Feedback System für das erfahrungsbasierte Erlernen von qualitativer Datenanalyse

BACHELOR THESIS

Lara Sulzbach

Eingereicht am 12. April 2024

Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik
Professur für Open Source Software

Betreuer:
Dr. Andreas Kaufmann
Prof. Dr. Dirk Riehle, M.B.A.

Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 12. April 2024

Lizenz

Erlangen, 12. April 2024

Abstract

One of the biggest criticisms of the Nailing Your Thesis (NYT) course, offered every year at Friedrich Alexander Universität (FAU) is the lack of feedback as evidenced by the course evaluation that students fill out at the end of the semester about the course. To meet this need, a feedback system is created as part of this bachelor thesis, which consists of automated feedback, peer review and instructor feedback. On the one hand the automated feedback serves to visualize the sources of error for the course participants and on the other hand it ensures transparency of the grading process, as participants receive a classification of their performance in the overall course and in the evaluation scheme specified by the course instructor. The functionalities of a peer review are implemented in a way that participants can evaluate other submissions and thus receive feedback from other participants and also gain insights into other proposed solutions. The third new feedback option is the instructor feedback, in which the course instructor evaluates individual submissions and can thereby provide single participants with individual feedback.

Zusammenfassung

Einer der größten Kritikpunkte am Kurs NYT, der jedes Jahr an der FAU angeboten wird, ist das fehlende Feedback, was eindeutig in den Evaluationsbögen der Studenten zu erkennen ist, die sie am Ende des Semesters über den Kurs ausfüllen. Um diesem Wunsch nachzukommen, wurde im Rahmen dieser Bachelorarbeit ein Feedbacksystem geschaffen, das aus einem automatisiertem Feedback, Peerreview und Betreuerfeedback besteht. Das automatisierte Feedback dient dabei zum einen der Visualisierung der Fehlerquellen für die Kursteilnehmer und zum anderen der Transparenz der Notengebung, indem die Teilnehmer für jede ihrer Abgaben eine Einordnung ihrer Leistung in die des gesamten Kurses und in das vom Kursleiter definierte Bewertungsschema bekommen. Die Funktionalitäten eines Peerreviews werden implementiert, damit die Teilnehmer andere Abgaben bewerten können und dadurch neben dem Feedback von weiteren Teilnehmern auch Einblick in unterschiedliche Lösungsvorschläge bekommen. Die dritte neue Feedbackoption ist ein Betreuerfeedback, bei dem der Kursleiter einzelne Abgaben bewertet und den Teilnehmern dadurch individuelles Feedback geben kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Acronyms

QDA Qualitative Datenanalyse

NYT Nailing Your Thesis

FAU Friedrich Alexander Universität

RP richtig positiven

FP falsch positiven

FN falsch negativen

1 Einleitung

„Es gibt absolut kein Feedback zu den Abgaben, wodurch man keinen Anreiz bzw es keine Möglichkeit gibt, sich zu verbessern“[1]. Dieses Zitat aus den Evaluationsbögen zum Kurs NYT, die am Semesterende von den Studenten ausgefüllt werden, beschreibt die aktuelle Problematik der Lehre von qualitativer Datenanalyse sehr gut. Der Kurs wird jedes Jahr an der FAU mit dem Ziel angeboten, Methodenkompetenz zum wissenschaftlichen Arbeiten zu vermitteln. Ein Teil des Kurses besteht aus dem Erlernen von QDA Techniken wie dem Kodieren von qualitativen Daten, bei dem theoretische Konstrukte in einem Text identifiziert und mit Codes markiert werden (Kaufmann et al., 2023). Dazu wird die Software QDAcity[2] verwendet, mit der die Studenten Kodieren in praktischen Hausaufgaben an einem Beispielprojekt üben können. Dieser Kurs stellt die Kursleiter in besonderem Maße vor die Herausforderung eines Zwiespalts zwischen Skalierbarkeit und Qualität, was zum einen ein Student im Evaluationsbogen feststellte: „Es sind viele Teilnehmer, die individuelle Kompetenzsteigerung leidet dadurch ein wenig“[3] und was zum anderen auch in der Fachliteratur ein bekanntes Thema ist. (DeLyser, 2008; Lowe, 1992) Das wissenschaftliche Arbeiten ist in allen Studiengängen wichtig, da spätestens für die Anfertigung der Bachelorarbeit diese Fähigkeit gebraucht wird. Deshalb haben sehr viele Studenten Interesse am Kurs NYT und die Anzahl der Kursteilnehmer steigt jedes Jahr auf bis zu 120 Teilnehmer. Denen möchte man gerecht werden und will sie auch nicht durch eine Maximalanzahl beschränken, um die für das wissenschaftliche Arbeiten zentralen Lerninhalte einem möglichst breiten Publikum zugänglich zu machen. Ein Maß für die Qualität des Kurses ist der Fähigkeitserwerb der Studenten, der wiederum stark mit der Qualität des Feedbacks für die einzelnen Studenten korreliert, aus dem sie lernen und sich verbessern können. Doch welches Feedback hatten die Studenten der oben zitierten Evaluation erhalten? Für die Aufgaben, bei denen Studenten Texte kodieren müssen, war bisher die einzige Rückmeldung ihr F-Maß,

[1] https://oss.cs.fau.de/wp-content/uploads/2021/02/WS20_21_Riehle_20w-OSS-NYT-VUE.pdf

[2] https://qdacity.com/de/

[3] https://oss.cs.fau.de/wp-content/uploads/2020/12/SS_19_Riehle_19s-OSS-NYT-VUE.pdf

Genauigkeit und Trefferquote (wird im Kapitel 1.2 näher erläutert) ihrer Abgaben und die Durchschnittswerte des Kurses im Vergleich. Gleichzeitig erhielten sie als Teil der NYT-Unterlagen ein Bewertungsschema, auf dessen Grundlage sie sich mit dem von ihnen erreichten F-Maß einordnen konnten. Dies half den Studenten kaum, aus ihren Fehlern zu lernen, da sie keine präzisen Hinweise darauf bekamen, wo ihre Fehler genau lagen und welche Punkte sie hätten besser umsetzen können. Somit war es schwierig für sie, ihre Abgaben zu reflektieren und dadurch von einem erfahrungsbasierten Lernen zu profitieren (Kolb, 1984), obwohl dieses Lehrkonzept ein selbstgesteuertes und lebenslanges Lernen fördert. (Jiusto & DiBiasio, 2006) Für solch ein wünschenswertes, individuell auf die Fehler des einzelnen eingehendes Feedback wäre jedoch ein massiver personeller Aufwand nötig, den die Betreuer des Kurses nicht leisten können. Um trotzdem die Skalierbarkeit des Kurses sicherzustellen, muss daher ein Feedbacksystem etabliert werden, das auch bei einer großen Anzahl von Kursteilnehmern sicherstellt, dass jeder ein individuelles Feedback bekommt. Im Rahmen dieser Bachelorarbeit ist ein System in QDAcity eingeführt worden, welches das Feedback für die Kodieraufgaben im Rahmen der QDA Lehre verbessert. Das System besteht dazu aus drei Teilen: Zum einen dem automatisiertem Feedback, woraus die Studenten die Schwächen ihrer Abgabe entnehmen können, und zum anderen einem Peerfeedback, bei denen sich die Studenten gegenseitig beurteilen und dadurch sowohl Einblick in andere Abgaben als auch ein zusätzliches individuelles Feedback bekommen. Als dritte Feedbackmöglichkeit wurde ein Betreuerfeedback implementiert, bei dem der Kursleiter einzelne Abgaben bewerten kann.

