Studienarbeit im Fach „Software Engineering 2“

Thema: Notenrechner

Markus Österle
Maximilian Schreiber
Tobias Schmidbauer
Stefan Memmel
Christoph Kammerer

Inhaltsverzeichnis

1 Einleitung & Motivation

Die folgende Gruppenarbeit aus dem Fach „Software Engineering 2“ beschäftigt sich mit einem für den Studiengang „Verwaltungsinformatik“ tatsächlich relevanten Problem, durch die Aufteilung unseres Studiengangs auf zwei Hochschulen (FHVR AIV & Hochschule Hof) und der damit verbundenen Aufteilung der Prüfungsleistungen, ergibt sich die Notwendigkeit einer zentralen Plattform, in die zum einen Noten eingetragen werden können (z.B. durch die Verwaltung oder auch berechtigte Dozenten) und zum anderen auch eingetragene Noten durch die Studierenden abgerufen werden können. Idealerweise soll bei dieser Gelegenheit auch eine Berechnung der Zwischen- und Endnoten erfolgen, um die im Studiengang kursierenden Exceltabellen durch eine rechtssichere und in jedem Fall richtig rechnende Plattform abzulösen.

Zur Realisierung einer solchen Plattform werden im folgenden Technologien eingesetzt, die im Rahmen der Lehrveranstaltungen

- Objektorientiertes Programmieren 1 & 2
- Algorithmen und Datenstrukturen
- Serverseitiges Programmieren mit Java
- Software Engineering 1 & 2

erlernt wurden.

Details zu den eingesetzten Techniken finden sich in den folgenden Kapiteln.

2 Lasten- und Pflichtenheft

2.1 Anforderungsliste

In der ersten Sitzung unserer Projektgruppe wurde eine Anforderungsliste an das Projekt „Notenrechner“ ausgearbeitet. Diese behandelt zu allererst Anforderungen aus zwei Sichten, auf der einen Seite Anforderungen, die die Verwaltung haben könnte und auf der anderen Seite Anforderungen der Studierenden. Diese Anforderungen wurde im Projektverlauf, wie in der Vorlesung gelernt, in ein Lasten- und Pflichtenheft umgearbeitet:

2.1.1 Hochschule

- Ortsunabhängiger Aufruf
- Dozenten sollen von jedem Ort der Erde Noten einpflegen können
- Dozenten können ihre Prüfungen aufrufen und auswerten
- Ein Admin (Studiengangsleiter) soll die Gewichting der Noten ändern können
- Admin soll Studiengänge modifizieren (wie z.B. Fächer zwischen den Semestern verschieben) und komplett neu anlegen können
- Admin soll die Fähigkeit besitzen, neue Studenten anzulegen bzw. bestehende Studenten zu modifizieren (z.B. Namen ändern)
- Statistik mit grafischer Aufarbeitung
  - Aufruf von jedem, persönliche Einzelnoten können nur vom jeweiligen (angemeldeten) Studenten gesehen werden
  - Nachprüfungen werden wie Erstversuch behandelt

  - Möglichkeit Leistungsnachweise in die Prüfungsnote einrechnen oder nicht

- Reminder (E-Mail Benachrichtigung) für Dozenten zur Eingabe der Prüfungsnoten (Mail 4 Wochen nach Prüfungsende)

- Reminder für Studenten wenn neue Noten vorliegen (E-Mail)

- Studiengang kann per Drag&Drop zusammengestellt werden (durch den Studiengangsleiter)

- Studiengang kann dupliziert und bearbeitet werden

- Authentifizierung: Über vorhandene User (JAAS mit LDAP)

  - benötigte Rollen:

    * Dozent/Prüfungsamt

      · kann nur Noten eintragen

      · seine Prüfungen einsehen

    * Studiengangsleiter (Admin)

      · kann Gewichtung ändern

      · kann Personen hinzufügen/ändern

      · kann Studiengänge hinzufügen/ändern

      · kann Noten ändern (z.B. bei Fehlern)

    * Student

      · Einsicht Wunsch-/Traumnoten

2.1.2 Student

- Reminder, wenn neue Noten eingetragen wurden

- Wunschnoten können eingetragen werden und werden überschrieben, wenn „echte“ Noten vorhanden sind ODER „echte“ Noten stehen neben den Traumnoten

- Statistik (farblich aufbereitet in der Notenübersicht, z.B. grün = über dem Durchschnitt & bestanden, rot = durchgefallen, gelb = bestanden, aber unter dem Durchschnitt)
- Ortsunabhängiger Abruf
- Übersicht über nicht bestandene Klausuren bzw. unterpunktete Leistungsnachweise
- Zwischennoten (welche Noten hatte ich in der Zwischenprüfung?), Anzeige welche Noten noch notwendig sind um zu bestehen (Leistungsnachweise)

2.1.3 Zusammenfassung Anforderungen

Noch notwendig????

- 2 Frontends (Verwaltung & Studierende)
- individualisierbare Notenliste/-berechnung pro Jahrgang
- Studiengangspezifisch, d.h. Programm nur für Vinf oder allgemeingültig?
- Wenn allgemeingültig müsste der Administrator in der Lage sein Studiengänge mit Spezifikation zu erstellen - kann schwierig werden
- Ablage der Noten in einer Datenbank -> Diskussionsbedarf, SQL oder NoSQL
- Generierung von Testdaten (Mock data) für die Datenbank
  - Ist möglich durch CSV Dateien der ersten Semester
- Ablage der Noten kann nur durch den Administrator/Dozenten erfolgen
- graphische Aufbereitung der Noten in Diagrammen
- statistische Kennzahlen berechen (Standartabweichung, Durchschnittsnote,...)
- Statistikfunktionen für den Jahrgang
- Farbliche Abhebung der Noten, ob durchgefallen (rot), über- (grün)/unterdurchschnittlich (gelb) usw...

- Durchschnittsnote für alle sichtbar (kann bedenklich sein wenn nur zwei Studenten das Fach geschrieben haben)
- Authentifizierung notwendig über JAAS
- Desktop Client mit JavaFX (optional?)
- Umsetzung in „gesprochene Noten“ (sehr gut, gut, usw...)
- Eintragung von „Traumnoten“ der Studenten und anschließende Berechnung der „resultierenden/prognostizierenden“ Endnote, die überschrieben werden durch Eintragung der echten Note durch den Dozenten
- Technologie fuer die Abhängigkeitsverwaltung?
  - Ant
  - Maven
  - Gradle
- Mobile Devices über App oder AngularJS? Wenn App, welche Plattformen?

2.2 Lastenheft

2.2.1 Zielbestimmung

Es soll eine Software entwickelt werden, die eine einfache Eingabe und Berechnung von Noten gemäß der gesetzlichen Bestimmungen erlaubt. Die Software soll sowohl von der Verwaltung intern, als auch von den Studierenden benutzt werden. Ein geeignetes Berechtigungsmodell muss implemetiert werden. Die Software soll von überall abrufbar sein, maximal portierbar sein und somit auf möglichst vielen Plattformen abgerufen werden können.

2.2.2 Produkteinsatz

Der Einsatz des Produktes ist auf einem zentralen Server der FHVR vorgesehen. Dieser Server soll nur über das „FHVR Intranet“ erreichbar sein. Die Authentifizierung soll im

Endausbau über bereits vorhandene AD (Active Directory) Konten realisiert werden.

2.2.3 Produktübersicht

Es soll ein Webservice mit mindestens zwei voneinander getrennten Oberflächen geschaffen werden. Der Zugang zu den Oberflächen soll sich nach den Benutzern zugeordneten Rollen richten. Es sind mindestens drei Rollen vorzusehen:

- Studierende
- Dozenten
- Administrator

Für die Speicherung der Noten ist eine geeignete performante Speichermethode vorzusehen (bspw. SQL oder NoSQL). Nach Möglichkeit soll für die Realisierung Software eingesetzt werden, für die keine Lizensierungskosten anfallen und deren Wartbarkeit und Sicherheit trotzdem auf absehbare Zeit gesichert ist.

