR-Lehre

Die/der VR-Lehre sieht die Prüfungsergebnisse aller LVAs in allen Studien, einschließlich der Ergebnisse von LVAs, deren Weiterleitung an den Stuko-Vorsitz widersprochen wurde. Die Benutzeroberfläche ist identisch zu jener für Stuko-Vorsitzende (Abbildung 8.6), mit dem Unterschied, dass alle Studien der JKU ausgewählt werden können und auch Prüfungsergebnisse angezeigt werden, deren Weiterleitung an den Stuko-Vorsitz widersprochen wurde. Solche LVAs werden farblich hinterlegt (Abbildung 8.8).

Wurde von den Lehrenden ein Kommentar angebracht, so erscheint dieser, wenn man mit der Maus über das entsprechende Symbol fährt.

Detailansicht einer LVA

Durch einen Klick auf den LVA-Titel gelangt man zur Detailansicht dieser LVA. Ihre Darstellung und Benutzung sind analog zu der Sicht für Stuko-Vorsitzende (siehe Abbildung 8.7).

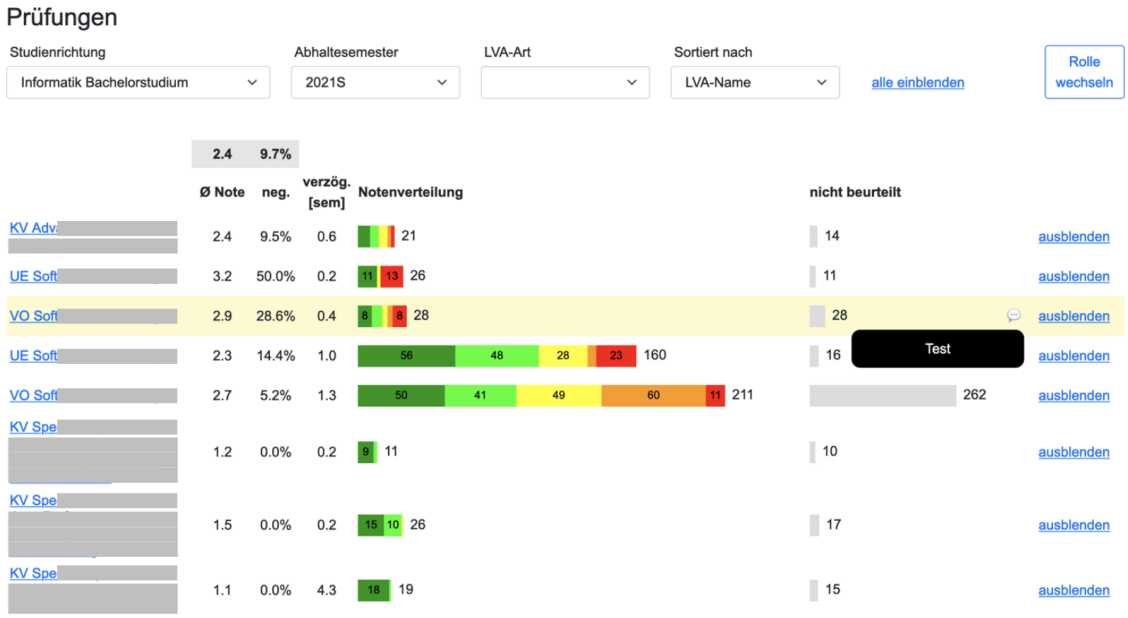


Abbildung 8.8: Anzeige Widerspruch/Kommentar

8.5 Ansicht für das Auslandsbüro

Für das Auslandsbüro bietet *xam-viz* eine Ansicht, bei der die Prüfungsergebnisse aller LVAs eines Studiums aggregiert werden, wobei die Ergebnisse einzelner LVAs nicht angezeigt werden. Dabei kann zwischen den Prüfungsergebnissen von Studierenden der JKU und den Prüfungsergebnissen von Austauschstudierenden gewählt werden. Diese Ansicht wird vom Auslandsbüro für das Reporting im Rahmen von Austauschprogrammen benötigt. Der Aufbau der Benutzeroberfläche ist wie folgt (siehe Abbildung 8.9):

1. *Auswahl der Semesterintervalls*
   Die maximale Semesterspanne beträgt drei Jahre.

2. *Studierenden-Kategorie*
   Hier kann zwischen "Studierende der JKU" und "Austauschstudierende" gewählt werden. Studierende in Joint-Degree-Programmen gelten als Studierende der JKU.

3. *Sortiermöglichkeit*
   Die Ansicht kann nach Studium, Durchschnittsnote, Durchfallrate und Anzahl der Prüfungen sortiert werden.

4. *CSV-Export*
   Die Daten der gewählten Studierenden-Kategorie können als CSV-Datei exportiert werden (siehe unten).

5. *Rolle wechseln*
   Die ausgewählte Rolle kann jederzeit gewechselt werden (sofern man die Berechtigung dafür hat). Dazu wird man wieder zur Rollenauswahl-Maske weitergeleitet.

6. *Studienrichtung*

7. *Durchschnittsnote*

8. *Notenverteilung in %*
   Siehe Abschnitt 8.8.

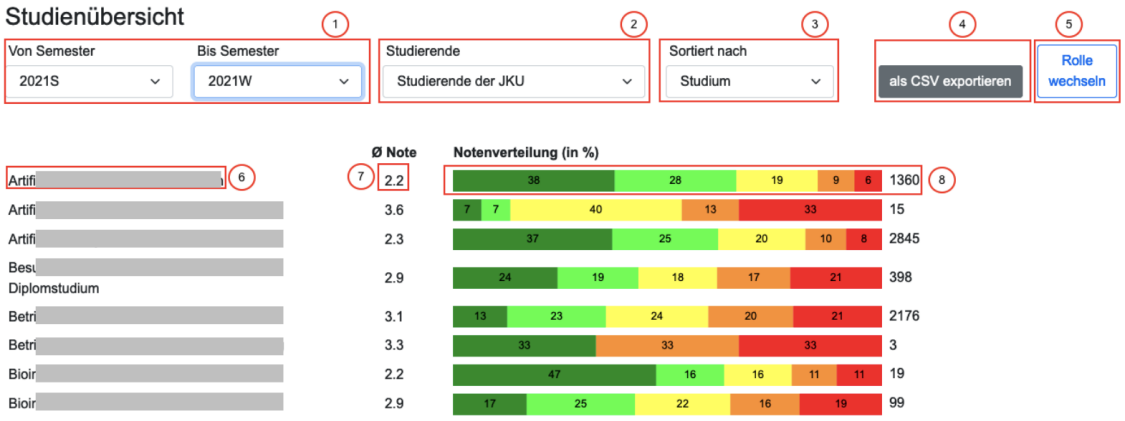


Abbildung 8.9: Benutzeroberfläche für das Auslandsbüro

CSV-Export

Durch einen Klick auf *CSV-Export* generiert *xam-viz* eine CSV-Datei mit allen Daten des eingestellten Semesterintervalls und der gewählten Studierenden-Kategorie. Mittels Excel können darauf weitere Auswertungen durchgeführt werden. Beim Importieren der CSV-Datei in Excel muss als Trennzeichen ein Beistrich und als Zeichencodierung UTF-8 eingestellt werden.