1.1 QDA in der Lehre mit der Webanwendung QDAcity

Das Feedbacksystem soll in die Webanwendung QDAcity integriert werden, die den Prozess von QDA unterstützt und zusätzlich einen eigenen Kursbereich zur Förderung der Lehre beinhaltet.[4] Als registrierter Nutzer kann man einen Kurs mit zugehöriger Semesterbezeichnung anlegen und seine Kursteilnehmer dazu einladen, diesem Kurs beizutreten. Innerhalb des Kursbereichs kann der Kursleiter dann Aufgaben erstellen. Diese basieren auf einer Referenzlösung, die er zunächst anlegen muss. Dafür erstellt er zunächst ein Codebuch für einen selbst hochgeladenen Text. Die Struktur eines Codebuchs besteht dabei nach MacQueen et al. (1998) aus den folgenden Bestandteilen: dem Code, einer kurze Erklärung, einer vollständige Erklärung, Richtlinien, wann und wann nicht man den Code nutzen sollte und aus Beispielen. Im Anschluss kodiert er dann mit den im Codebuch definierten Codes diesen Text und erstellt davon eine Revision. Zusätzlich zum Auswählen der Referenzlösung kann der Kursleiter bei der Aufgabenerstel-

[4] https://qdacity.com/de/qda-lehre/

lung auch den Aufgabentyp auswählen. QDAcity unterstützt zwei Aufgabentypen. Zum einen *Codebuch Verwendung*, bei dem die Kursteilnehmer den Text der Referenzlösung mit dem vom Kursleiter erstelltem Codebuch kodieren. Das hat den Vorteil, dass man die Abgabe der Teilnehmer gut mit der Lösung vergleichen kann und mithilfe des *Intercoder Agreement* (siehe Kapitel 1.2) bewerten kann. Zum anderen gibt es auch den Aufgabentyp *Codebuch Erstellung*. In dieser Variante erstellt der Kursteilnehmer ein eigenes Codebuch mit dem er dann den Text der Referenzlösung kodiert. Aufgrund der dadurch entstehenden individuellen Codes kann man die Abgabe jedoch nicht mehr automatisiert mit der Lösung vergleichen, sodass hier ein individuelles Feedback nötig ist. Nachdem sie dem Kurs und Semester beigetreten sind, können die Kursteilnehmer selbständig die Aufgabe lösen. Dabei ist es ihre Aufgabe, den Text zu analysieren und mit den jeweiligen Codes bestimmte Textteile zu markieren. Wenn die Zeit der Aufgabe abgelaufen ist, kann der Kursleiter beim Aufgabentyp *Codebuch Verwendung* einen Report erstellen, der ihm anzeigt, wie die Teilnehmer das Projekt im Vergleich zu seiner eigenen Lösung kodiert haben.

1.2 Intercoder Agreement

Um Kodierungen zu vergleichen, gibt es in der QDA das Intercoder Agreement. Darunter versteht man eine numerische Messmethode, mit der unterschiedliche Kodierungen verglichen werden können (Cliodhna & Helene, 2020). Dabei kodieren mehrere Personen die gleichen Daten und anschließend wird geprüft, inwieweit die Kodierungen übereinstimmen. Die gesetzten Codes werden immer in einer bestimmten Längeneinheit, wie Zeile, Satz oder Absatz, verglichen. Dies wird auch als *unit of coding* bezeichnet. Diese Längeneinheiten sollen die Gewohnheiten unterschiedlicher Menschen ausgleichen. Manche Personen tendieren eher dazu, einzelne Zeilen einem Kontext zuzuordnen, andere betrachten ganze Absätze (Campbell et al., 2013). Im Aufgabenbereich von QDAcity wird das Intercoder Agreement zwischen der Referenzlösung und einer einzelnen Abgabe berechnet. Dafür wird mit einer Standard-Einstellung jeweils absatzweise die Kodierung verglichen und die Übereinstimmung mit dem unten erläuterten F-Maß berechnet. Das Ergebnis wird für die Bewertung der Abgabe verwendet. Für die Berechnung werden zunächst die richtig positiven (RP) Codes gezählt, also die Codes, die sowohl in der Abgabe, als auch in der Musterlösung in der gewählten Längeneinheit, vorkommen. Zusätzlich werden auch die falsch positiven (FP), also Codes, die in dem Abschnitt verwendet wurden, aber nicht in der Musterlösung vorkommen und auch die falsch negativen (FN), die zwar in der Musterlösung vorkommen, aber in der Abgabe nicht verwendet wurden, gezählt. Mithilfe dieser Zahlen, kann für jeden Absatz die Trefferquote und die Genauigkeit bestimmt werden, woraus man dann die Übereinstimmung mithilfe des F-Maßes berechnet. (Harrison, 2019)

$$Trefferquote = \frac{RP}{RP + FN} \tag{1.1}$$

$$Genauigkeit = \frac{RP}{RP + FP} \tag{1.2}$$

$$F - Ma\beta = \frac{2 \cdot Trefferquote \cdot Genauigkeit}{Trefferquote + Genauigkeit} \tag{1.3}$$

Diese Messmethode belohnt nicht nur das korrekte Setzen von Codings durch eine hohe Trefferquote, sondern bestraft auch die Verwendung von Codes, die in der Referenzlösung nicht vorkommen, durch einen niedrigen Genauigkeitswert. (Kaufmann et al., 2016)
Daher eignet sich das F-Maß gut für die Einordnung der Leistung einer Abgabe in ein Bewertungsschema. Wenn die Grenzwerte dieses Bewertungsschemas auf die konkrete Aufgabe eingestellt sind, zeigt hat das Paper von Kaufmann et al. (2023), dass die Bewertung stark mit einer manuellen Bewertung korreliert.

2 Anforderungen

Um das Ziel der Bachelorarbeit, den Kursteilnehmern ein besseres Feedback zu ihren Abgaben zu geben, zu erreichen, sind neue Funktionalitäten in der Anwendung gefordert, die im Rahmen der Arbeit in QDAcity implementiert werden sollten. Diese werden im Unterkapitel Funktionale Anforderungen spezifiziert. Gleichzeitig soll eine hohe Qualität der Software sichergestellt werden, die im Kapitel 2.2 thematisiert wird.

2.1 Funktionale Anforderungen

Die funktionalen Anforderungen lassen sich in die vier Kategorien Abgabe einer Aufgabe, automatisiertes Feedback, Peerreview und Feedback des Kursleiters aufteilen. Für eine einheitliche Repräsentation der Anforderungen wird die Vorlage der SOPHISTen (2016) mit dem folgenden Aufbau verwendet.

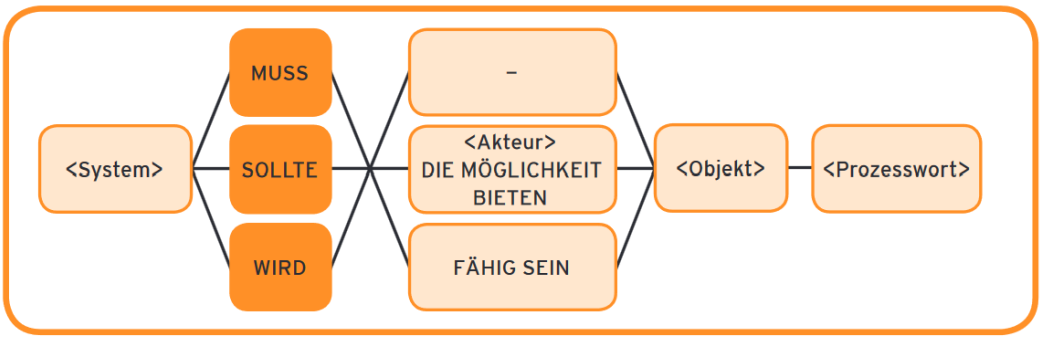

Abbildung 2.1: Vorlage für den Aufbau funktionaler Anforderungen

Diese Vorlage verwendet die drei Schlüsselwörter *muss*, *sollte* und *wird*, um die Wichtigkeit der Anforderung auszudrücken. Eine mit *muss* formulierte Anforderung, muss verpflichtend umgesetzt werden. Wenn *sollte* verwendet wird, dann handelt es sich um den Wunsch des Stakeholders, der nicht erfüllt werden muss. *Wird* drückt eine Absicht des Stakeholders aus. Diese muss verpflichtend um-

gesetzt werden, da sie die Vorbereitung für eine zukünftige Funktion ist. Die Erfüllung der Anforderung wird aber zunächst nicht getestet.

2.1.1 Abgabe einer Hausaufgabe

Die Kursteilnehmer sollen durch eine neue Abgabefunktion beeinflussen können, welcher Stand ihres Projektes bewertet werden soll und es soll ihnen die Möglichkeit gegeben werden, sich diesen Stand auch nach Abgabe und Bewertung der Aufgabe erneut anschauen zu können. Diese Funktion dient als Grundlage für sowohl das automatisierte Feedback als auch das manuelle Feedback durch Peers und Kursleiter. Folgende funktionale Anforderungen sollen erfüllt werden:
FA 1.1: Der Aufgabenbereich muss dem Kursteilnehmer die Möglichkeit geben, die Codings und bei Aufgabentypen mit Codebuch Erstellung auch das Codebuch abzugeben.
FA 1.2: Der Aufgabenbereich sollte dem Kursteilnehmer die Möglichkeit geben, sich seine Abgaben jederzeit im *Coding Editor* anzuschauen.