2.2.4 Produktfunktionen

Es sind folgende Funktionen vorzusehen:

- Persistente Speicherung der Daten mithilfe einer geeigneten Technologie
- Eingabe von Prüfungsleistungen
- Für Studierende ist die Möglichkeit der Eingabe von „Wunschnoten“ vorzusehen, diese sollen genau wie die „echten“ Noten behandelt werden, d.h. gespeichert werden und es sollen aus diesen Werten die Zwischen- und Endnoten berechnet werden.
- Farbliche Markierung für Notenwerte, die im Grenzbereich und unterhalb der Anforderungen liegen (niedrige Priorität), sodass für die Studierenden auf einen Blick zu erfassen ist, wie ihre Leistungen einzustufen sind
- Berechnung muss entsprechend der gesetzlichen Vorgaben umgesetzt werden
- Die Anwendung soll modular gehalten werden, d.h. es sollen komplett neue Studiengänge angelegt werden können

- Benutzeroberfläche soll Plattformunabhängig sein, dies umfasst auch mobile Endgeräte. Die Oberfläche ist so auszulegen, dass sie auch auf mobilen Endgeräten bedient werden kann.

- Authentifizierungstechnologie muss vorhandenes Active Directory (AD) unterstützen

- Rollenbasiertes Zugriffsmodell, die Zuordnungen zu den Rollen sollen aus dem AD übernommen werden. Es sind 3 Rollen mit den folgenden Berechtigungen vorzusehen:

  - Administrator

    * Anlegen von neuen Studiengängen (inkl. Eingabe der abzulegenden Prüfungsleistungen und Notengewichtung)

    * Modifizieren von vorhandenen Studiengängen

    * Eintragung von Noten (ohne Beschränkungen auf bestimmte Fächer)

    * Anlegen von neuen Nutzern (sofern nicht automatisch realisiert)

    * Vergabe der Berechtigungen für Dozenten (Zuteilung der Fächer)

  - Dozenten

    * Eintragen von abgelegten Prüfungsleistungen für zugewiesene Fächer

    * Abruf der aus den eingetragenen Prüfungsleistungen berechneten Statistiken

  - Studierende

    * Eintragen von Wunschnoten, diese sind genauso zu behandeln wie eingetragene „Echtnoten“

    * Abrufen bereits abgelegter Prüfungsleistungen und (daraus) berechneter Zwischen- und Endnoten

    * Abruf statistischer Daten (Durchschnitt, Median, etc.) zu den eingepflegten Prüfungsleistungen

2.2.5 Produktdaten

2.2.6 Produktleistungen

Das Produkt stellt eine Plattform zur Verfügung, auf der aus eingegeben Prüfungsleistungen Statistiken abgeleitet werden, die eine geordnete und berechnete Ausgabe zur Verfügung stellt.

2.2.7 Qualitätsanforderungen

Es ist sicherzustellen, dass die angebotene Software die Noten entsprechend der gesetzlichen Vorgaben richtig berechnet. Der Datenschutz muss in jeder Betriebssituation gewahrt sein. Es ist sicherzustellen, dass die benutzten Technologien und Programme in den nächsten Jahren noch Weiterentwicklung und Support bekommen.

2.2.8 Ergänzungen

2.3 Pflichtenheft

2.3.1 Zielbestimmung

2.3.1.1 Musskriterien

- Authentifizierung (vorerst noch nicht per LDAP)
- Oberfläche für Studierende mit Abfrage der Noten und Möglichkeit zur Eingabe der Wunschnoten
- Oberfläche für Dozenten mit der Möglichkeit Prüfungsleistungen einzugeben
- Plattformunabhängigkeit

2.3.1.2 Wunschkriterien

- Farbliche Markierung der Prüfungsleistungen
- Mobile Oberfläche
- Eigene Anwendung zur Verwendung auf Desktop PCs (Desktop Client)
- Anlegen von komplett neuen Studiengängen mit eigenen Fächern und Gewichtungen der Prüfungsleistungen durch den Administrator
- Bereits geschriebene Note automatisiert als Wunschnote eintragen (Oberfläche für Studierende)

2.3.1.3 Abgrenzungskriterien

2.3.2 Produkteinsatz

2.3.2.1 Anwendungsbereiche

2.3.2.2 Zielgruppen

Zielgruppe der Anwendung sind im wesentlichen die Studierenden und die Dozenten der Hochschulen. Die Verwaltung der FHVR als Dienstherr der Verwaltungsinformatiker ist von eher untergeordneter Bedeutung für die Entwicklung, da diese einen zahlenmäßig sehr kleinen Teil der gesamten Nutzerschaft ausmacht.

2.3.2.3 Betriebsbedingungen

2.3.3 Produktumgebung

2.3.3.1 Software

Um die Kosten für Lizenzen und Support möglichst gering zu halten, wird der Einsatz von Linux als Serverbetriebssystem empfohlen, als Datenbank wird MySQL oder MariaDB empfohlen und als Applicationserver soll Wildfly in einer aktuellen Version zum Einsatz kommen.

2.3.3.2 Hardware

Das Produkt ist auf allen Plattformen lauffähig, die die benötigten Softwareprodukte unterstützen, neben x86_64 also bspw. auch SPARC und ARMv6/v7/v8.

2.3.3.3 Orgware

TODO: Sicherheitsanforderungen, Benutzerhandbücher

2.3.3.4 Produkt – Schnittstellen

- LDAP als Schnittstelle zum vorhandenen Active Directory (AD)

2.3.4 Produktfunktionen

2.3.4.1 Produktspezifisch

2.3.5 Produktdaten

2.3.5.1 Produktspezifisch

2.3.6 Produkt - Leistungen

2.3.7 Benutzungsoberfläche

Als Benutzeroberfläche kommt im ersten Entwicklungsschritt die Web Technologie Java Server Faces (JSF) zum Einsatz. In weiteren Entwicklungsschritten ist angedacht eine (lokale) Java Application mit einer JavaFX Oberfläche zur Verfügung zu stellen.

2.3.8 Qualitäts-Zielbestimmung

2.3.9 Globale Testszenarien/Testfälle

2.3.10 Entwicklungsumgebung

Als Java (EE) - Entwicklungsumgebung kommt Netbeans in der Version 8.1 zum Einsatz.

Für die Entwicklung der Datenbankschemata und Skripte wird die MySQL Workbench in der Version 6.3 zum Einsatz kommen.

Als Testumgebung wird eine virtuelle Maschine auf Suse Linux Basis verwendet.

2.3.11 Ergänzungen

2.3.12 Glossar, Begriffslexikon

3 Verwendete Technologien

Im ersten Meeting (Kick-Off meeting) des Teams wurden die zu verwendenden Softwareversionen festgelegt. Diese wurden im Verlaufe des Projekts nur noch aufgrund äußerer Begebenheiten (bekannte Fehler mit Fix in höherere Version, finale Version) angepasst. Die jeweils festgelegte Version und warum diese unter Umständen noch angepasst wurde, findet sich im jeweiligen Unterpunkt.

3.1 Entwicklung

3.1.1 Java SDK

Als Java Umgebung kam während der ganzen Projektdauer das Oracle Java SDK Version 8 Update 60 zum Einsatz.

3.1.2 Entwicklungsumgebung - Netbeans

festgelegte Version: **8.1RC2**
während des Projektverlaufs verändert? **ja**
Grund: **erscheinen der finalen Version**
geändert zu Version: **8.1 final**

3.1.3 SQL Editor - MySQL Workbench

3.1.4 Versionsverwaltung - GIT

Als Versionsverwaltung und zum Austausch des aktuellen Versionsstand des Projektes wurde während des gesamten Projektes Git eingesetzt. Das zentrale Repository liegt

auf dem webbasierten Online-Dienst Github: `https://github.com/themanwhosold/notenrechner`

3.1.5 Bibliotheksverwaltung mit Maven

3.2 Test

3.2.1 Unit Tests mit JUnit

Als Unit-Test Framework wurde JUnit eingesetzt und Testpackages für jede Klasse erzeugt. In diesen Testklassen wurden Testfälle definiert und bei jedem Build auf korrektheit überprüft. Hierdurch wurde sichergestellt, dass neu hinzugekommene Funktionalitäten nicht die Funktionsfähigkeit bereits vorhandener Komponenten verändert. Weiterhin wurde für den Test der Java EE Webanwendung Mockito in Verbindung mit Arquillian eingesetzt, da zur Laufzeit die jeweilige Unit getestet werden muss. Hierdurch können die Schnittstellen der jeweiligen Objekte definiert und angesteuert werden.