8.6 Verzögerung

Für den Stuko-Vorsitzenden und die/der VR-Lehre wird die durchschnittliche Verzögerung in Semestern angezeigt. Die unter Verzögerung angegebene Zahl ist eine Schätzung, mit wie vielen Semestern Verzögerung eine Prüfung im Schnitt absolviert wurde. Sie wird gebildet, indem man den in einer Prüfung am häufigsten auftretenden Jahrgang (d.h. Semester ab Studienbeginn) ermittelt und die mittleren Abweichungen des Jahrgangs jedes einzelnen Prüflings von diesem Durchschnittsjahrgang berechnet. Bei Pflichtlehrveranstaltungen ist der am häufigsten auftretende Jahrgang in der Regel jener, für den die Prüfung laut Curriculum vorgesehen ist. Jedoch ist bei Wahllehrveranstaltungen der am häufigsten auftretende Jahrgang einem gewissen Zufall unterworfen. Daher ist die Zahl lediglich eine Schätzung.

8.7 Fristen

Lehrende können innerhalb der unten angegebenen Fristen die Prüfungsergebnisse ihrer LVAs für das aktuelle Semester kommentieren und/oder der Weiterleitung dieser Ergebnisse an den Stuko-Vorsitz widersprechen (siehe Abschnitt 8.2, Detailansicht). Die Sicht für Stuko-Vorsitzende (Abschnitt 8.3) wird erst nach Ablauf dieser Fristen für das betreffende

Semester	Fristen
Wintersemester	01. Dezember - 30. April
Sommersemester	01. Mai - 30. November

Tabelle 8.1: Fristen

Semester freigeschaltet. Das aktuelle Semester ist zwischen dem 1. Mai und 30. November das Sommersemester und zwischen dem 1. Dezember und 30. April des Folgejahres das Wintersemester. Beim Drop-down-Menü zur Auswahl eines Semesters durch Lehrende, Stuko-Vorsitzende und die/den VR-Lehre wird standardmäßig das aktuelle Semester angezeigt.

8.8 Notenverteilung

Die Notenverteilung von Prüfungsergebnissen wird durch farbige Balken dargestellt, wobei die Farben folgende Bedeutung haben:

Note	Farbe
Sehr Gut	
Gut	
Befriedigend	
Genügend	
Nicht Genügend	
P	
N	

Tabelle 8.2: Beurteilungen

Die Noten *P* und *N* fassen alle nicht numerischen Prüfungsbeurteilungen zusammen, wobei *P* die positiven Beurteilungen (*bestanden, mit Auszeichnung bestanden, positiv, mit Erfolg teilgenommen, abgeschlossen, mit ausgezeichnetem Erfolg*) zusammenfasst und *N* die negativen Beurteilungen (*nicht bestanden, ohne Erfolg teilgenommen*).

Die Zahl in einem Balken stellt die absolute Anzahl der Beurteilungen dar (in der Sicht des Auslandsbüros stellen die Zahlen relative Häufigkeiten dar). Die Beurteilungen *P* und *N* werden für das Auslandsbüro ignoriert. Wenn ein Balken keine Zahl enthält, wird diese als Pop-up angezeigt, wenn man die Maus über den Balken bewegt.

9 Implementierung

In diesem Kapitel werden essenzielle Implementierungsdetails vereinfacht dargestellt. Da diese Masterarbeit nicht JS an sich erklärt, wird in den nachfolgenden Abschnitten ein gewisses Know-how vorausgesetzt. Der Source-Code befindet sich in einem Bitbucket-Repository, welches vom IM verwaltet wird. Die URL dazu lautet: `https://stash.im.jku.at/projects/XAM/repos/xam-viz/browse`. Um auf das Repository zuzugreifen, muss der Zugriff zuerst vom IM freigeschaltet werden.

9.1 Struktur

Die Applikation ist, wie unten gezeigt, gegliedert und wird in den nachfolgenden Abschnitten genauer erklärt.

```
/
├── bin/
├── config/
├── middleware/
├── controllers/
├── service/
├── repository/
├── routes/
├── views/
├── test/
├── public/
│   ├── javascripts/
│   ├── stylesheets/
│   └── images/
├── app.js
├── auth.js
├── .env
├── package.json
└── stukoACL.json
```

9.2 Datenfluss

Die Express-Applikation besteht aus einzelnen Blöcken, wobei die Daten von einem Block in den anderen durchwandern. Abbildung 9.1 zeigt den Datenfluss in grafischer Form. Der Startpunkt für die Applikation ist der **Frontend**-Block, über den die Benutzer/-innen, durch die Interaktion mit der Applikation, Anfragen an die Schnittstellen senden. Das Anwenden von Filter bzw. Sortiermöglichkeiten wird mittels JS als AJAX-Anfrage an die entsprechenden Schnittstellen übermittelt.

Bevor die Anfragen weiterverarbeitet werden, müssen diese den **Middleware**-Block durchlaufen. Es gibt verschiedene Middleware-Implementierungen, die im Abschnitt 9.5 näher erläutert werden. Allerdings existieren Anfragen, für die keine Middleware-Aktionen vorgesehen sind, wie z.B. die Rollenauswahl.

Im **Controller**-Block werden die Anfragen an die Schnittstellen verarbeitet. In diesem Schritt werden die Parameter ausgelesen, Daten vom Service-Block geladen und in PUG-Templates verpackt als Antwort zurückgesendet. Die Autorisierung der Anfragen erfolgt ebenfalls in diesem Block.