2.1.2 Automatisiertes Feedback

Verdeutlichung der Bewertung der Leistung innerhalb der eigenen Abgabe

Dem Nutzer soll die Bewertung der eigenen Leistung und insbesondere die Problemstellen im abgegebenen Projekt grafisch visualisiert werden. Dieses Ziel soll mit zwei verschiedenen Herangehensweisen sichergestellt werden. Zum einen sollen die Kursteilnehmer erkennen können, welche Absätze gut und welche schlechter kodiert wurden. Dazu sollen sogenannte *Agreement Maps* benutzt werden, die absatzweise berechnen wie die Abgabe im Vergleich zur Musterlösung kodiert wurde. Dabei kann man zwischen der *Agreement Map* für RP, die die Anzahl an Codes die sowohl in der eigenen Abgabe, als auch in der Musterlösung vorkommen, zählt, für FP, also Codes, die in der Abgabe verwendet wurden, aber nicht in der Musterlösung vorkommen und für FN, die zwar in der Musterlösung vorkommen, aber in der Abgabe nicht verwendet wurden, unterscheiden.
Zum anderen sollen die Kursteilnehmer auch erkennen können, welche Codes Probleme bereitet haben. Dies wird mit einer tabellarischen Übersicht gelöst. Im Detail ergeben sich daraus folgende Einzelanforderungen:
FA 2.1.1 Das automatisierte Feedback sollte aus der letzten Abgabe des Kursteilnehmers erstellt werden. Falls jedoch keine Abgabe getätigt worden ist, soll eine Abgabe vom aktuellen Bearbeitungsstand automatisiert erstellt werden.
FA 2.1.2: Die *Agreement Maps* müssen den Grad an Abweichung zur Musterlösung visualisieren.

FA 2.1.3: Die Einfärbungen der *Agreement Maps* sollte dem Kursteilnehmer die Möglichkeit geben, die Absätze seiner Abgaben mit den größten Abweichungen herauszufinden und die Einfärbungen sollen relativ zur Leistung innerhalb der eigenen Abgabe sein.
FA 2.1.4: Die Nutzeroberfläche sollte dem Kursteilnehmer eine Hilfestellung zur Terminologie bieten.
FA 2.1.5: Die *Agreement Maps* sollten dem Kursteilnehmer die Möglichkeit geben, mit einer Mehrfachauswahl die RP, FP und die FN nebeneinander zu sehen.
FA 2.1.6: Eine Übersichtsseite muss dem Kursteilnehmer die Möglichkeit geben, einen Überblick zu bekommen, welche Codes in der Abgabe abweichend von der Musterlösung verwendet wurden. Dazu soll es einen tabellarischen Überblick mit den F-Maß der einzelnen Codes über alle kodierten Dokumente der Abgabe geben.

Verdeutlichung der Bewertung der Leistung im Vergleich zu den anderen Kursteilnehmern

Das Ziel der Anforderungen in diesem Abschnitt ist es, dass die Kursteilnehmer ihre Leistung im Vergleich zu anderen Teilnehmern desselben Kurses beurteilen können. Dies soll mit zwei neuen Funktionen sichergestellt werden. Zum einen wird berechnet, in welchem Perzentil aller Leistungen die eigene Abgabe liegt und zum anderen wird vom Kursleiter ein Bewertungsschema angelegt, in welches die Abgabe eingeordnet wird.
FA 2.2.1 Die Übersichtsseite der Aufgabenbewertung sollte dem Kursteilnehmer die Möglichkeit geben, zu sehen, in welchem Quantil innerhalb des Kurses seine Abgabe war. Dabei soll die Auswertung auf Zehntel genau ausgegeben werden. Das Ergebnis sollte dem Kursteilnehmer grafisch durch eine Anzeige visualisiert werden.
FA 2.2.2 Der Dialog zum Erstellen einer neuen Aufgabe muss dem Kursleiter die Möglichkeit geben, ein Bewertungsschema festzulegen. Dazu legt er die Grenzen des F-Maßes für eine minimale, durchschnittliche und außergewöhnliche Leistung fest.
FA 2.2.3 Die Felder zum Anlegen eines Bewertungsschemas sollten durch ein Info Icon mit aufklappbaren Informationen erklärt werden.
FA 2.2.4 Die Übersichtsseite der Aufgabenbewertung muss dem Kursteilnehmer die Möglichkeit geben, grafisch mit einer Anzeige die Einordnung in das Bewertungsschema nachzuvollziehen.

2.1.3 Peerreview

Das Ziel des Peerreviews ist es, dem Kursteilnehmer zusätzliches Feedback zu geben und auch die Möglichkeit, Einblicke in andere Abgaben zu bekommen und

sich mit deren Lösungsvorschlägen auseinander zu setzen.
FA 3.1 Der Dialog zum Erstellen einer neuen Aufgabe muss dem Kursleiter die Möglichkeit geben, anzugeben, ob ein Peerreview durchgeführt werden soll und wenn ja wie viele Reviews mindestens pro Teilnehmer durchgeführt werden müssen.
FA 3.2 Die Übersichtsseite der Aufgabenbewertung muss dem Kursteilnehmer die Möglichkeit geben, eine fremde Abgabe zum Reviewen anzufordern.
FA 3.3 Die Reviewseite sollte dem Reviewer die Möglichkeit geben, mit Likert-Skalen, offenen Fragen und einer Gesamtbewertungsskala die zu beurteilende Abgabe zu bewerten.
FA 3.4 Die Reviewseite sollte dem Reviewer die Möglichkeit geben, parallel zum Review die Abgabe im *Coding Editor* einsehen zu können.
FA 3.5 Die Reviewseite sollte dem Reviewer die Möglichkeit geben, das Review zwischenzuspeichern und später fortzuführen. Solange ein Review nicht final abgegeben worden ist, ist es für den Autor der Abgabe nicht einsichtig. Gleichzeitig kann der Reviewer auch noch kein neues Review anfangen.
FA 3.6 Die Reviewseite sollte dem Reviewer die Möglichkeit geben, das Review abzugeben und dies durch einen zusätzlichen Dialog zu bestätigen.
FA 3.7 Die Übersichtsseite der Aufgabenbewertung muss dem Reviewer die Möglichkeit geben, die von ihm gegebenen Urteile zu seinen Reviews anzuschauen. Dabei muss aber ein Bearbeiten nach Abgabe verhindert werden und der User darüber aufgeklärt werden, dass das Review bereits abgegeben worden ist.
FA 3.8 Die Übersichtsseite der Aufgabenbewertung muss dem Kursteilnehmer die Möglichkeit geben, Reviews zu seinen Abgaben anzuschauen.
FA 3.9 Die Reviewseite sollte dem Kursteilnehmer, der das Review erhalten hat, die Möglichkeit geben, das erhaltene Review zu bewerten.

2.1.4 Feedback Kursleiter

Der Kursleiter soll einzelne Abgaben gezielt bewerten können.
FA 4.1 Der Dialog zum Erstellen einer neuen Aufgabe muss dem Kursersteller die Möglichkeit geben, ein Bewertungsschema für das Betreuerfeedback festzulegen.
FA 4.2 Die Reportseite sollte dem Kursleiter die Möglichkeit geben, Feedback zu einzelnen Abgaben zu geben.
FA 4.3 Die Reportseite sollte dem Kursleiter die Möglichkeit geben, die Standardabweichung der Gesamtbewertung erhaltener Peerreviews von einzelnen Abgaben zu sehen.
FA 4.4 Die Übersichtsseite der Aufgabenbewertung sollte den Kursteilnehmer die Möglichkeit geben, das Feedback des Kursleiters einzusehen.

2.2 Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen orientieren sich am ISO Standard für Software Qualität 25010[1]. Demnach werden die Anforderungen nach den 8 Themen Stabilität, Performanz, Kompatibilität, Benutzbarkeit, Zuverlässigkeit, Sicherheit, Wartbarkeit und Portabilität unterteilt. Diese Unterteilung wird auch in dieser Bachelorarbeit verwendet. Die Anforderungen folgen, wie auch schon die funktionalen, der Vorlage der SOPHISTen (2016):

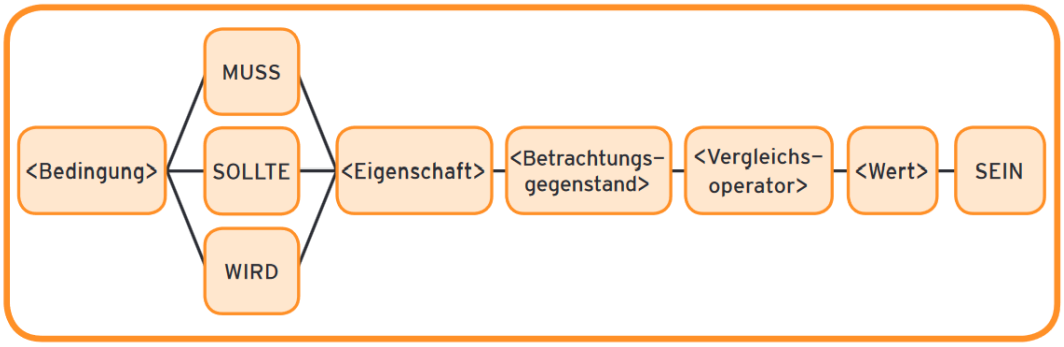

Abbildung 2.2: Vorlage für den Aufbau nicht funktionaler Anforderungen

2.2.1 Stabilität

NFA 1.1 Die Implementierung darf keine vorhandenen Funktionalitäten beeinträchtigen.
NFA 1.2 Der Rollout einer neuen Funktion sollte über ein Feature Flag für einen Nutzer verborgen bleiben, bis die Funktionalität angemessen umgesetzt ist.