3.2.2 Continous Integration Tests mit Travis, Jenkins und Sonarqube

Um einen möglichst fehlerfreien Master Zweig zu erhalten wurden Pull Requests nur nach fehlerfreiem Durchlauf der beiden CI Tools Travis & Jenkins in den Master Zweig übernommen.

3.2.2.1 Travis

Travis[1] wird von Github als Tool für CI angeboten, das macht die Integration in die Github Oberfläche sehr einfach, man muss sich nicht mit Hooks und Anmeldeinformationen rumschlagen und bei Bearbeitung eines Pull Requests ist auf einen Blick der Status des Travis Builds sichtbar.

[1] `https://travis-ci.org/`

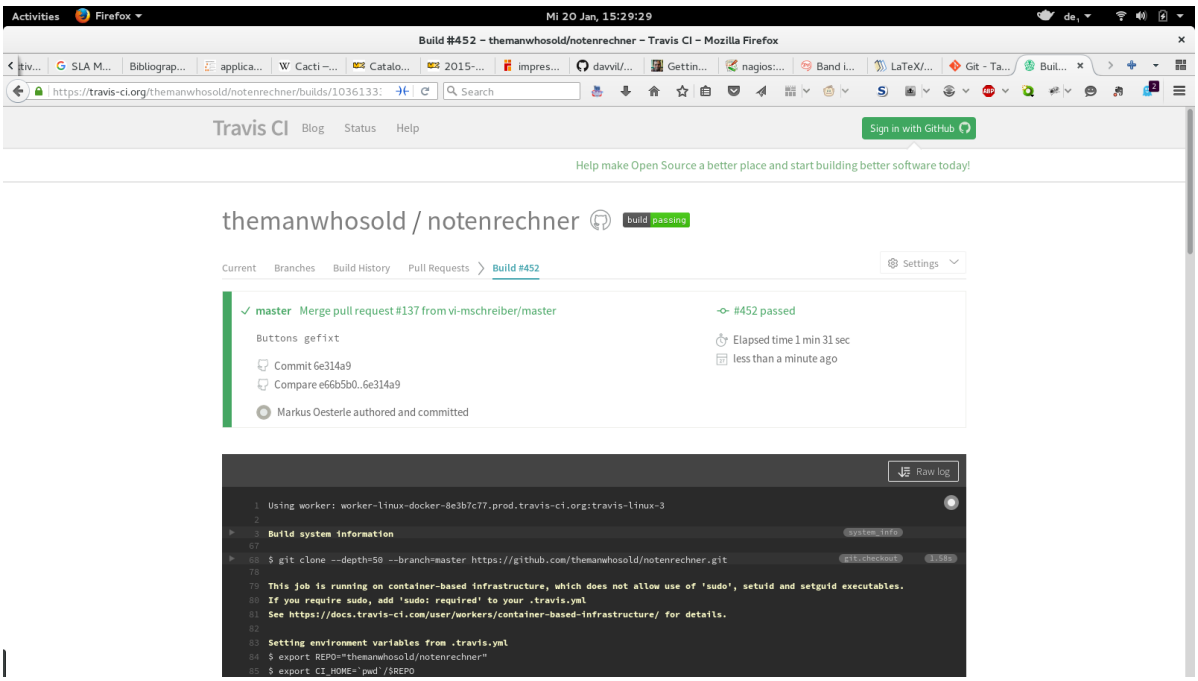

Abbildung 3.1: Screenshot eines Travis Builds

3.2.2.2 Jenkins

Jenkins wurde als zweites CI Tool eingesetzt

3.2.2.3 Sonarqube

Zur Analyse der Codequalität wurde Sonarqube[2][3] eingesetzt, die Analyse wurde von Jenkins bei jedem Build angesteuert. Sonarqube ist auf dem selben Server installiert wie Jenkins und liefert keinerlei Werte zurück an Git (anders als Jenkins selbst).