Der **Service**-Block enthält die Geschäftslogik der Applikation. Hier werden Daten aus der DB über das Repository geladen und in interne JSON-Strukturen umgewandelt. Zusätzlich findet in diesem Block jede Art von Logik statt, wie z.B. Gruppieren, Filtern, etc.

Der **Repository**-Block ist auch als Datenschicht zu interpretieren. Hier werden nämlich die Verbindungen und die einzelnen Aufrufe zu der DB verwaltet.

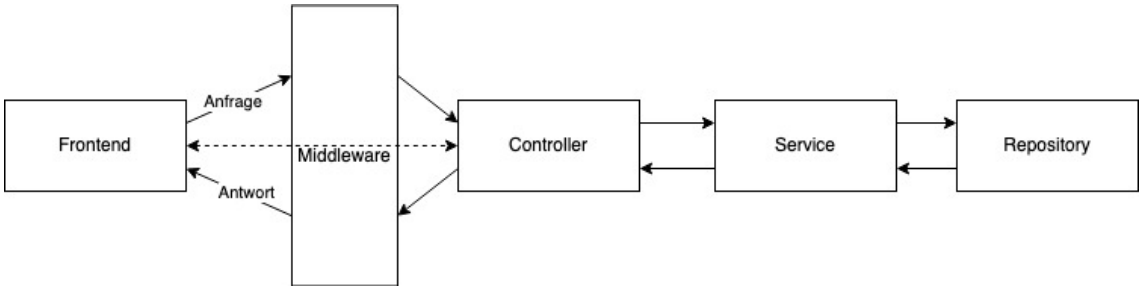


Abbildung 9.1: Datenflussdiagramm

9.3 Schnittstellen

Bevor auf die einzelnen Blöcke näher eingegangen wird, werden die existierenden Schnittstellen der Applikation näher erläutert. Die folgenden Schnittstellen sind in *routes/index.js* definiert und sind jeweils einer Controller-Aktion zugewiesen. Tabelle 9.1 stellt eine Übersicht über die vorhandenen Schnittstellen und deren Aufgaben dar. Gelb hinterlegte Schnittstellen erfordern keine aktive Authentifizierung mit Shibboleth.

HTTP Methode	Schnittstelle	Aufgabe
GET	/	Applikationseinstieg
GET	/logout	Session terminieren
GET	/callback	Shibboleth-Callback
GET	/login	Rollenauswahl
POST	/login	Rollenauswahl
GET	/vr	VR-Haupansicht
GET	/vr/ajax/get-data	AJAX VR-Hauptansicht
GET	/vr/lva/classId	VR-Detailansicht
GET	/vr/lva/ajax/get-data	AJAX VR-Detailansicht
GET	/stuko	Stuko-Hauptansicht
GET	/stuko/ajax/get-data	AJAX Stuko-Hauptansicht
GET	/stuko/lva/classId	Stuko-Detailansicht
GET	/stuko/lva/ajax/get-data	AJAX Stuko-Detailansicht
GET	/teacher	Lehrende-Hauptansicht
GET	/teacher/ajax/get-data	AJAX Lehrende-Hauptansicht
GET	/teacher/lva/classId	Lehrende-Detailansicht
GET	/teacher/lva/ajax/get-data	AJAX Lehrende-Detailansicht
GET	/international-office	Auslandsbüro-Hauptansicht
GET	/international-office/ajax/get-data	AJAX Auslandsbüro-Hauptansicht
GET	/international-office/export	Datenexport

Tabelle 9.1: Schnittstellenübersicht

9.4 Frontend

Das Frontend setzt sich aus den Sichten für die Benutzer/-innen und den entsprechenden Interaktionsmöglichkeiten mit der Applikation zusammen. Die GUI wird mithilfe von PUG-Templates bestimmt, die in *views* abgelegt sind. Diese PUG-Templates werden vom Controller als Antwort einer HTTP-Anfrage zurückgesendet. Die folgende Auflistung soll die Funktionsweise von PUG-Templates näher illustrieren:

```
1  const controller = {
2    ...
3    index(req, res) {
4      if (req.user) { // already logged in
5        res.redirect("/");
6      } else {
7        res.render("auth/login", {
8          message:
9            req.query.code == 403 ? "Ausgewählte Rolle nicht vorhanden" : ""
10       });
11     }
12   },
13   ...
14 }
```

Auflistung 9.1: Aufruf PUG-Template

Das Beispiel zeigt den Aufruf der Maske für die Rollenauswahl im Login-Controller. Ist keine aktive Rolle vorhanden, wird das PUG-Template *views/auth/login.pug* als Antwort verarbeitet (Zeile 7-10). Die Angabe eines PUG-Templates entspricht immer dem Pfad relativ zum *views*-Ordner und ohne der Endung .pug (siehe Zeile 7). Zusätzlich können dem Template Parameter (hier "message", Zeile 8 und 9) mitgegeben werden. Das Template sieht wie folgt aus:

```
1  extends layout
2  block content
3      form.form-signin(method='post' action='/login')
4          img.mb-4(src='/images/jku-logo.svg' width='100%' height='100em')
5          h3.mb-3.font-weight-normal Prüfungsvisualisierung
6          if message.length > 0
7              div.alert.alert-danger(role='alert')
8                  span #{message}
9          div.input-wrapper
10             label.sr-only(for='role') Rolle
11             select#role.form-select.form-control(name='role')
12                 option(value='teacher') Lehrende*r
13                 option(value='stuko') Stuko-Vorsitz
14                 option(value='vr') VR
15                 option(value='international-office') Auslandsbüro
16         div
17             button.btn.btn-primary(type='submit') Login
```

Auflistung 9.2: PUG-Rollenauswahl-Template

In Zeile 1 wird das Master-Template angegeben, von dem dieses Template erbt. In der Zeile 2 erfolgt die Angabe des PUG-Inhalts, der dann im Master-Template eingefügt wird. Aus der Zeile 6 kann entnommen werden, wie eine Bedingung über den mitgegebenen Parameter geprüft werden kann. Das dazugehörige Master-Template sieht wie folgt aus:

```
1  doctype html
2  html
3    head
4      title Exam Data Visualization
5      link(rel='stylesheet', href='/stylesheets/lib/bootstrap.min.css')
6      link(rel='stylesheet', href='/stylesheets/auth.css')
7
8      script(src='/javascripts/lib/jquery-3.5.1.min.js')
9      script(src='/javascripts/lib/bootstrap.bundle.min.js')
10     script(src='/javascripts/login.js')
11
12     meta(charset='UTF-8')
13     meta(name='viewport' content='width=device-width, initial-scale=1.0')
14   body(class='text-center')
15     block content
```

Auflistung 9.3: PUG-Master-Template

Das Master-Template importiert alle externen JS- und CSS-Frameworks. Der Fokus liegt hier in Zeile 15, wo der Inhalt des Child-Templates eingefügt wird. Durch diese Art von "Vererbung" kann eine Webanwendung in einzelne Komponenten gegliedert werden. Dadurch werden Duplikate vermieden und es herrscht eine bessere Übersicht.