2.2.2 Performanz

NFA 2.1 Längere und aufwändigere Rechenaufwände sollen in das Backend verlagert werden, um die Performance im Frontend zu beschleunigen.

2.2.3 Kompatibilität

NFA 3.1 Die Implementierung muss kompatibel mit den bereits verwendeten Technologien sein.

[1] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

2.2.4 Benutzbarkeit

NFA 4.1 Die Texte in der Benutzeroberfläche müssen alle eine deutsche Übersetzung haben.
NFA 4.2 Das Design muss das vorhandene Farbschema von QDAcity übernehmen.
NFA 4.3 Das System muss sicherstellen, dass der Reviewer das Review ausgefüllt hat, bevor er es abgeben kann.
NFA 4.4 Die Benutzeroberfläche muss den Web Content Accessibility Guidelines (WCAG) 2.1 entsprechen.
NFA 4.5 Das System muss sicherstellen, dass das eingegebene Bewertungsschema geeignet ist.

2.2.5 Sicherheit

NFA 5.1 Alle Endpunkte müssen die Autorisierung prüfen, bevor sie die Anfrage durchführen können.
NFA 5.2 Das System muss sicherstellen, dass der Reviewer nicht die Identität des Autors der Abgabe herausfinden kann.
NFA 5.3 Das System muss sicherstellen, dass der Kursteilnehmer nicht die Identität des Reviewers herausfinden kann.
NFA 5.4 Das System muss sicherstellen, dass der Kursteilnehmer nicht die Musterlösung rekonstruieren kann.

2.2.6 Wartbarkeit

NFA 6.1 Alle neuen Endpunkte müssen eine Testabdeckung durch JUnittests von mindestens 90% haben.
NFA 6.2 Alle neuen Funktionen müssen mit Akzeptanztests geprüft werden.
NFA 6.3 Das System muss für neue Feedbackarten erweiterbar sein.
NFA 6.4 Das System muss sicherstellen, dass unterschiedliche Zuständigkeiten auch in getrennten Klassen implementiert werden.

3 Architektur

Dieses Kapitel beschreibt zunächst die Ausgangslage der Architektur im Bereich der Übungsaufgaben in Abschnitt 3.1 und erläutert dann in den Abschnitten 3.2 bis 3.10 die erforderlichen Anpassungen, um die Erfüllung der Anforderungen aus Kapitel 3 zu ermöglichen

3.1 Ausgangslage

Da sich die Anforderungen auf dem Übungsbereich von QDAcity beschränken, wird in diesem Kapitel auch nur die Architektur aus diesem Bereich betrachtet. In Abbildung 3.1 sind die relevanten Klassen vereinfacht dargestellt.

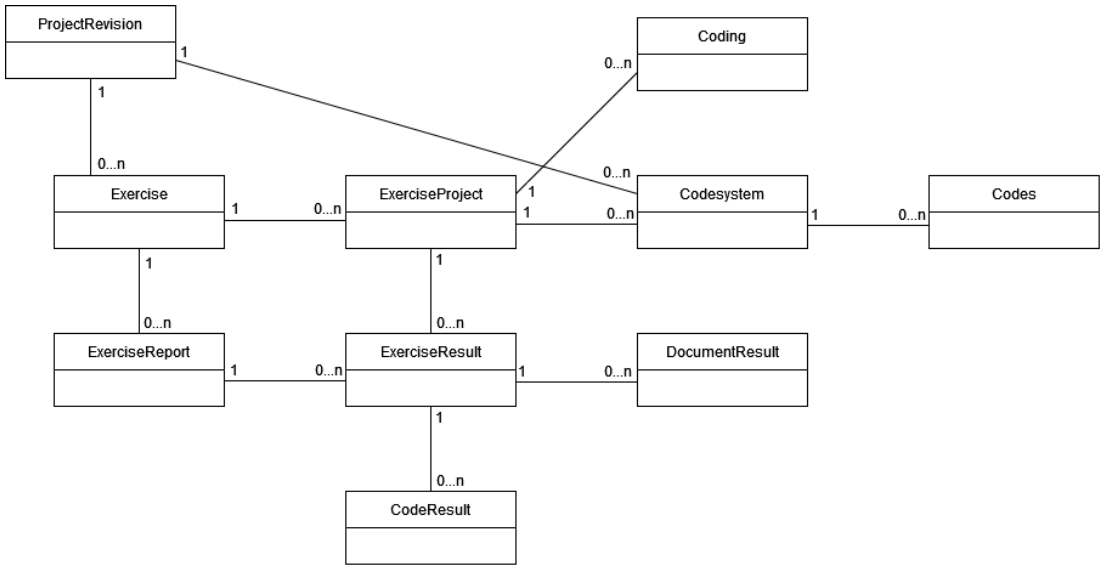

Abbildung 3.1: Ausgangslage im Klassendiagramm

Die vom Kursleiter erzeugte Musterlösung ist als ein Objekt der Klasse *ProjectRevision* gespeichert. Auf dieser Grundlage können beliebig viele *Exercises* erstellt werden. Für jeden Kursteilnehmer, der die Aufgabe bearbeitet, wird ein *Exercise-*

Project erzeugt. Zu diesem werden die *Codings* gespeichert und beim Aufgabentyp *Codebuch Erstellung* auch das *Codesystem* mit den entsprechenden *Codes*, die im Projekt zur Verfügung stehen. Im Falle des Aufgabentyps *Codebuch Verwendung* wird aus dem *ExerciseProject* auf das *Codesystem* des *ProjectRevision* referenziert. Der Kursleiter hat die Möglichkeit einen *ExerciseReport* zu erstellen. Dazu wird für jedes der *Exercise* zugeordnetem *ExerciseProject* ein *ExerciseResult* erzeugt. In diesem werden die erreichten Werte für F-Maß, Trefferquote und Genauigkeit gespeichert. Für die Berechnung dieser Werte wird auf zwei andere Objekte zurückgegriffen, die bei der Erstellung des *ExerciseReport* erzeugt werden. Zum einen wird ein *DocumentResult* erzeugt, in dem gespeichert ist, welche Codes RP, FN und FP in den jeweiligen Absätzen des kodierten Dokumentes sind. Zum anderen werden im *CodeResult* pro Dokument und Code die F-Maße, Trefferquoten und Genauigkeiten berechnet. Diese werden dann für die Werte im *ExerciseResult* aggregiert. Im *ExerciseReport* stehen diese wiederum aggregiert über alle Teilnehmer dem Kursleiter zu Verfügung.

3.2 Abgabe einer Hausaufgabe (Snapshot)

Die Erstellung von Snapshots unterschiedlicher Abgaben ist die Grundlage des Feedbacks, da nur aufgrund dieser gespeicherten Daten die Abgabe bewertet werden soll. Dafür muss die Abgabe komplett aus den gespeicherten Daten rekonstruierbar sein. Der Aufgabentext muss nicht zusätzlich gespeichert werden, da bei den Übungen alle Kursteilnehmer den gleichen Text codieren. Das *Codebuch* muss nur bei Aufgaben vom Typ *Codebuch Erstellung* gespeichert werden (*Codes*), die *Codings* müssen aber auf jeden Fall abgespeichert werden. Für die Funktion wurde ein neue, in Abbildung 3.2 illustrierte, persistente Java-Klasse implementiert.

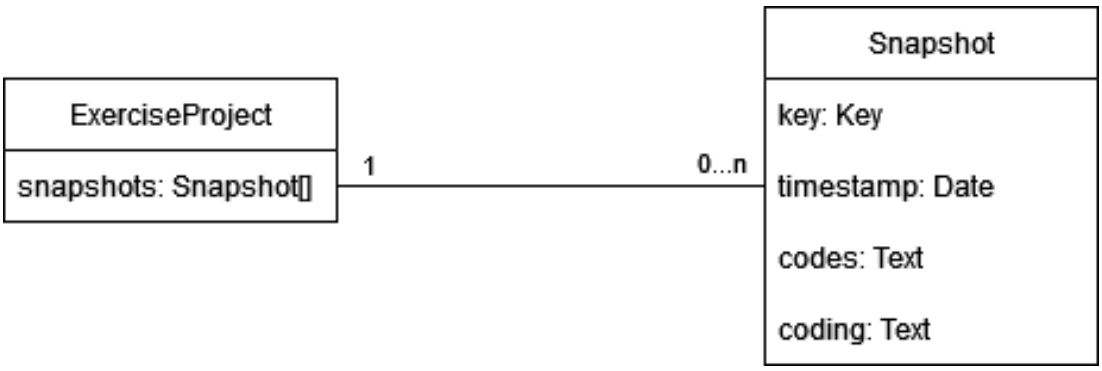

Abbildung 3.2: Klassendiagramm Snapshot

Jedes *ExerciseProject*, das einer Übungsaufgabe eines einzelnen Kursteilnehmers entspricht, kann keinen oder eine beliebige Anzahl an Snapshots haben - nämlich genau so viele, wie oft der Kursteilnehmer seine Bearbeitung abgegeben hat.