[2] http://www.sonarqube.org/
[3] https://de.wikipedia.org/wiki/SonarQube

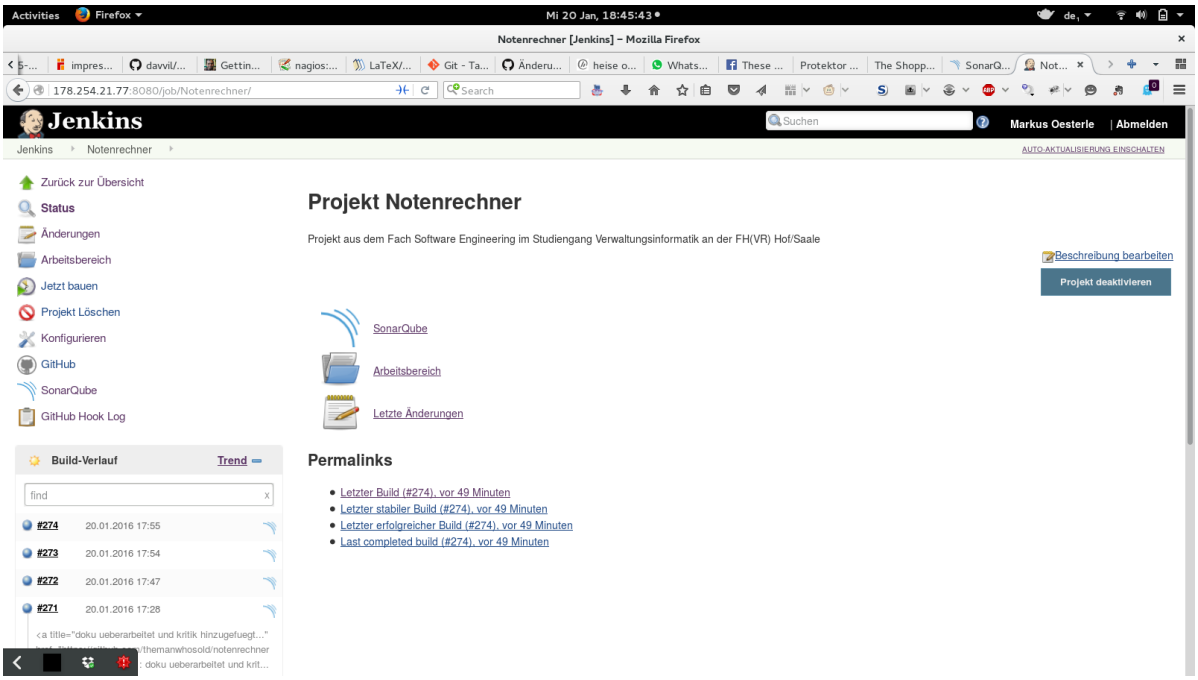

Abbildung 3.2: Jenkins Screenshot

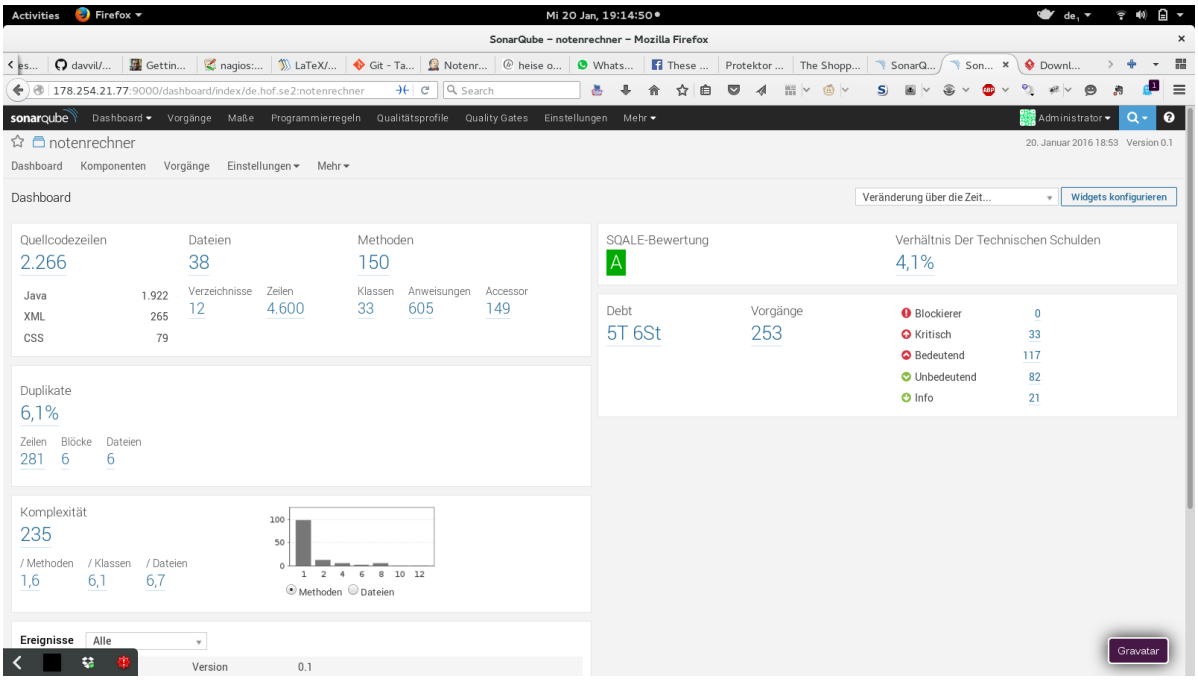

Abbildung 3.3: Sonarqube Screenshot

3.3 Betrieb

3.3.1 Wildfly

3.3.2 MySQL

3.4 Übersicht über die final verwendeten Versionen

Software	Version
Oracle Java SDK	8u60
Java EE	7
Netbeans	8.1
MySQL	
MySQL Workbench	6.3
Wildfly Application Server	9.0.1

4 Teamstruktur und Arbeitsverteilung

4.1 Gemeinsame Codeentwicklung mit GIT

Es wurde gegen ein gemeinsames Github-Repository entwickelt, von dem sich jeder im Team einen eigenen Fork erstellt hat. Entwickelter Code wurde per Pull Request an den Eigner des Hauptrepositories geschickt und von diesem nach erfolgreichen CI-Tests in den Hauptzweig gemerged. Diese Vorgehensweise hat sich nach einigen anfänglichen Schwierigkeiten als die sicherste herausgestellt, da Code, der zu Fehlern im Build-Prozess führt, vor dem Zusammenführen erkannt und nachgebessert werden kann und somit nicht der komplette Master-Branch unbrauchbar bzw. fehlerhaft wird.

4.2 Arbeitsverteilung

Es wurde angedacht das Projekt im Scrum Verfahren zu entwickeln. Jedoch wurde schnell klar, dass die Programmierfähigkeiten aller Teammitglieder und die Erfahrung mit Programmierobjekten nicht ausreichen um dies umzusetzen. Die im Kickoff-Meeting festgelegten Anforderungen wurden in einem Lasten- und Pflichtenheft festgehalten und dienten als Grundlage der Entwicklungsarbeiten. In den wöchentlichen Meetings wurden von den Teammitgliedern Themen ausgewählt und soweit möglich bis zum nächsten Meeting realisiert. Die Ergebnisse wurden vorgestellt und sofern noch Ergänzungsbedarf oder Fragen bestanden ergänzt oder zusammen mit einem anderen Mitglied weiterentwickelt.

Zur Abstimmung des Codes wurde von jedem Teamangehörigen ein Repository auf Git verwaltet, nach erledigter Programmiertätigkeit wurde der geänderte Code per Pull Request in das Master Repository übertragen, wo es nach einem kurzen Review gemerged wurde.

Nachdem im Gesamtprojekt eine gewisse Codequalität erreicht wurde, wurde der Code von einem Teammitglied, das nicht an der Entwicklung der jeweiligen Passagen beteiligt

war refactored und überarbeitet.

4.2.1 Protokolle der wöchentlichen Meetings

«««< HEAD Wöchentliche „Sprint“ Meetings auf denen die Fortschritte, aufgetretene Probleme und Lösungen besprochen werden und bei Bedarf neue Aufgaben verteilt werden. Leider ist die Protokollierung dieser Meetings nur sehr lückenhaft erfolgt. Die vorhandenen Protokolle finden sich aber in Folge. ======= Wöchentliche „Sprint“ Meetings auf denen die Fortschritte, aufgetretene Probleme und Lösungen besprochen werden und bei Bedarf neue Aufgaben verteilt werden. »»»> Sequenzdiagramme ergänzt

4.2.1.1 Kickoff Meeting

Protokoll siehe Punkt 2.1 Anforderungsliste

4.2.1.2 2tes Meeting

Beginn der Programmierarbeiten Verfeinerung der Anforderungen an die Maske für den Administrator: Maske für den Admin:

- Neuen Studiengang anlegen
- Wieviele Semester hat Studium
- Welche Semester = Grundstudium, Ausschlußprinzip alle anderen gehören zum Hauptstudium
- Stellenwert des Grundstudiums in der Endnote, Ausschlußprinzip -> Hauptstudium
- Studienfächer zum Grundstudium hinzufuegen
- zu vergebende Noten & Art der Noten festlegen
- Gewichtung festlegen (Noten untereinander & Leistungsnachweise zu normalen Noten)

- Art der Noten (FHVR Klausur, FH Klausur, FH Leistungsnachweis, FHVR Leistungsnachweis) vs. jede Note einzeln eintragen -> Art der Noten (extra Tabelle), jeweils für Grund- und Hauptstudium
  - FHVR Klausur
  - FH Klausur
  - FHVR Leistungsnachweis
  - FH Leistungsnachweis
  - FH Mündl. Prüfung
  - FHVR Mündl. Prüfung
  - FHVR Studienarbeit
  - Praxisbeurteilung
- Notentyp | Studienjahr | Gewichtung GS | Gewichtung HS

4.2.2 weiteres Protokoll

Bisheriger Arbeitsstand:

- es fehlen noch große Teile der Dokumentation
- Stefan erstellt Diagramme (Stefan)
- Lasten-/Pflichtenheft (Markus)
- Voraussetzung/Installationsanleitung (Markus)
- Ergänzung der Javadocs (Tobias)
- Administratorseite:
- Anlegen eines neuen Studenten in einem vorhandenen Studiengang (Max)
- Notenberechnung:

- Durchschnittsnoten der Leistungsnachweise sind nicht korrekt
- Datenmodell müsste feingranularer werden (8 verschiedene Gewichtungen) - mgl. Lösung zusätzliche Boolean Werte in der Notenliste
- Enum für Umrechnung von Notenpunkten in gesprochene Noten (Christoph)

5 (realisierte) Funktionalitäten

Hier sollen die Ablaufdiagramme hin Realisierung der Statistik-Funktionen: Dazu wird eine zentrale Java Enterprise Bean genutzt. Die Session-Bean StatistikBean übernimmt die Analyse der empirischen Daten. Die Bean stellt alle gängigen statistischen Kennzahlen zu Verfügung. Folgende statistische Maßzahlen werden berechnet: - Median - arithmetisches Mittel - Beste und schlechteste Note - Varianz - Standartabweichung

Die in StatistikBean implementierte Methode getStatistik(int idStudienfach) benötigt als Übergabeparameter die Id des zu analysierenden Studienfaches. Daraufhin werden alle Noten des jeweiligen Studiengangs aus der MySql Datenbank abgefragt. Diese Liste von Noten werden im folgenden analysiert und die oben gennanten statistischen Kennzahlen berechnet. Die Methode liefert ein Objekt der Klasse Statistik zurück, die die Werte der Berechnungen enthält.

Realisierung der Berechnung der End- bzw. Zwischennoten:

Die Logik zur Berechnung der Noten stellt die Java Enterprise Bean BerechnungNoten bereit. Die Berechnung findet für jeden Studenten individuell auf Basis der in der Datenbank persitierten Daten statt. Dazu ist die Methode getEndnote(int matrikelNr) in der Bean implementiert. Diese erwartet als Übergabeparameter den Primärschlüssel matrikelNr, um die Noten des jeweiligen Studenten aus der Datenbank abzufragen. Daraufhin werden die Noten zusammengerechnet und die Zwischen- und Endnote zu ermitteln. Zunächst wird eine Zwischennote (Note nach der Zwischenprüfung) berechnet. Daraufhin wird die Endnote ermittelt.