9.5 Middleware

Wie bereits mehrfach erwähnt, gibt es sogenannte Middleware-Funktionen, die bei jeder Anfrage ausgeführt werden. Die Middleware-Funktionen werden in *app.js* initialisiert (erkennbar an *app.use()*). Neben einer Vielzahl an vordefinierten Middleware-Funktionen gibt es auch einige selbstdefinierte. Als Beispiel soll das untenstehende Beispiel dienen, das die Überprüfung einer Session definiert.

```
1 const sessionCheck = (req, res, next) => {
2   if (req.session.role && req.user) next();
3   else res.redirect("/login");
4 };
5
6 app.use(sessionCheck);
```

Auflistung 9.4: Middleware

Die Middleware (Zeile 1-4) ist sehr einfach gehalten. Ist eine Rolle in der Session gesetzt und existiert ein Benutzer-Objekt, kann die Applikation fortfahren (Zeile 2), ansonsten wird zur Rollenauswahl navigiert. In der Zeile 6 wird die Middleware in die Express-Applikation eingebunden und zusätzlich werden Schnittstellen definiert, an denen die Middleware nicht ausgeführt werden soll.

9.6 Controller

Ein Controller besitzt die Aufgabe, Anfragen zu verarbeiten. Die Aktionen bzw. Funktionen eines Controllers sind immer einer Schnittstelle zugeordnet und übernehmen die Verarbeitung der Anfrage. Im Controller findet im Wesentlichen die folgende Prozedur statt:

1. Auslesen der Anfrage-Parameter (falls vorhanden)
2. Laden der Daten aus dem Service-Block (falls notwendig)
3. Antworten mit einem PUG-Template, einer Weiterleitung oder einem Fehler

Der Ablauf eines Controllers soll mithilfe der Callback-Schnittstelle näher demonstriert werden. Auflistung 9.5 demonstriert die Überprüfung, ob die ausgewählten Rollen der Benutzer/-innen tatsächlich verwendet werden dürfen.

```
1  const controller = {
2    ...
3    // this gets called after successful auth on shibboleth
4    callback(req, res) {
5      let claimedRole = req.query.t;
6      const entitlements = ...
7      if (claimedRole === "teacher" && entitlements.includes(claimedRole)) {
8        req.session.role = "teacher";
9        res.redirect("/teacher");
10     } else if (claimedRole === "vr") {
11       let vr = ...
12       if (vr) {
13         req.session.role = "vr";
14         res.redirect("/vr");
15       } else ...
16     } else if (claimedRole === "stuko") {
17       let stuko = ...
18       if (stuko) {
19         req.session.role = "stuko";
20         res.redirect("/stuko");
21       } else ...
22     } else if (claimedRole === "international-office") {
23       let abroad = ...
24       if (abroad.find((x) => x.role === "international-office")) {
25         req.session.role = "international-office";
26         res.redirect("/international-office");
27       } else ...
28     } else {
29       ...
30     }
31   },
32   ...
33 };
34
35 module.exports = controller;
```

Auflistung 9.5: Login Controller

In Zeile 5 wird der Parameter einer Anfrage ausgelesen, welcher der selektierten Rolle entspricht. Auf Basis der ausgewählten Rolle wird überprüft, ob die Benutzer/-innen diese Rolle tatsächlich besitzen. Bei Lehrenden wird dafür bei dem entsprechenden Shibboleth-

Attribut (siehe Kapitel 5.1) auf die Existenz der URI `https://xam-viz.jku.at/roles/teacher` überprüft (Zeile 7). Bei den anderen Rollen wird überprüft, ob ein Eintrag in der Zugriffsliste mit der passenden AK-Nummer und Rolle vorhanden ist (siehe Zeilen 11-15, 17-21 und 23-27). Ist eine Überprüfung gelungen, wird eine Weiterleitung als Antwort zurückgesendet (Zeile 9, 14, 20 und 26), ansonsten wird ein Fehler geworfen (Zeile 29).

9.7 Service

Die Abfrageergebnisse aus der DB werden als JSON zurückgegeben. Der Service-Block hat die Aufgabe diese Ergebnisse den Controllern zur Weiterverarbeitung zu übermitteln. Dazu wird entweder das JSON aus dem Repository-Block ohne Veränderung weiterpropagiert oder es werden spezielle JSON-Strukturen aus den Abfrageergebnissen gebaut. Folgende spezielle JSON-Schemas sind definiert:

Daten über eine bestimmte LVA

```
{
      nrs: "LVA-Nummern ; getrennt",
      classId: "Klassen-Code",
      name: "LVA Titel",
      subname: "Untertitel",
      type: "LVA-Typ (VL, UE, ...)",
      participants: "Anzahl Teilnehmer",
      evaluations: "Anzahl Evaluierungen",
      averageGrade: "Durchschnittsnote",
      failureRate: "Durchfallrate",
      delay: "Verzögerung (nicht bei Lehrenden)",
      stats: [
        {
          grade: "1",
          count: "Anzahl Sehr Gut",
        },
        {
          grade: "2",
```

```
            count: "Anzahl Gut",
          },
          {
            grade: "3",
            count: "Anzahl Befriedigend",
          },
          {
            grade: "4",
            count: "Anzahl Genügend",
          },
          {
            grade: "5",
            count: "Anzahl Nicht Genügend",
          },
          {
            grade: "P",
            count: "Anzahl nicht-numerische positive
               Beurteilungen",
          },
          {
            grade: "N",
            count: "Anzahl nicht-numerische negative
               Beurteilungen",
          },
        ],
        diff: "Anzahl nicht beurteilt",
        comment: "Kommentar",
        disagreed: "Widerspruch J/N"
}
```