Die Abgabe (Snapshot) beinhaltet dabei den Zeitpunkt der Abgabe (*timestamp*) und die *Codings*. Der Abgabezeitpunkt ist wichtig, um für die Bewertung diejenige Abgabe auswählen zu können, die als Letztes vor dem Abgabetermin der Aufgabe gemacht worden ist. Der Snapshot enthält *Codes* nur für den Fall einer Aufgabe vom Typ *Codebuch Erstellung*, *Codings* aber auf jeden Fall. Für eine optimierte Speicherung werden diese beiden Attribute, die eigentlich jeweils ein Feld mit Elementen des Datentyp *Codings* bzw. *Codes* sind, zum Datentyp Text umgewandelt. Dafür werden zunächst mithilfe der Java-Bibliothek Jackson die Objekte in ein JSON Format transformiert, um sie dann zu einem String-Format umwandeln zu können und im Text-Format abzuspeichern. Das hat den Vorteil, dass die Abgabe komplett rekonstruiert werden kann und parallel dazu die ursprünglichen *Codings* und *Codes* veränderbar bleiben. Im Frontend gibt es in JavaScript schon nativ einen JSON-Parser mit dem JSON-Text wieder in die entsprechenden Objekte umgewandelt werden kann. (Ackermann, 2021) Dadurch wird es möglich, die Abgabe im *Coding Editor* anzuschauen.

3.3 *Agreement Maps* für die Analyse der Schwachstellen innerhalb der Abgabe

Da Kursteilnehmer oft kritisierten, dass nicht ersichtlich ist, wo genau in der Abgabe sich die Fehler befanden, sollen diese mithilfe sogenannter *Agreement-Maps* angezeigt werden. Dabei sollen die Absätze farblich markiert werden, die besonders viele Abweichungen zur Musterlösung beinhalten. Wenn der Kursleiter einen *Report* erstellt und dadurch die Bewertung gestartet hat, wurden bereits in der ursprünglichen Implementierung für jeden Teilnehmer, die RP, FN und die FP absatzweise im *DocumentResult* gespeichert. Da die bisherige Datenstruktur aber genau festgehalten hat, welches *Coding* vergessen wurde und welches fälschlicherweise gesetzt worden ist, wurde das Ergebnis nicht an den Teilnehmer herausgegeben, denn die Verbreitung dieser Details wäre bei wiederverwendeten Texten in zukünftigen Semestern problematisch und Anforderung NFA 5.4 nicht erfüllt. Dadurch könnten die Lösungen unter den Studenten weitergeben werden und eine faire Benotung wäre unmöglich. Deshalb ist im Rahmen dieser Bachelorarbeit eine neue persistente Klasse *AgreementMap* entstanden, die die vorher bereits gespeicherten Daten so aggregiert, dass sie gefahrlos den Studenten übermittelt werden können und die Anforderung (NFA 5.4) erfüllen. Diese speichert nur, wie viele RP, FN und FP im Absatz vorkommen. Die Information, um welche Codings es sich genau handelt ist nicht mehr enthalten.

3.4 Perzentil Berechnung

Um das erreichte Perzentil einer Abgabe mit dem F-Maß auszurechnen, müssen alle F-Maße eines Reports geordnet und geprüft werden, an welcher Stelle der jeweilige Wert einzuordnen ist. Um die Anforderung zur Performanz (NFA 2.1) zu erfüllen, soll die Sortierung der Werte nicht im Frontend passieren, sondern im Backend. Dadurch reduziert sich die Wartezeit für den Endnutzer. Die aufwändige Sortierung wird bereits bei der Reporterstellung getätigt und die geordnete Liste wird dann in der Datenbank als neues Attribut des *Reports* gespeichert. Gleichzeitig werden Abgaben herausgefiltert, die keine *Codings* gesetzt haben, um Verzerrungen der Auswertung durch Studenten zu vermeiden, die die Übung nicht gemacht haben.

3.5 Review und ReviewFeedback

Für das gewünschte Review, bei dem sich auf der einen Seite die Kursteilnehmer gegenseitig und auf der anderen Seite der Kursleiter einzelne Abgaben bewerten, gibt es im Backend bisher noch keine Datenstrukturen. Daher wurden im Zuge der Bachelorarbeit die beiden persistenten Klassen *Review* und *ReviewFeedback* angelegt, die in Abbildung 3.3 zu sehen sind.

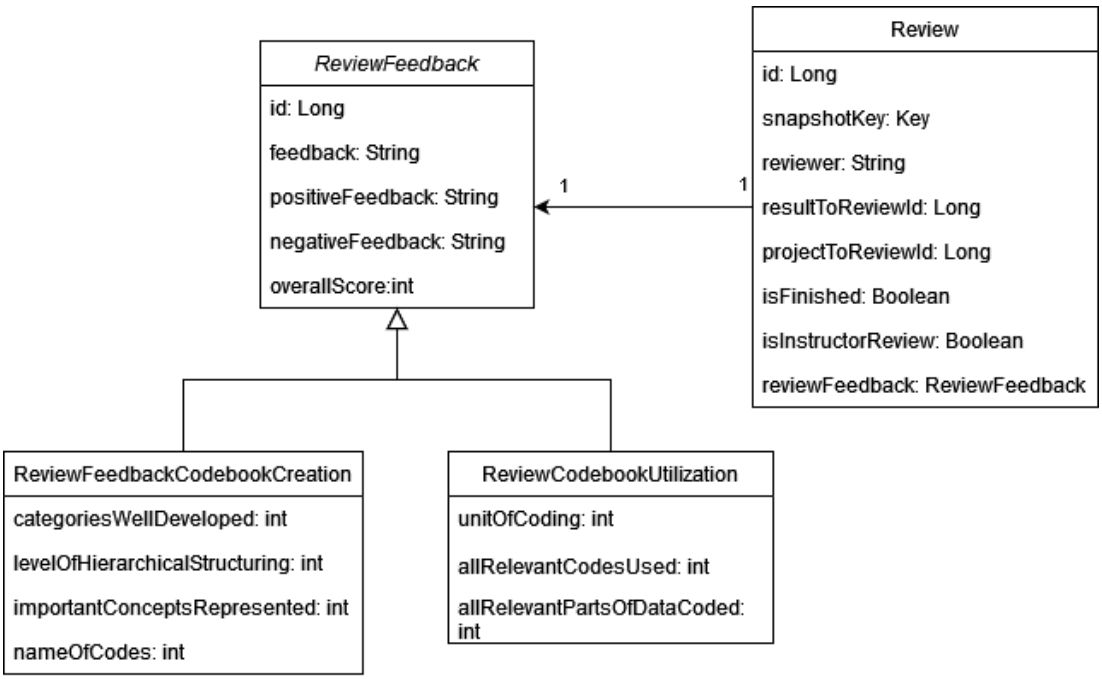

Abbildung 3.3: Klassendiagramm mit *Review* und *ReviewFeedback*

Dabei enthalten die *Review*-Objekte alle Metadaten, die für das Peerreview gespeichert werden sollen - also wer, welches *ExerciseResult* mit zugehörigem *ExerciseProject* und *Snapshot* bewertet und ob es schon abgeschlossen ist. Dies sind Daten, die nach den Sicherheitsanforderungen NFA 5.3 und NFA 5.4 nicht für die Teilnehmer einsehbar sein dürfen, da das Review anonym bleiben soll. Einzig der Kursleiter soll die Reviews zu Kursteilnehmern zuordnen können. Bei dem *Review*-Objekt eines Kursleiters ist das Attribut *isInstructorReview* gesetzt. Außerdem hat das *Review*-Objekt eine Referenz auf ein *ReviewFeedback*-Objekt, in dem dann die eigentliche Bewertungen gespeichert wird. Ein Vorteil dieser Aufteilung ist, dass man mit Vererbung unterschiedliche Feedbackarten modellieren kann. Dies stellt auch die in NFA 6.3 geforderte Erweiterbarkeit sicher. So gibt es für die beiden Aufgabentypen *Codebuch Verwendung* und *Codebuch Erstellung* jeweils eine einzelne Unterklasse, die von der Klasse *ReviewFeedback* erben. *Feedback, positiveFeedback, negativeFeedback* und *overallScore* sind Attribute, die für beide Aufgabentypen von Bedeutung sind. Daher werden diese Attribute in der Oberklasse *ReviewFeedback* gespeichert und an die Unterklassen vererbt. Die Fragen, die mit einer Likert-Skala beantwortet werden sollen, sind jedoch an die unterschiedlichen Anforderungen der Aufgabentypen angepasst und deshalb als Attribute der Unterklassen *ReviewFeedbackCodebookUtilization* und *ReviewFeedbackCodebookCreation* modelliert. Die Bewertung der Likert-Fragen werden dabei als einzelne Attribute mit Datentyp Integer gespeichert. Wenn ein neues Review angefordert ist, wird je nach Aufgabentyp ein Objekt der jeweiligen Unterklasse von *ReviewFeedback* erzeugt und von dem neu angelegtem *Review*-Objekt aus referenziert.