Die Methode getEndnote(int matrikelNr) liefert ein Objekt der Klasse Endnote zurück, die die Werte der Berechnungen enthält. Die Klasse Endnote enthält immer zwingend eine Zwischenprüfungsnote. Dadurch ist gewährleistet, dass auch die Zwischenprüfungsnote dem Studenten mitgeteilt werden kann.

5.1 Oberfläche für Studierende

5.1.1 Funktionsumfang

5.1.2 UML-Diagramm

5.2 Oberfläche für Dozenten

5.2.1 Funktionsumfang

5.3 Oberfläche für den Administrator

6 Einsatz der Software

6.1 Systemvoraussetzungen

6.2 Installation

7 Diagramme

Auf den folgenden Seiten finden sich einige UML Diagramme, diese finden sich in Originalgröße auf der

7.1 UML Diagramme

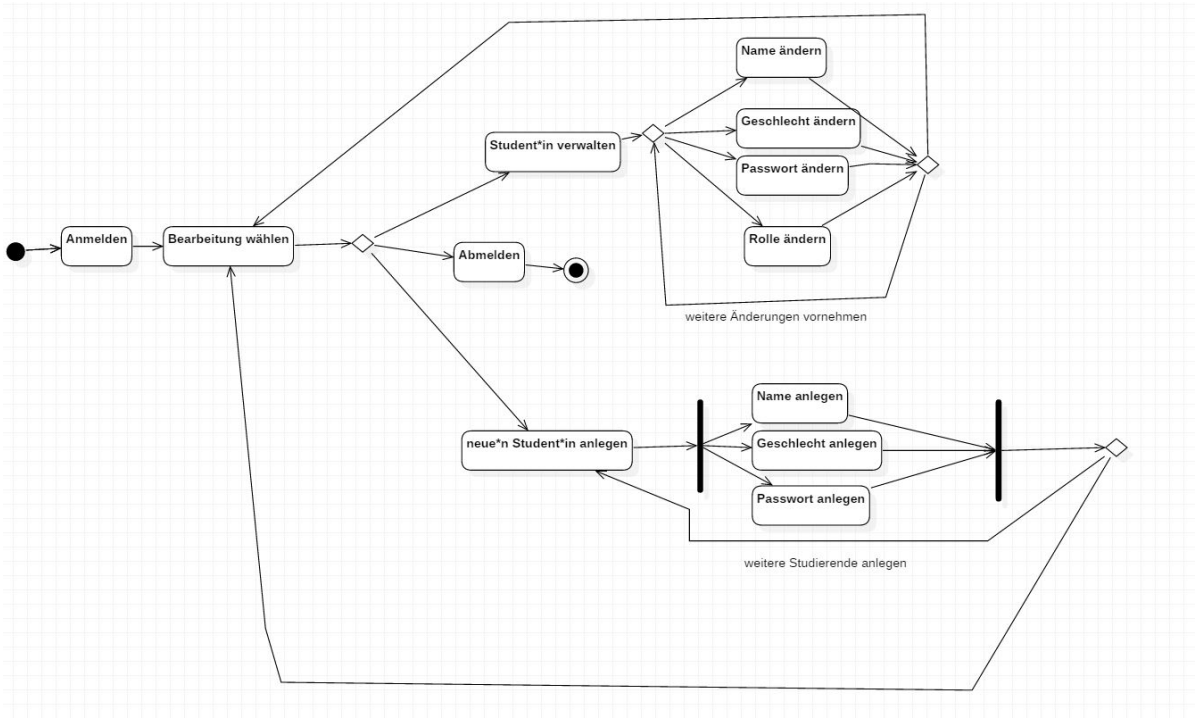

Abbildung 7.1: UML Diagramm Administrationsoberfläche

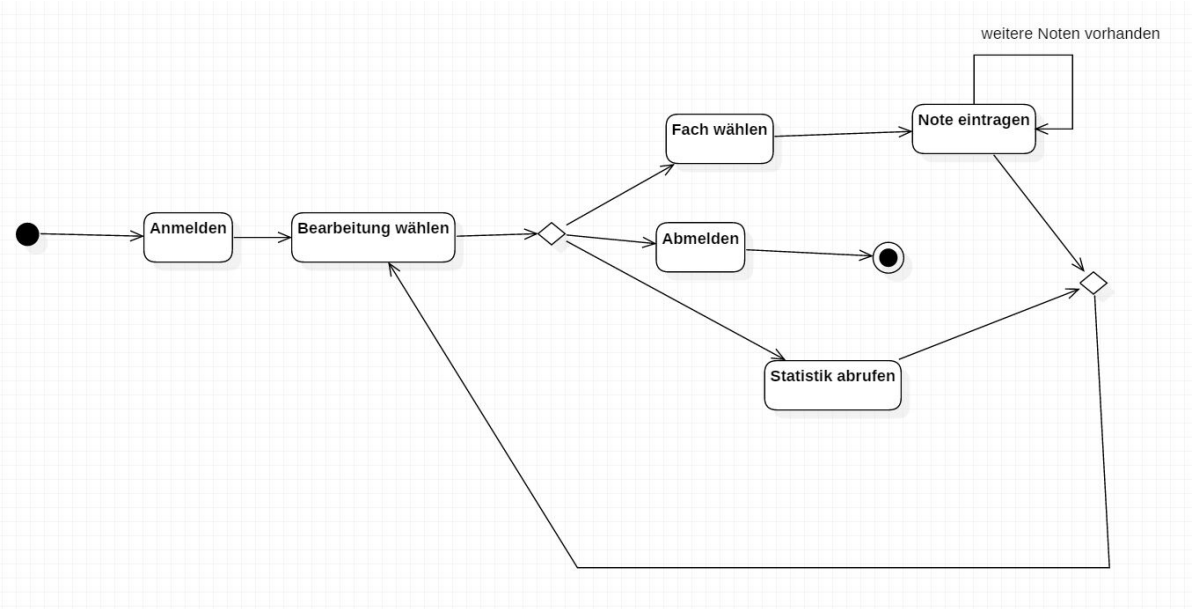

Abbildung 7.2: UML Diagramm Oberfläche Dozenten

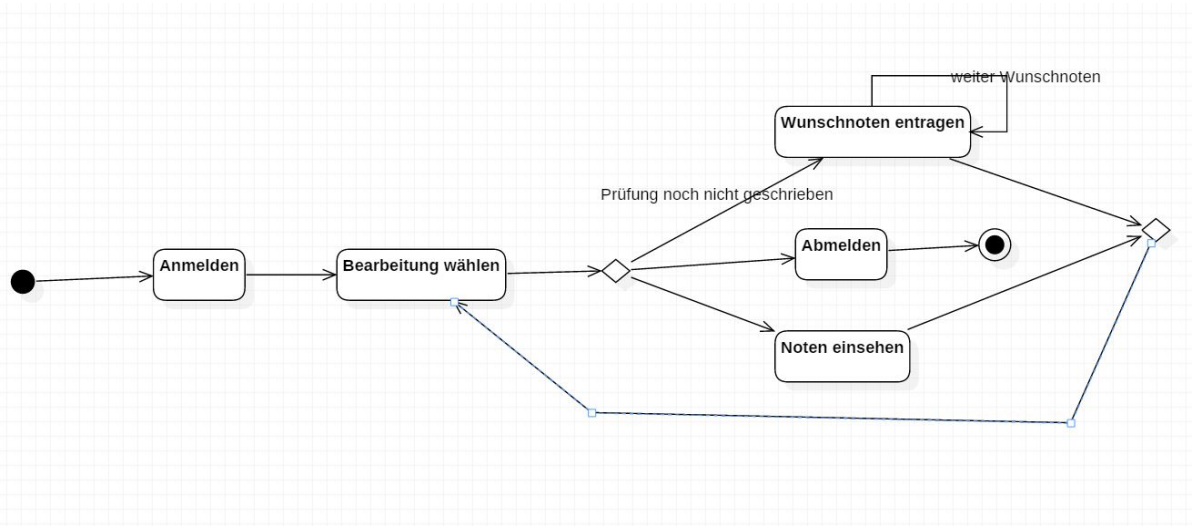