Daten für die Gliederung auf der Detailseite

```
{
      key: "Schlüssel (Beginnjahr, SKZ, Prüfungsdatum)",
      evaluations: "Anzahl Prüfungen",
      stats: [
        {
          grade: "1",
          count: "Anzahl Sehr Gut",
        },
        {
          grade: "2",
          count: "Anzahl Gut",
        },
        {
          grade: "3",
          count: "Anzahl Befriedigend",
        },
        {
          grade: "4",
          count: "Anzahl Genügend",
        },
        {
          grade: "5",
          count: "Anzahl Nicht Genügend",
        },
        {
          grade: "P",
          count: "Anzahl nicht-numerische positive
             Beurteilungen",
        },
        {
          grade: "N",
          count: "Anzahl nicht-numerische negative
             Beurteilungen",
```

```
        },
      ],
      list: [] // Daten für zweite Ebene
}
```

Daten über ein bestimmtes Studium (für das Auslandsbüro)

```
{
      study: "Bezeichnung des Studiums",
      type: "Studium-Typ (Bachelor, Master, ...)",
      skz: "Studienkennzahl",
      stats: [<Anzahl Sehr Gut>, <Anzahl Gut>, <Anzahl
         Befriedigend>, <Anzahl Genügend>, <Anzahl Nicht Genü
         gend>],
      statsP: ["Anzahl Sehr Gut (%)", "Anzahl Gut (%)", "
         Anzahl Befriedigend (%)", "Anzahl Genügend (%)", "
         Anzahl Nicht Genügend (%)"],
      evaluations: "Anzahl Evaluierungen",
      averageGrade: "Durchschnittsnote",
      failureRate: "Durchfallrate"
}
```

9.8 Repository

Der Zugriff auf die DB erfolgt mit dem *oracledb*-Modul[8]. Dazu wird beim Start der Applikation ein Connection-Pool erzeugt, aus dem die Verbindungen für die SQL-Anfragen entnommen werden. Die Verbindungen zur DB sind asynchron gestaltet und auf Performance optimiert. Im *repository*-Ordner sind die Zugriffe auf die DB enthalten. Das Senden eines SQL-Statements ist in *repository/db.js* definiert. Dort sind Funktionen enthalten, wie etwa das Erzeugen bzw. Schließen eines Pools oder das Senden einer SQL-Abfrage. Im untenstehendn Beispiel wird das Ausführen einer SQL-Abfrage gezeigt.

```
1  executeSqlStmt = async (sql, params = {}, options = {}) => {
2    let con;
3    try {
4      con = await oracledb.getConnection(config.db.poolAlias);
5      const result = await con.execute(sql, params, options);
6      return result.rows;
7    } catch (err) {
8      ...
9    } finally {
10     ...
11   }
12 };
13
14 getExams = async (term, spkkeys, akNr = null) => {
15   let sql =
16     `SELECT id "studentId", TO_CHAR(TO_DATE(epdat, 'DD.MM.YY'), 'RRRR-MM-DD
         ') "date", beginnsem_skz_pruefung "beginYear", ` +
17     `skz_pruefung "skz", epnote "grade", lvsem "sem", lvanr "nr", klaid "
         classId", kzbez "study", kztypbez "type" ` +
18     (akNr
19       ? `FROM TABLE (EXAMDV.LVANotenMWL('${akNr}', '${term}'))`
20       : `FROM TABLE (EXAMDV.LVANoten('${spkkeys}', '${term}'))`);
21
22   return await executeSqlStmt(sql);
23 };
```

Auflistung 9.6: Express Oracle-Zugriff

Die asynchrone Funktion **executeSqlStmt** (Zeile 1-12) fasst die Funktionalität zum Absenden einer SQL-Abfrage zusammen. Dazu wird in Zeile 4 eine Verbindung vom Connection-Pool angefordert und in Zeile 5 die SQL-Abfrage ausgeführt. Zu einer SQL-Abfrage gehö-

ren auch die Parameter eines Prepared-Statemens bzw. zusätzliche Optionen, wie z.B. die Timeout-Dauer der Abfrage. Die Funktion in den Zeilen 14-23 zeigt die Verwendung von **executeSqlStmt** (Zeile 22), indem ein SQL-Statement auf *LVANotenMWL* bzw. *LVANoten* ausgeführt wird. Das Ergebnis einer Abfrage wird als JSON zurückgegeben, wobei die abgefragten Spalten einer Tabelle die Attribute bilden.

9.9 Authentifizierung & Autorisierung

Die Benutzer/-innen dieser Applikation werden mittels Shibboleth authentifiziert. Wie im Kapitel 5.1 erklärt, werden die Attribute der Benutzer/-innen in die HTTP-Header des Webservers injiziert. Natürlich muss die Applikation bei jeder Anfrage diese HTTP-Header analysieren. Um diese HTTP-Header in Express auszulesen, wird das *passport-reverseproxy*-Modul[13] für Passport.js verwendet. Passport.js[14] ist eine Authentifizierungs-Middleware, welche verschiedenste Authentifizierungsmechanismen anbietet. Hier ein Auszug aus der aktuellen Implementierung in *auth.js*:

```
1  const passport = require("passport"),
2    ReverseProxyStrategy = require("passport-reverseproxy");
3
4  passport.use(
5    new ReverseProxyStrategy(
6      {
7        headers: {
8          "X-Forwarded-User": { alias: "username", required: true },
9          "X-Forwarded-Entitlements": { alias: "entitlements", required:
               false },
10         displayName: { alias: "name", required: false },
11         mail: { alias: "mail", required: false },
12       },
13     },
14     function (headers, user, done) {
15       var err = null;
16       return done(err, user);
17     }
18   )
19 );
20
21 app.use(passport.initialize());
```

```
22 app.use(
23   unless(
24     passport.authenticate("reverseproxy", {
25       session: false,
26       failureRedirect: "/login",
27     }),
28     "/login",
29     "/test"
30   )
31 );
```

Auflistung 9.7: Authentifizierung Passport.js

In den Zeilen 8-11 werden die auszulesenden HTTP-Header mitgegeben und definiert, ob es sich um verpflichtende HTTP-Header handelt. Das bedeutet, dass der *X-Forwarded-User* zwingend existieren muss, damit die Benutzer/-innen in der Applikation fortfahren können. Sind die Bedingungen erfüllt, wird ein Benutzer-Objekt erstellt, welches mit dem *req*-Objekt aufgerufen werden kann (*req.user*). Die ausgelesenen HTTP-Header werden in dieses Objekt mit dem alias-Namen eingebettet, d.h. der Zugriff auf die AK-Nummer würde mit *req.user.username* erfolgen. In den Zeilen 22-31 wird die Authentifizierung als applikationsbezogene Middleware mit den Ausnahmen (Zeile 28 und 29) definiert.