3.6 Datenfluss des Reviews

Neben den persistenten Klassen, die in der Datenbank gespeichert werden, muss auch der Datenfluss implementiert werden, um Reviews zu speichern und diese auch wieder anschauen zu können. Die Schritte, die dabei durchlaufen werden, sind in der folgenden Abbildung 3.4 visualisiert. Im Frontend wird in der Klasse *ReviewEndpoint* der Request mit einem API-Aufruf auf der Clientseite abgeschickt, der seine Antwort mit einem *Promise* erwartet. Ein *Promise* ist ein Stellvertreter für ein Objekt, das erst später zur Verfügung stehen wird. (Narayn, 2022) Die Anfrage wird dann auf der Serverseite in der Klasse *ReviewEndpoint* verarbeitet, wenn der Request die passende HTTP-Methode und die geforderten Parameter hat. Ansonsten wird der Fehlercode *404 Bad Request* zurückgeschickt. (Ackermann, 2021) Im betreffenden Endpunkt der Klasse *ReviewEndpoint* wird dann die in NFA 5.1 verlangte Autorisierung geprüft. Je nachdem, welche Berechtigung angefordert ist, wird geprüft, ob der Nutzer einer bestimmten Personengruppe angehört. Das *ReviewFeedback* anschauen darf der Beurteilende, der Beurteilte oder der Kursleiter. Das *ReviewFeedback* bearbeiten darf nur der Be-

urteilende. Ein neues Review anfordern darf jeder, der den Kurs belegt. Versucht eine nicht für die Anfrage autorisierte Person zuzugreifen, dann antwortet das Backend mit einem *401 Unauthorized* Status. Ist die Anfrage geprüft, wird eine Methode in der *ReviewController* Klasse aufgerufen, die, die angeforderte Bearbeitung ausführt. Wenn im Zuge dessen Daten persistiert werden müssen, ruft der Controller eine Methode in der DAO-Klasse auf. Dies erfüllt die Anforderung NFA 6.4 zur Trennung der Zuständigkeiten. DAO steht für *Data Access Object* und ist eine Klasse, die direkt auf die Datenbank zugreift und diese verändert. Für das Review und das ReviewFeedback gibt es zwei separate DAOs. Nach erfolgreicher Bearbeitung erhält das *Promise*-Objekt die Antwort des Servers. Diese kann in den *state* der Komponente geschrieben werden. (Springer, 2023) Wie in Kapitel 3.5 erläutert, wird dabei niemals das *Review*-Objekt an sich zurückgegeben, sondern nur das zugehörige *ReviewFeedback*-Objekt.

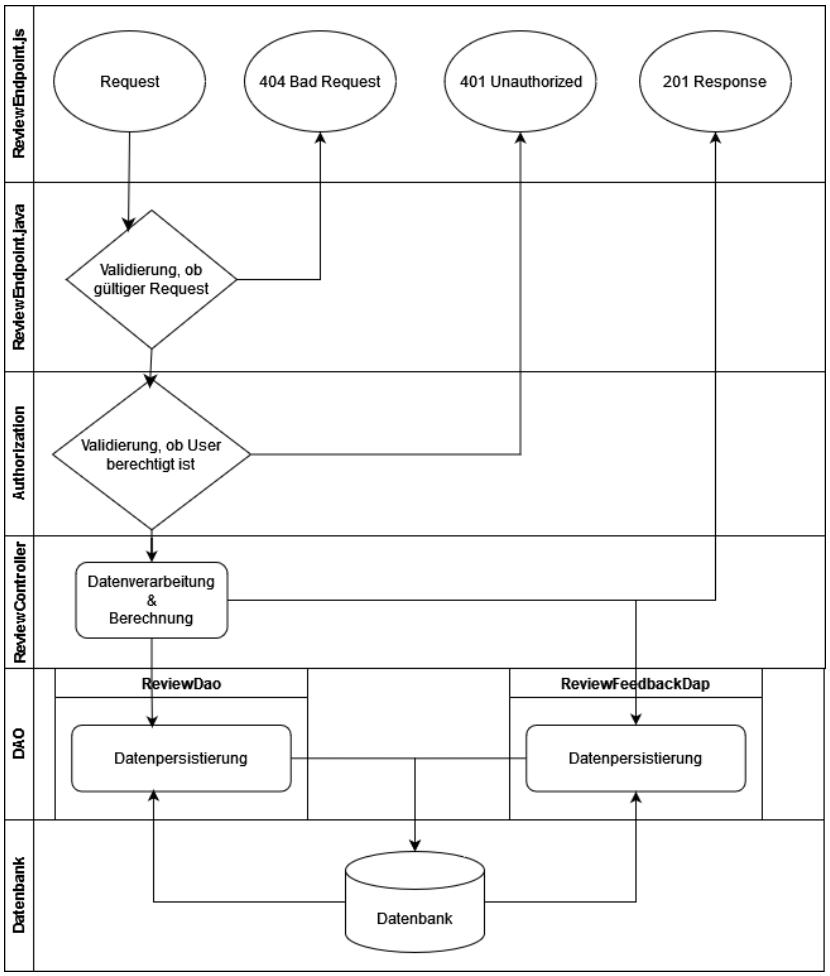

Abbildung 3.4: Datenfluss des Reviews

3.7 Zuordnung von Peerreviews

Für die Zuordnung von Peerreviews zu den Kursteilnehmern wurde die Klasse *ExerciseResult* um die beiden Felder *reviewsReceived* und *reviewsGiven* erweitert, die jeweils den Datentyp *Array<Long>* haben. In diesen werden die IDs der Reviews gespeichert, sobald diese anfordert worden sind.
Mit dem API-Aufruf *getResultToReview* kann ein Kursteilnehmer ein Review anfordern. Dazu wird zunächst geprüft, ob das letzte von ihm angeforderte Review bereits abgegeben worden ist, indem geprüft wird, ob im Review, auf welches das letzte Element des Feldes *reviewsGiven* referenziert, das Attribut *isFinished* auf wahr gesetzt ist. Falls nicht, wird dieses zurückgegeben.
Ansonsten wird ihm ein *ExerciseResult* eines anderen Kursteilnehmers zugeteilt, das wie folgt ausgewählt wird: Zunächst wird eine Liste mit allen *ExerciseResults*, die dem gleichen *ExerciseReport* zugeordnet sind, erstellt und diese aufsteigend nach der Anzahl der Reviews sortiert, die im jeweiligen *ExerciseResult* im Feld *reviewsReceived* gespeichert sind. Dadurch wird zwar sichergestellt, dass die Abgaben ähnlich vielen Kursteilnehmern als Review zugeordnet werden, aber nicht, dass auch alle ähnlich viele Rückmeldungen erhalten, da das Feld *reviewReceived* auch Reviews enthält, die noch nicht fertiggestellt wurden und dies vielleicht auch überhaupt nicht werden. Dies muss hingenommen werden, da eine 100% gleichmäßige Verteilung nur durch eine fest definierte Zuordnungstabelle umsetzbar wäre, die eine feste Anzahl an Peerreviews, die von jedem Kursteilnehmer durchgeführt werden müssen, voraussetzt. Im Anschluss wird das erste *ExerciseResult* ausgewählt, dass weder das eigene noch eins ist, das bereits vom Kursteilnehmer, der das neue Review angefordert hat, bewertet worden ist. Wenn die so erzeugte Liste nach dem Filtern leer ist, bedeutet dieses, dass der Review anfordernde Teilnehmer bereits alle anderen Kursteilnehmer bewertet hat und somit kein weiteres Review mehr durchführen kann. Dies wird dem Frontend durch einen *null-Wert* als Rückgabe signalisiert. Ansonsten wird ein neues *Review*-Objekt mit der *userId* des Reviewers und der Id des zu reviewenden *ExerciseResults*, dessen *SnapshotId* und *projektId* initialisiert. Zusätzlich wird abhängig vom Aufgabentyp auf ein neues Objekt einer Unterklasse von *ReviewFeedback* referenziert. Die Id des neuen *Review*-Objektes wird dann sowohl im *ExerciseResult*, welches demjenigen gehört, der das Review durchführt, im Feld *reviewsGiven* hinzugefügt als auch bei demjenigen, dem die Abgabe gehört, im Feld *reviewsReceived* gespeichert. Die Feedback erhaltene Person ist jedoch nur autorisiert auf das Review über die Methode getReviewFeedbackByIdIfFinshed zuzugreifen und erhält somit erst Einblick in das Feedback, wenn das Review abgeschlossen worden ist.