Abbildung 7.3: UML Diagramm Oberfläche Student

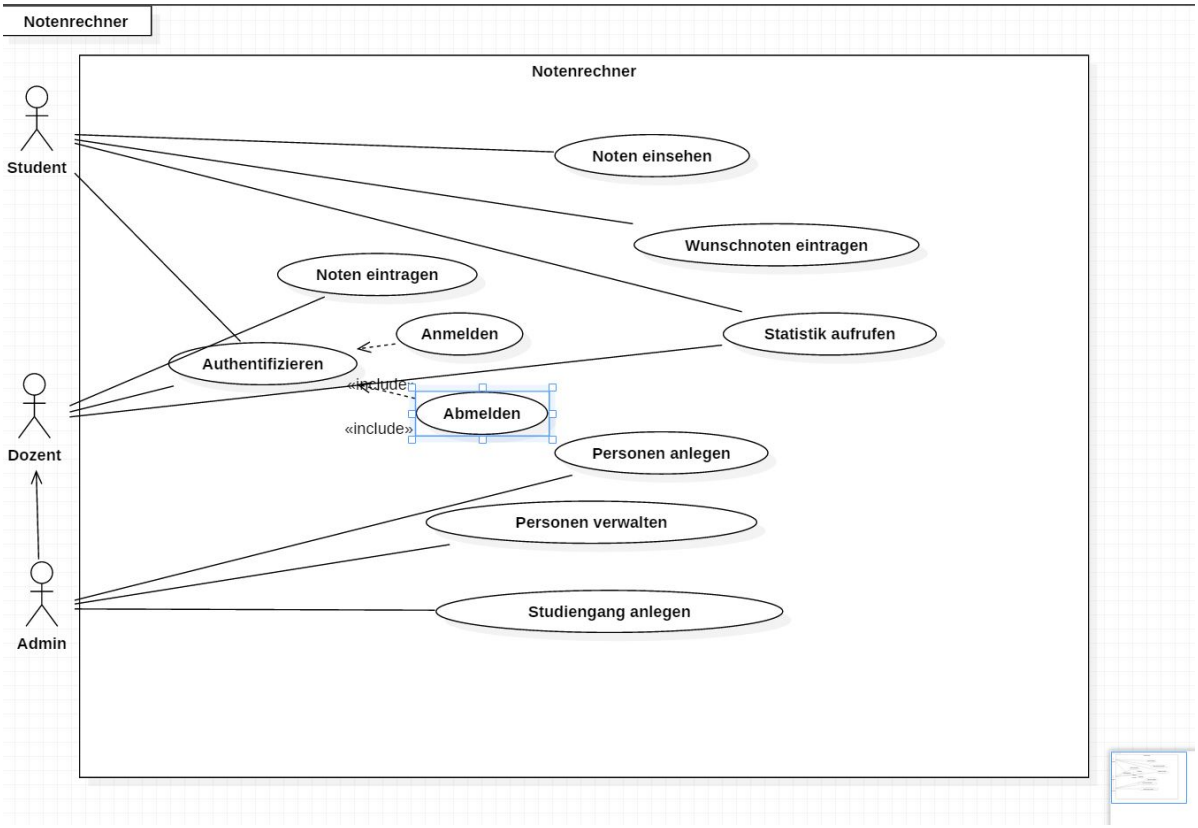

Abbildung 7.4: Use Case Diagramm

7.2 Use Case Diagramme

7.3 Klassendiagramme

7.4 Datenbank Entity Relationship Diagramm

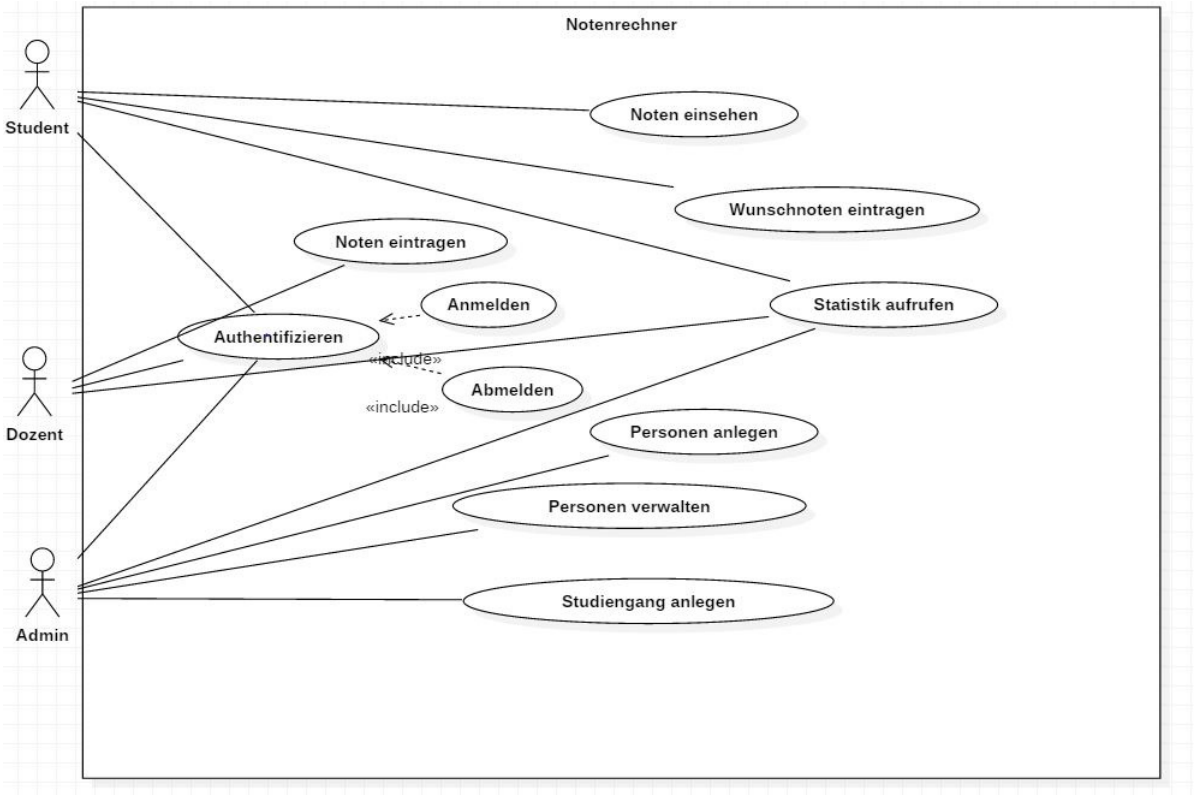

Abbildung 7.5: Use Case Diagramm 2

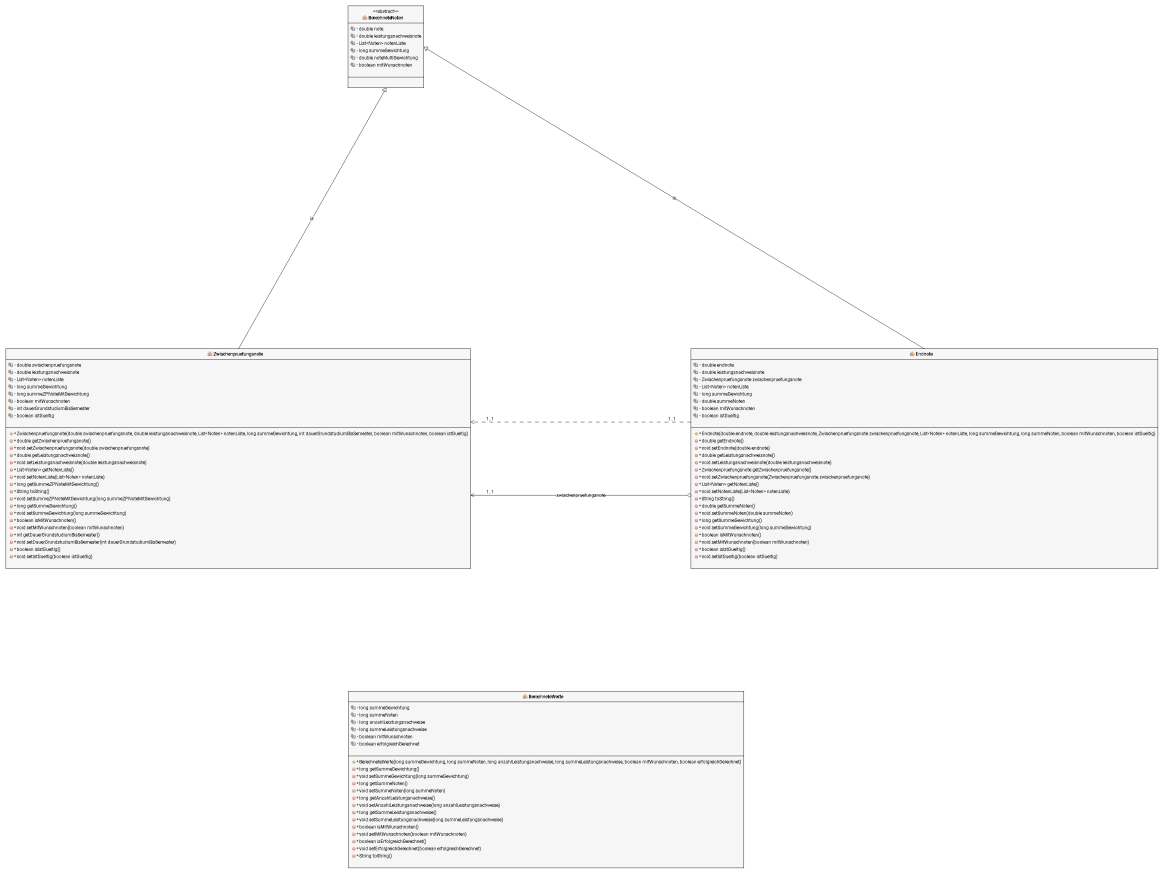

Abbildung 7.6: UML Klassendiagramm des Packages „Eigene Noten“

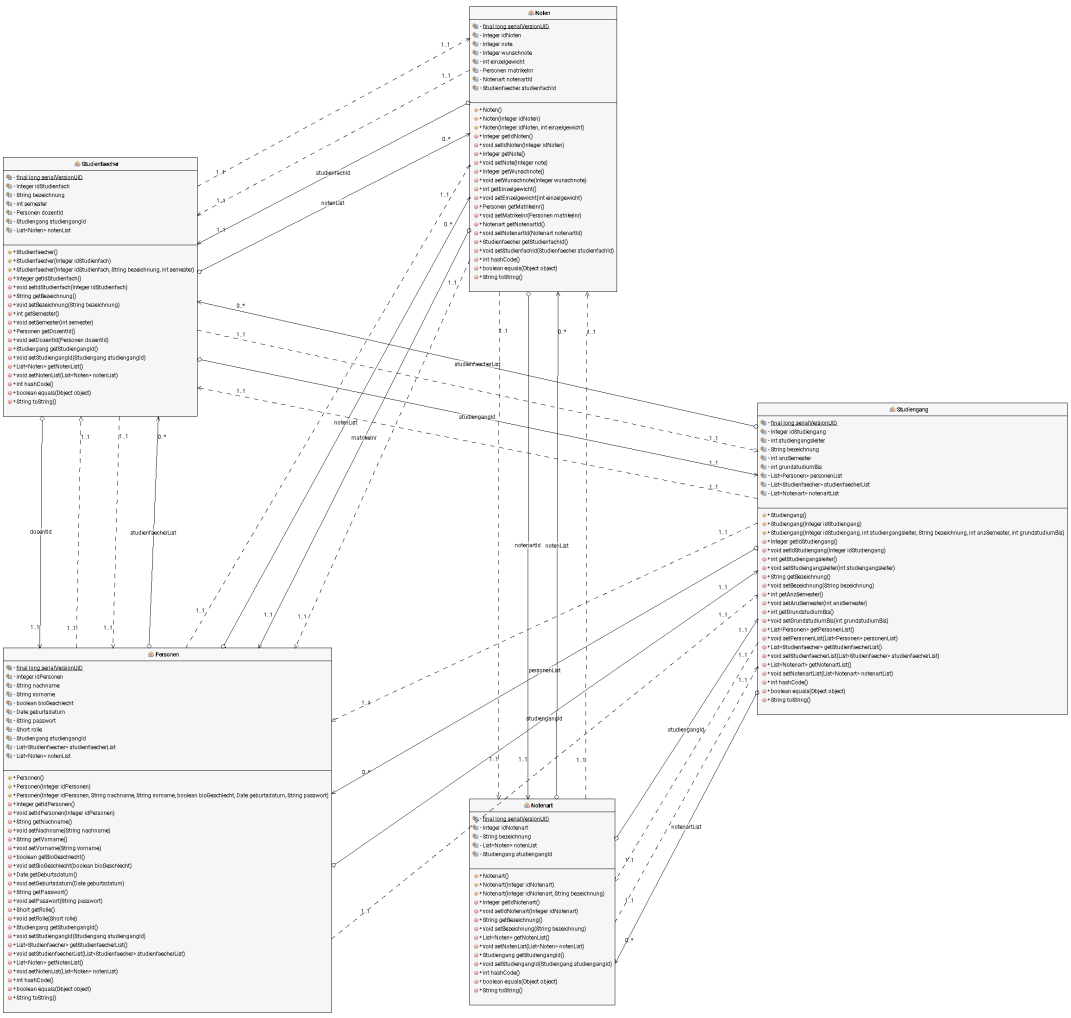