Die Autorisierung erfolgt, nachdem die Anfrage authentifiziert wurde. Dazu sind in den Controllern von den einzelnen Sichten eine Reihe von Überprüfungen definiert. Als Beispiel für die Logik soll die Stuko-Sicht dienen:

```
1  async getAjaxData(req, res) {
2      if (req.session.role !== "stuko" ||
3      !Utils.hasStudyPermission(req.query.spkkeys, req.session.studies) ||
4      Utils.canDisagree(req.query.term) ||
5      req.query.term < "2021W") {
6        res.status(403).send();
7      } else {
8        // proceed
9      }
10   },
```

Auflistung 9.8: Autorisierung

In den Zeilen 2-5 sind Überprüfungen definiert, die den Zugriff auf die Stuko-Sicht erlauben. In der Zeile 3 wird z.B. überprüft, ob die angefragten Studien tatsächlich erlaubt

sind. Die Schnittstelle liefert in den Fällen, wo die Autorisierung fehlschlägt, den HTTP-Code 403 zurück. Im Frontend wird dann eine entsprechende Meldung angezeigt.

9.10 Tests

Zum Testen der Implementierung wurde eine Reihe von Testfällen entwickelt. Im Wesentlichen werden zwei Frameworks dafür verwendet, *Mocha*[15] und *Chai*[16]. *Mocha* ist ein Test-Framework für Node, womit Testfälle (ähnlich zu JUnit in Java) formuliert werden können. Das Besondere am *Mocha*-Framework ist der Support für asynchrone Code-Exekutionen. Das *Chai*-Framework stellt eine Assertion-Bibliothek für Node dar und kann mit *Mocha* gepaart werden, um die Standard-Assertions zu erweitern.

Alle Testfälle sind im *test*-Ordner enthalten und testen hauptsächlich die Service-Schicht, da hier die Geschäftslogik stattfindet. Als Beispiel soll hier ein Auszug der Testfälle zu den *Utils*-Methoden dienen, womit auch das *Mocha*- und *Chai*-Framework demonstriert werden:

```
1  const assert = require("chai").assert; // Chai-Assertion-Framework
2  const ServiceUtils = require("../service/utils");
3
4  describe("Utils Test Suite", () => {
5    ...
6    describe("#getAvgDelay()", () => {
7      ...
8      it("should return the average delay as 1.375 for a set of exams", async
           () => {
9        // most visited term is 2020W
10       const exams = [
11         { beginYear: "2020S" },
12         { beginYear: "2019S" },
13         { beginYear: "2018W" },
14         { beginYear: "2020W" },
15         { beginYear: "2020S" },
16         { beginYear: "2020W" },
17         { beginYear: "2020W" },
18         { beginYear: "2019W" },
19       ];
20
21       const expected = 1.375;
```

```
22        const actual = await ServiceUtils.getAvgDelay(exams);
23        assert.equal(actual, expected);
24      });
25      it("should return the average delay as 0 for a set of exams", async ()
            => {
26        const exams = [
27          { beginYear: "2020S" },
28          { beginYear: "2020S" },
29          { beginYear: "2020S" },
30          { beginYear: "2020S" },
31        ];
32
33        const expected = 0;
34        const actual = await ServiceUtils.getAvgDelay(exams);
35        assert.equal(actual, expected);
36      });
37      ...
38    });
39    ...
40  });
```

Auflistung 9.9: Utils Tests

Im obigen Beispiel eines Testfalls kann folgendes beobachtet werden:

- Mit dem Schlüsselwort *describe* kann in *Mocha* eine Ansammlung von Testfällen (Test-Suite) spezifiziert werden. Eine Test-Suite akzeptiert einen Titel und eine Funktion zum Ausführen von weiteren Test-Suites oder konkreten Testfällen. In Zeile 4 wird eine Test-Suite für die *Utils*-Klasse definiert. In Zeile 6 wird eine Test-Suite für eine konkrete Methode erstellt.
- Mit dem Schlüsselwort *it* werden die konkreten Testfälle in einer Test-Suite spezifiziert. Ein Testfall akzeptiert einen Titel und eine Funktion, welches die Testlogik implementiert. In den Zeilen 8-24 bzw. 25-36 wird die Logik zum Berechnen der Durchschnittsverzögerung getestet.
- Das *Chai*-Framework kann in der Zeile 23 oder 35 beobachtet werden. Mit einem *assert.equal* wird der zu erwartende mit dem aktuellen Wert verglichen.

Zusätzlich werden auch die internen JSON-Strukturen, die in der Service-Schicht aus den Daten der DB gebaut werden, getestet (siehe Abschnitt 9.7). Hierfür muss kurzzeitig

eine Verbindung zur DB hergestellt werden. Der Vergleich der JSON-Strukturen wird mit einer Erweiterung des *Chai*-Frameworks durchgeführt. Der folgende Auszug, der das JSON-Schema der LVA-Statistik testet, soll das Konzept näher beschreiben:

```
const chai = require("chai");
chai.use(require("chai-json-schema"));
const assert = chai.assert;
const StukoService = require("../service/stukoService");
const DB = require("../repository/db");

describe("StukoService Test Suite", () => {
  before(async () => {
    await DB.init();
  });
  after(async () => {
    await DB.close();
  });
  ...
  describe("#getStukoLVAStatistics()", () => {
    it("should return correct JSON contract", async () => {
      const expectedSchema = {
        title: "LVA statistic schema",
        type: "array",
        items: {
          type: "object",
          required: [
            ...
            "nrs",
            "classId",
            "type",
            "failureRate",
            ...
          ],
          properties: {
            ...
            nrs: {
              type: "string",
              pattern: "^([0-9a-zA-Z]+)(;[0-9a-zA-Z]+)*$",
            },
            classId: {
              type: "string",
              minLength: 1,
```

```
39              },
40              failureRate: {
41                type: "number",
42                minimum: 0,
43                maximum: 1,
44              },
45              ...
46            },
47          },
48        };
49        const actual = await StukoService.getStukoLVAStatistics(
50          "2021S",
51          "166,193",
52          "name",
53          "0"
54        );
55        assert.jsonSchema(actual.lvas, expectedSchema);
56      });
57    });
58    ...
59  });
```