3.8 Automatisierte Reporterstellung für Aufgabentyp *Codebuch Erstellung*

Besonders für die Aufgabenbewertung des Typs *Codebuch Erstellung* hat das Peerreview große Vorteile. Da bei diesem Aufgabentyp die Kursteilnehmer die Codes, mit denen die Texte kodiert werden, selber erzeugen, kann kein automatisiertes Feedback gegeben werden, denn die Berechnung des *Intercoder Agreement* basiert darauf, die Verwendung der Codes zwischen der Musterlösung und der Abgabe zu vergleichen. Wenn die Codes aber individuell gewählt werden, gibt es keine berechenbare Übereinstimmung, die eine Aussagekraft über die Qualität der Abgabe hätte. Beim Peerreview wird dagegen jede Abgabe einzeln betrachtet, sodass sich auch die Abgaben mit eigenen Codes bewerten lassen. Um dies zu unterstützen, werden die Bewertungsfragen im Reviewprozess an den Aufgabentyp angepasst.
Als Voraussetzung des Reviewfeedbacks muss vom Kursleiter ein Report erstellt werden. Denn damit wird zum einen ein *ExerciseResult*-Objekt und zum anderen ein *Snapshot*-Objekt für jeden Kursteilnehmer erzeugt, der die Aufgabe bearbeitet hat, und ein *ExerciseReport*-Objekt, welches Referenzen auf alle *ExerciseResult*-Objekte hat. Auf diesen Objekten basiert die Reviewzuordnung, wie in Kapitel 3.7 erläutert. Beim Aufgabentyp *Codebuch Verwendung* wird diese Reporterstellung manuell vom Kursleiter ausgelöst und kann dabei in Bezug auf die *Intercoder Agreement*-Berechnung konfiguriert werden. Eine Reporterstellung für den Aufgabentyp *Codebuch Erstellung* war bisher nicht implementiert, da eine Berechnung des F-Maßes nicht gewollt war. Für den implementierten Peer-Reviewprozess sollte die Reporterstellung jetzt automatisch getriggert werden, wenn die Abgabefrist von einer Aufgabe mit Typ *Codebuch Erstellung* abgelaufen ist. Weil auch weiterhin auf ein automatisiertes Feedback inklusive Berechnung der Übereinstimmung verzichtet wird und deshalb keine Konfiguration durch den Kursleiter nötig ist, muss der Prozess nicht manuell gestartet werden.
Um die Reporterstellung zu automatisieren, wird der Cron-Dienst von Google App Engine[1] verwendet. Mit diesem Dienst können sogenannte Cronjobs definiert werden, die zu einem festgelegten Intervall eine HTTP-GET-Anfrage an die Anwendung stellt. Für die Reporterstellung wurde ein Cronjob konfiguriert, der einmal täglich die Klasse *ExerciseDeadlineServlet* anspricht. Wenn diese Klasse getriggert wird, erzeugt sie für alle Aufgaben, deren Abgabezeitpunkt vergangen ist und noch keine Snapshots erzeugt worden sind, Snapshots und für den Aufgabentyp *Codebuch Erstellung* einen Report ohne *Intercoder Agreement* Berechnung.

[1] https://cloud.google.com/appengine/docs/legacy/standard/java/config/cron-yaml?hl=de

3.9 Bewertung des Feedbacks

Zusätzlich wurde auch die Funktion implementiert, ein erhaltenes Feedback zu bewerten. Dazu wurde die neue persistente Klasse *FeedbackOfReviewFeedback* mit den in Abbildung 3.5 dargestellten Attributen hinzugefügt. Der Autor der Abgabe kann dann die Bewertung zum Feedback, welche zum einen aus einer Punktzahl und zum anderen aus einem Kommentar bestehen kann, mit dem neuen Endpunkt *setFeedbackOfReviewFeedback* speichern, sobald er ein Peerreview erhalten hat.

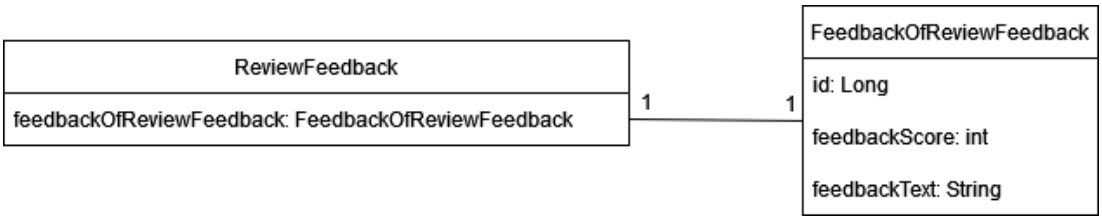

Abbildung 3.5: Klassendiagramm *FeedbackOfReviewFeedback*

3.10 Berechnung der Standardabweichung der Gesamtbewertung aller erhaltener Peerreviews einer Abgabe

Innerhalb der Bachelorarbeit wurde auch die Funktion, als Betreuer einzelne Abgaben zu bewerten, implementiert. Da die Teilnehmerzahl wie im Kurs NYT oft zu hoch ist, um jede Abgabe bewerten zu können, muss der Betreuer einzelne Abgaben auswählen. Ein Maß, das die Entscheidung, wer ein Betreuerfeedback erhalten soll, beeinflussen könnte, ist die Standardabweichung der Gesamtbewertung aller erhaltener Peerreviews einer Abgabe. Denn wenn die Abweichung sehr hoch ist, dann waren die Teilnehmer, die die Abgabe bewerten haben, sich nicht über die Qualität der Abgabe einig. Ein Betreuerfeedback kann dann für Aufklärung sorgen. Dafür wird die Standardabweichung für jedes *ExerciseResult* berechnet, sobald ein Peerreview abgegeben worden ist, also mit Aufruf des Endpunktes *submitReview*. Für die Berechnung werden alle erhaltenen Reviews einer Abgabe, die im Attribut *reviewsReceived* des *ExerciseResults*-Objektes gespeichert und bereits abgegeben worden sind, geladen und deren Gesamtbewertungen in das Array *overallScores* der Länge n gespeichert. Dann wird zunächst das arithmetische Mittel $\overline{x}$ (Benning, 2023a) dieser Gesamtbewertungen berechnet und damit dann die Standardabweichung σ (Benning, 2023b):

$$\overline{x} = \frac{1}{n} \cdot \sum_{i=0}^{n-1} overallScores[i] \tag{3.1}$$

$$\sigma = \sqrt[2]{\frac{1}{n} \cdot \sum_{i=0}^{n-1} (overallScores[i] - \overline{x})^2} \tag{3.2}$$

Die berechnete Standardabweichung wird dann als neues Attribut *standardDeviation* im *ExerciseResult* gespeichert.

4 Design und Implementierung

In diesem Kapitel wird das Design und die Implementierung der Anforderungen auf Grundlage der in Kapitel 3 vorgestellten Architekturentscheidungen dargestellt.

4.1 Abgabe einer Hausaufgabe

Die Abbildung 4.1 zeigt das *ExerciseDashboard* mit der neuen Abgabefunktion.

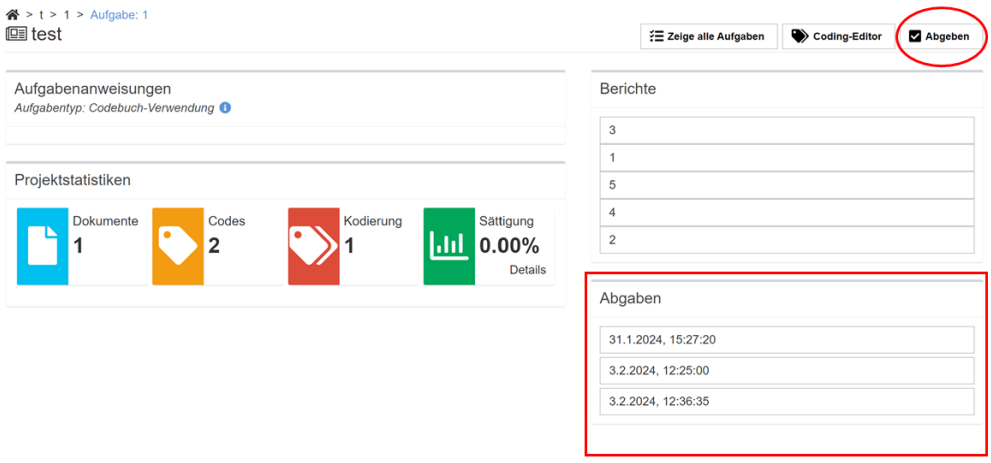

Abbildung 4.1: *ExerciseDashboard* mit neuer Abgabefunktion

Im Frontend wurden bereits verwendete Bauteile des existierenden Designsystems übernommen, um dem Nutzer ein konsistentes Erscheinungsbild zu geben. So wurde aus der QDAcity Bibliothek sowohl der Button, der das Abspeichern des Projekts auslöst und vom Design an dem *Coding Editor* Button angepasst ist,

als auch eine ItemList zur Anzeige der gemachten Abgaben genutzt, deren Layout mit dem der Berichte-Liste übereinstimmt. Jede einzelne Abgabe erhält als Listeneintrag den Zeitpunkt, an dem die Abgabe generiert wurde. Beim Klick auf einen Eintrag öffnet sich die Abgabe im *Coding Editor*. Dazu wird der Snapshot mit *JSON.parse* wieder zu *Coding* und *Code* Objekten umgewandelt, die dann als Labels am Text erscheinen. Diese sind im *Coding Editor* jedoch nicht weiter bearbeitbar, damit die Abgabe nicht im Nachhinein weiter verändert werden kann. Der Button *Abgeben* löst beim Anklicken die neue HTTP-Rest-Methode *addSnapshotToExerciseProject* aus, die im Backend dann wie in Kapitel 3.2 erklärt, einen neuen Snapshot anlegt.