Abbildung 7.7: UML Klassendiagramm des Packages „Entity“

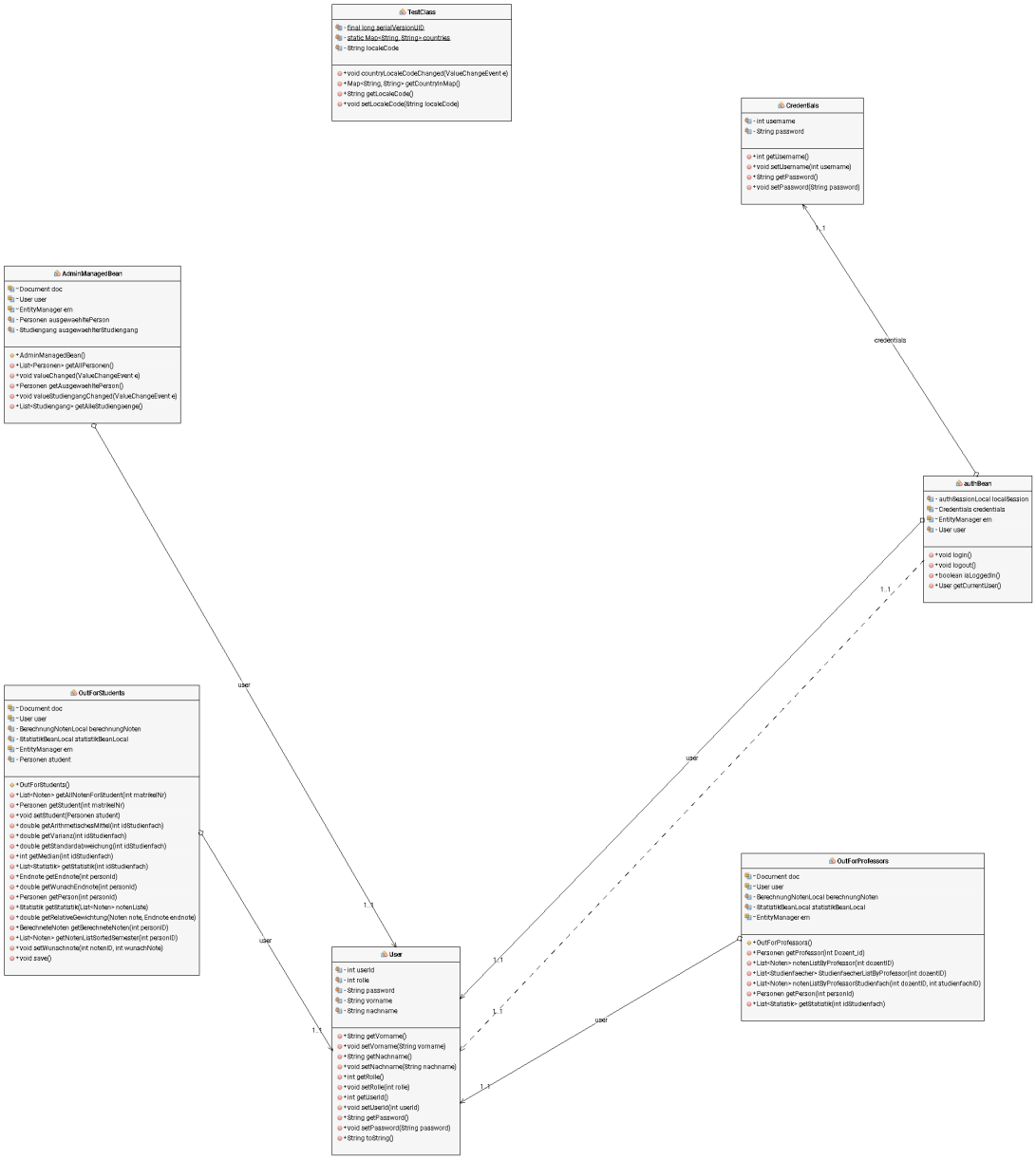

Abbildung 7.8: UML Klassendiagramm des Packages „Managed Bean“

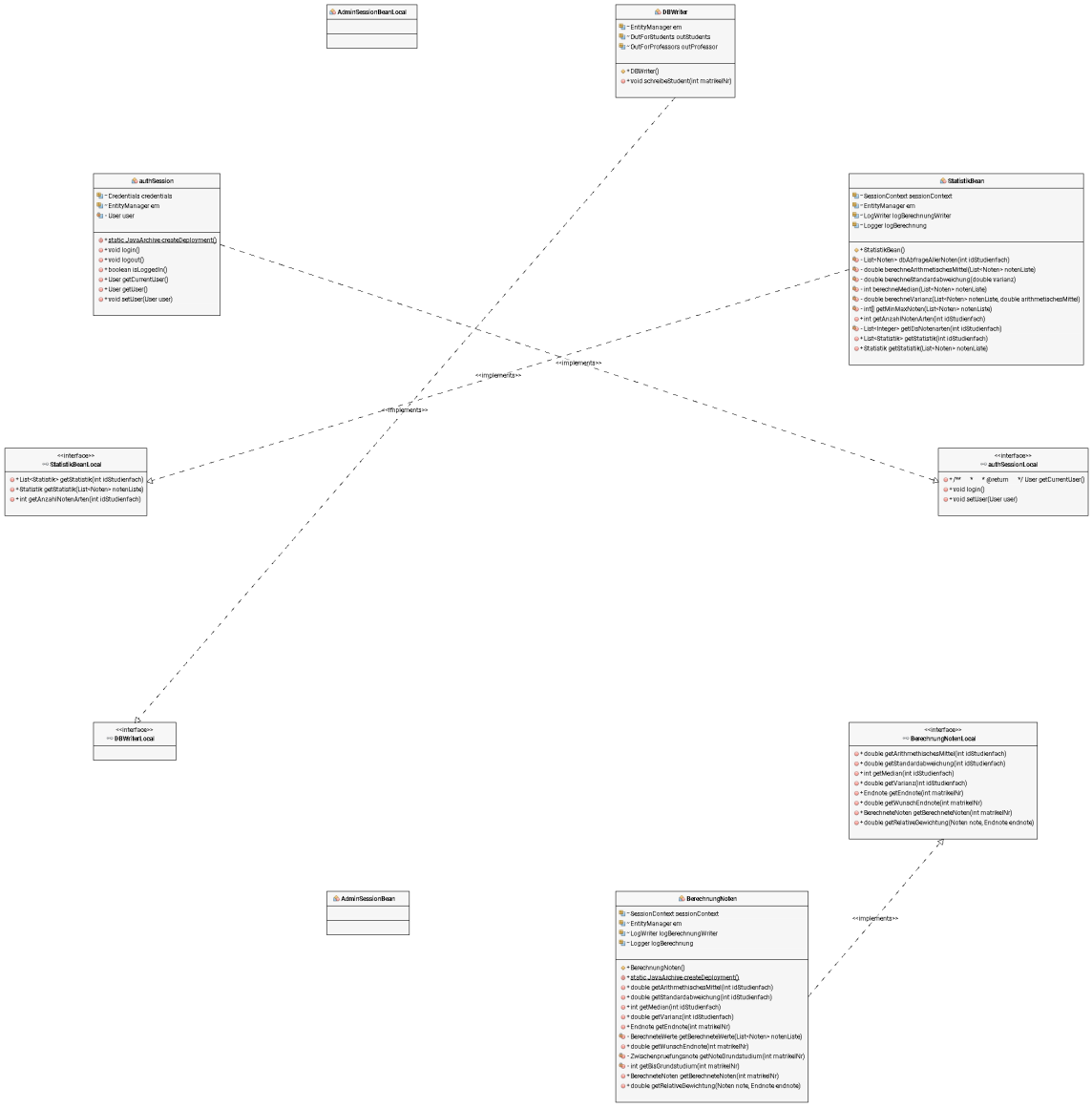

Abbildung 7.9: UML Klassendiagramm des Packages „Session Bean“

Abbildung 7.10: UML Klassendiagramm des Packages „Test“

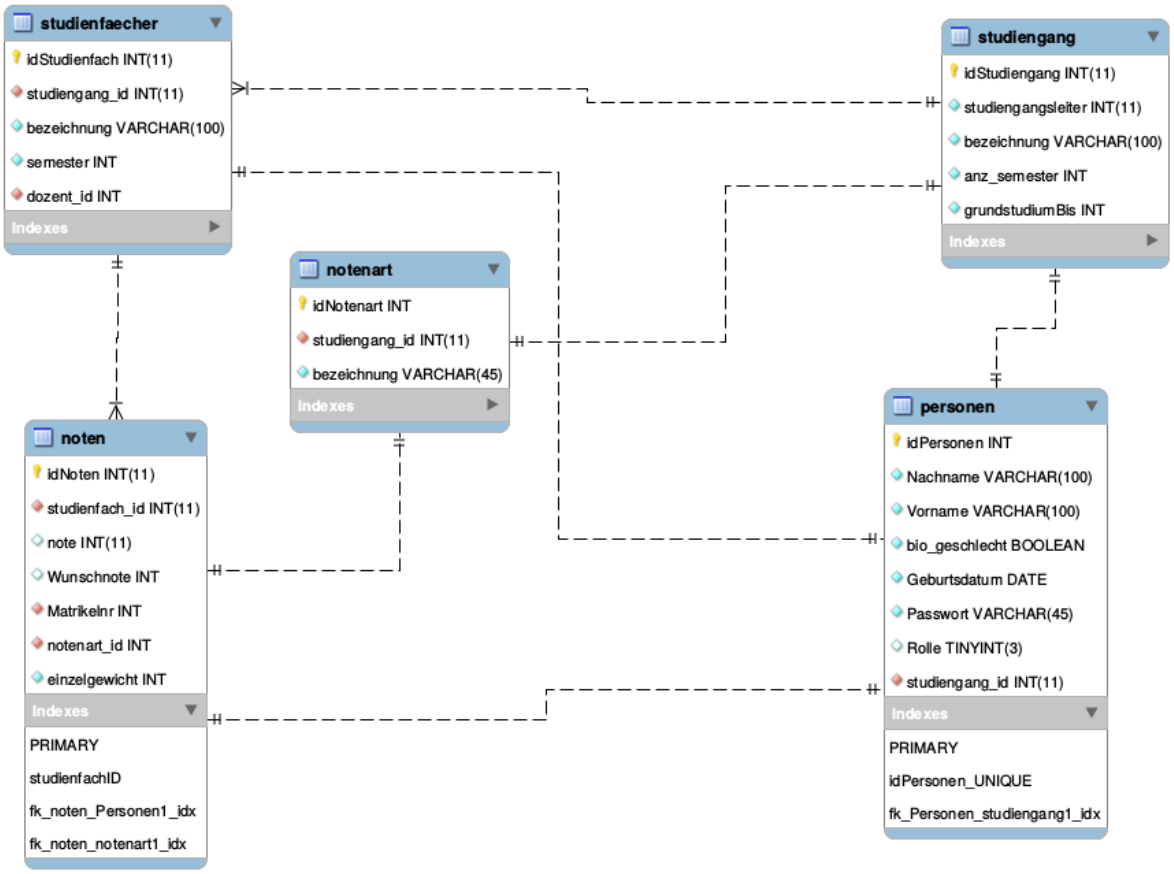

Abbildung 7.11: Entity Relationship Modell für die Datenbank

Abbildungsverzeichnis