Auflistung 9.10: JSON-Schema Tests

Folgendes kann beobachtet werden:

- In Zeile 2 wird die Erweiterung von *Chai* geladen.
- Mit dem Schlüsselwort *before* bzw. *after* können Aktionen definiert werden, die vor und nach Ausführung einer Test-Suite ausgeführt werden sollen. In dem Fall wird eine Verbindung zur DB hergestellt bzw. wieder beendet (siehe Zeilen 8-10 bzw 11-13).
- In den Zeilen 17-48 wird eine Definition des JSON-Schemas angegeben. Hierfür kann der Typ des JSONs und die einzelnen Attribute definiert werden. Für die Attribute können ebenfalls die Typen, Formate, etc. definiert werden. In Zeile 19 wird angegeben, dass ein JSON-Array erwartet wird. Die Inhalte des Arrays sind vom Typ *object* (Zeile 21) und müssen verpflichtende Attribute besitzen (Zeilen 22-29). Das Attribut aus Zeile 32 muss vom Typ *string* sein und zum angegebenen regulären Ausdruck passen. Das Attribut aus Zeile 40 muss vom Typ *number* sein,

und der Wert muss zwischen 0 und 1 liegen. Auf diese Art und Weise kann eine Validierung für JSON-Schemas gebaut werden.

- In Zeile 55 sieht man, wie zwei Schemas miteinander verglichen werden können.

Die Tests können mit dem Befehl **npm test** ausgeführt werden. Allerdings muss darauf geachtet werden, dass eine Verbindung zur DB hergestellt werden kann. Die Ergebnisse der Testfälle können dann wie folgt aussehen:

```
oktayakguel@debian:~/xam-viz$ npm test

> xam-viz@1.0.0 test
> NODE_ENV=test mocha -r dotenv/config --timeout 120000

  StukoService Test Suite
Init DB pool
    #getStukoLVAStatistics()
      ✓ should return correct JSON contract (737ms)

  Utils Test Suite
    #getAvgDelay()
      ✓ should return the average delay as 1.375 for a set of exams
      ✓ should return the average delay as 1.25 for a set of exams
      ✓ should return the average delay as 0 for a set of exams
    #isInformatikStuko()
      ✓ should return true for SKZ 521
      2) should return true for SKZ 921

  5 passing (16s)
  1 failing

  1) Utils Test Suite
       #isInformatikStuko()
         should return true for SKZ 921:

     AssertionError: expected false to be true
```

```
+ expected - actual

-false
+true

at Context.it (test/utils.js:212:14)
```

9.11 Konfiguration

Über diverse Umgebungsvariablen kann die Applikation entsprechend konfiguriert werden. In Tabelle 9.2 werden alle existierenden Umgebungsvariablen aufgelistet. Die Umgebungsvariablen werden aus einer **.env**-Datei ausgelesen, die im Format **KEY=VALUE** angegeben werden. Da diese Datei sensible Informationen, wie das Passwort für die DB, enthält, ist sie nicht im Git-Repository enthalten und muss initial im Wurzelverzeichnis der Applikation erzeugt werden. Dazu existiert eine Vorlage im Git-Repository, **.env.example**, welche kopiert und umbenannt werden kann. Wenn der Wert einer Umgebungsvariable nicht gesetzt wird, wird der Standardwert automatisch übernommen. Diese Umgebungsvariablen werden beim Start der Applikation eingelesen und in den Konfigurationsdateien im *config*-Ordner verwendet. Falls eine Umgebungsvariable verändert wird, ist ein Neustart der Applikation notwendig.

Schlüssel	Default	Bedeutung
PORT	3000	Port, an dem die Express-Applikation hinhört
NODE_ENV	development	Auskunft über Entwicklungs- oder Produktivsystem
SESSION_SECRET	keyboard cat	Secret für die Session
DB_CONNECT_STRING	tohk01.im.jku.at/todb01	Verbindungsstring zur DB
DB_USER		DB Benutzer
DB_PASSWORD		DB Passwort
DB_POOL_PING_INTERVAL	30	Intervall für Pings an den DB-Pool (s)
DB_POOL_INCREMENT	1	Anzahl der Verbindungen, um die der Pool erhöht werden soll
DB_POOL_MIN	4	Mindestanzahl der Pool-Verbindungen
DB_POOL_MAX	4	Maximalanzahl der Pool-Verbindungen
DB_ENABLE_STATISTICS	false	Aktivieren der Statistikausgaben
DB_QUEUE_TIMEOUT	60000	Maximale Wartezeit einer Verbindung in der Warteschleife (ms)
DB_QUEUE_MAX	500	Maximal erlaubte Anzahl an Verbindungen in der Warteschleife
DB_STATEMENT_CACHE_SIZE	30	Anzahl der gespeicherten Statements im Cache
DB_HEALTHCHECK_TIMEOUT	1000	Maximale Dauer eines Healthchecks des Pools
DB_CALL_TIMEOUT	0	Timeout für das Ausführen eines Statements

Tabelle 9.2: Umgebungsvariablen

10 Installation & Wartung

Bevor die Applikation auf einem Server installiert werden kann, müssen folgende Elemente am jeweiligen Server vorhanden sein:

1. git[17]
2. Node.js[18]
3. npm[19]
4. Oracle Instant Client[20]

Bei Punkt 3 handelt es sich um den Packet-Manager von Node, und Punkt 4 wird benötigt, um sich mit der Oracle-DB zu verbinden.

Sind die oben genannten Voraussetzungen erfüllt, kann der Source-Code vom Git-Repository (siehe Kapitel 9) heruntergeladen und die zusätzlichen Node-Bibliotheken mit *npm install* installiert werden. Zuletzt muss noch eine *.env*-Datei für die Umgebungsvariablen erstellt werden, welche am besten mithilfe der *.env.example* erzeugt wird. Alle Umgebungsvariablen im Abschnitt 9.11, die keine Default-Werte besitzen, müssen entsprechend gesetzt werden. Als Beispiel der einzelnen Installationsschritte werden die Befehle der Kommandozeile aus einer Debian-VM illustriert:

```
oktayakguel@debian:~$ node -v # check node
v10.24.1
oktayakguel@debian:~$ npm -v # check npm
7.21.0
oktayakguel@debian:~/xam-viz$ git --version # check git
git version 2.20.1
oktayakguel@debian:~$ sqlplus64 # check oracle instant client
SQL*Plus: Release 21.0.0.0.0 - Production on Sat Jul 2 16:40:42 2022
```

```
Version 21.3.0.0.0
Copyright (c) 1982, 2021, Oracle.  All rights reserved.
Enter user-name:
oktayakguel@debian:~$ git clone https://stash.im.jku.at/scm/xam/xam-viz.git
oktayakguel@debian:~$ cd xam-viz/
oktayakguel@debian:~/xam-viz$ npm install # skipping output
oktayakguel@debian:~/xam-viz$ cp .env.example .env # generate .env
```

Wenn die Installation erfolgreich abgeschlossen wurde, kann die Applikation mit dem Befehl **node bin/www** gestartet werden.

Die Wartung der Applikation sieht vor, Bugs zu beheben und eine aktualisierte Liste der Stuko-Vorsitzenden samt den zuständigen Studien einzuspielen. Das kann lokal in einer ausgecheckten Version des Repository erfolgen oder direkt am Web-Client von Bitbucket. Sobald ein Push auf den prod-Zweig des Repository erfolgt, wird eine Jenkins-Pipeline gestartet, welche automatisiert die Applikation neu kompiliert und einspielt. Am Ende werden die Administratorinnen und Administratoren mit einer E-Mail über den Status des Deployments benachrichtigt.

11 Fazit & Ausblick

Zum Analysieren der Prüfungsergebnisse an der JKU wurde ein webbasiertes Tool namens, *xam-viz*, entwickelt. Die Applikation visualisiert Kennzahlen, wie den Notendurchschnitt, die Durchfallrate, den Notenspiegel, usw. der Prüfungsergebnisse für bestimmte Benutzergruppen (Lehrende, Stuko-Vorsitz, VR-Lehre und Auslandsbüro) und bietet die Möglichkeit, die Ergebnisse nach bestimmten Kriterien zu filtern und zu gliedern. Dadurch haben Stuko-Vorsitzenden einen besseren Überblick über die Performance-Zahlen der zuständigen Studien und können somit aktiv die Qualität der Lehre steigern. Zusätzlich gibt es für Lehrende die Sicherheitsfunktion, der Veröffentlichung der LVA-Ergebnisse an den Stuko-Vorsitz innerhalb einer Frist zu widersprechen bzw. zu kommentieren. Diese Maßnahme dient dem Schutz der Lehrenden, damit die Prüfungsergebnisse nicht zu Vorurteilen seitens der Stuko-Vorsitzenden führen.

Die Entwicklung der Applikation war großteils von externen Beteiligten abhängig. Da *xam-viz* eine Shibboleth-Authentifizierung vorsieht, muss die Anwendung vom IM gehostet und betreut werden. Dadurch ist die Installation der Applikation auf dem Server eine Blackbox und konnte nur in Zusammenarbeit mit der IM eingerichtet werden.

Nichtsdestotrotz, bietet *xam-viz* viele Weiterentwicklungsmöglichkeiten. Einerseits können, z.B. die Filter um eine Suche erweitert, andererseits können die Gliederungskriterien um weitere Attribute ergänzt werden.

Literatur

[1] Scott Cantor und T Scavo. „Shibboleth architecture“. In: *Protocols and Profiles* 10 (2005), S. 16 (siehe S. 12).

[2] John Hughes und Eve Maler. „Security assertion markup language (saml) v2. 0 technical overview“. In: *OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08* 13 (2005) (siehe S. 12).

[3] *Shibboleth: Attribute Access*. URL: https://shibboleth.atlassian.net/wiki/spaces/SP3/pages/2065335257/AttributeAccess (besucht am 13. 09. 2022) (siehe S. 13).

[4] *Lexikon der Informatik*. ger. 2011. URL: http://dx.doi.org/10.1007/978-3-642-15126-2 (siehe S. 17, 21).

[5] Stefan Tilkov und Steve Vinoski. „Node. js: Using JavaScript to build high-performance network programs“. In: *IEEE Internet Computing* 14.6 (2010), S. 80–83 (siehe S. 17).

[6] *Node.js web application framework*. URL: https://expressjs.com/ (besucht am 15. 06. 2022) (siehe S. 18).

[7] *Get started with table functions 1: Overview*. URL: https://livesql.oracle.com/apex/livesql/file/tutorial_GSOTSK8FWLYZOG5CJJ9KPX7RX.html (besucht am 15. 06. 2022) (siehe S. 20).

[8] *Node.js node-oracledb version 5.4*. URL: https://oracle.github.io/node-oracledb/ (besucht am 15. 06. 2022) (siehe S. 20, 49).

[9] Max Pekarsky. *Does your web app need a front-end framework?* Feb. 2020. URL: https://stackoverflow.blog/2020/02/03/is-it-time-for-a-front-end-framework/ (siehe S. 21, 22).

[10] URL: https://pugjs.org/ (besucht am 15. 06. 2022) (siehe S. 22).

[11] JS Foundation - js.foundation. URL: https://jquery.com/ (besucht am 07. 08. 2022) (siehe S. 23).

[12] Jacob Thornton Mark Otto. *Bootstrap*. URL: https://getbootstrap.com/ (besucht am 07. 08. 2022) (siehe S. 23).

[13] Steve-Jansen. *Steve-Jansen/passport-REVERSEPROXY: Reverse HTTP proxy authentication strategy for passport and node.js*. URL: https://github.com/steve-jansen/passport-reverseproxy (besucht am 15. 08. 2022) (siehe S. 50).

[14] URL: https://www.passportjs.org/ (besucht am 15. 08. 2022) (siehe S. 50).

[15] *The fun, simple, flexible JavaScript test framework*. URL: https://mochajs.org/ (besucht am 25. 06. 2022) (siehe S. 52).

[16] *Chai assertion library*. URL: https://www.chaijs.com/ (besucht am 25. 06. 2022) (siehe S. 52).

[17] URL: https://git-scm.com/ (besucht am 02. 07. 2022) (siehe S. 59).

[18] Node.js. URL: https://nodejs.org/en/ (besucht am 02. 07. 2022) (siehe S. 59).

[19] *NPM*. URL: https://www.npmjs.com/ (besucht am 02. 07. 2022) (siehe S. 59).

[20] *Oracle Instant Client*. URL: https://www.oracle.com/database/technologies/instant-client.html (besucht am 02. 07. 2022) (siehe S. 59).