4.2 Automatisiertes Feedback

In diesem Kapitel wird verdeutlicht, wie das berechnete *Intercoder Agreement* beim Aufgabentyp *Codebuch Verwendung* genutzt wird, um den Kursteilnehmern ein Feedback über die Qualität ihrer Abgabe zu können.

4.2.1 Verdeutlichung der Bewertung der Leistung innerhalb des eigenen Projekts

Mit dieser Funktion sollen die Kursteilnehmer ihre Schwächen innerhalb der Abgabe finden. Dazu gibt es auf der Übersichtseite eines Reports, die die Teilnehmer öffnen können, nachdem die Abgabe ausgewertet worden ist, einen neuen Button *Agreement Map*.

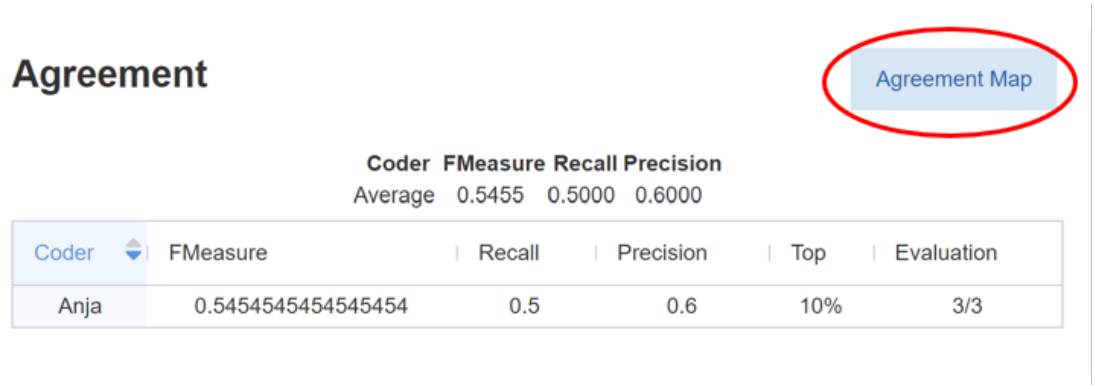

Abbildung 4.2: *Agreement Map*-Button auf der Übersichtsseite der Aufgabenbewertung

Durch Anklicken dieses Buttons öffnet sich der *Coding Editor*. Doch statt den gewöhnlichen Tabs *Coding, Editor* und *Code* gibt es nun die beiden Ansichten *Agreement pro Absatz* und *Agreement pro Code*, die als *Views* im Datenmodell hinzugefügt wurden.

Agreement pro Absatz

Die standardmäßig angezeigte Seite ist das *Agreement pro Absatz*. Hier wird die Abgabe, auf deren Grundlage der Report erstellt wurde, im *Coding Editor* angezeigt. Neu ist die hinzugefügte rechte Seitenleiste, welche die in Kapitel 3.3 angesprochenen *Agreement Maps* zeigt.

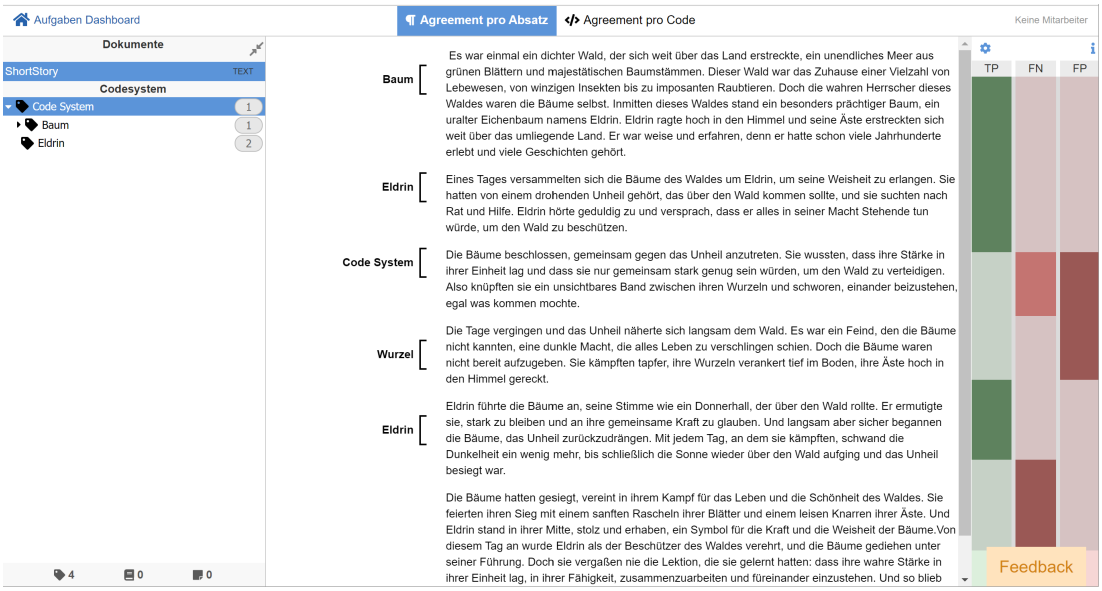

Abbildung 4.3: Ansicht Agreement pro Absatz

In dieser Seitenleiste können mithilfe einer Mehrfachauswahl ein, zwei oder auch alle drei Leisten parallel angezeigt werden. Dazu wurde ein DropDown Menü implementiert mit dem die verschiedenen *Agreement Maps* ausgewählt werden können, welches durch das Anklicken des Zahnradsymbols erscheint. Beim Öffnen des rechten Info Icons erhält der Teilnehmer mehr Informationen zur Bedeutung der Einfärbungen.
Wie in FA 2.1.3 gefordert, soll die Färbung Hinweise über die Qualität des kodierten Absatzes geben. Da die RP-Ergebnisse positiv zu bewerten sind, wird diese Leiste grün gefärbt. Die Schattierung wird dabei entsprechend dem Anteil der RP ihrer möglichen Gesamtzahl im betreffenden Absatz gewählt. Die Anzahl, der in der Abgabe gefundenen Codes, wird durch die Summe, der in der Musterlösung vorhandenen Codes, geteilt. Um die Anzahl der Codes in der Musterlösung auszurechnen, wird RP, also die, die auch gefunden wurden, mit FN addiert, also die, die vergessen wurden. Der dunkelste Grünton wird erreicht, wenn über 90% aller Codes im Absatz gefunden wurden. Weitere Trennung wurde bei 60%, 30% und 0% durchgeführt. Für die FN und FP Leisten wird jeweils das Maximum der Codes ausgerechnet, die in einem Absatz falsch gesetzt worden sind und dann linear die Anzahl an Fehler in 5 Gruppen mit jeweils gleicher Rotschattierung unterteilt. Gab es in einem Absatz zum Beispiel 10 FN Codes und alle anderen

Absätze hatten weniger FN, dann wird die Fehleranzahl in 5 Gruppen unterteilt, sodass die Absätze mit 0 bis 2 FN die hellste Rotfärbung haben, 3 und 4 eine Stufe dunkler sind und bis 9 und 10 Fehler die dunkelste Färbung zeigen. Dadurch ist sichergestellt, dass jeder Teilnehmer den für ihn problematischsten Absatz findet, egal wie viele falsche Codes gesetzt worden sind.
Die *Agreement Maps* werden so skaliert, das sie zusammen den Platz in der rechten Leiste ausfüllen. Ist die Texthöhe größer als das Ansichtsfenster, werden die einzelnen Absätze der *Agreement Maps* relativ zur eigenen Höhe gestaucht, ist der Text kürzer, werden die Absätze gestreckt. Um die genaue Höhe der Absätze in der *Agreement Map* auszurechnen, muss erst die Höhe, die unter der Spaltenbeschriftung bleibt, mit folgender Formel ausgerechnet werden:

$$h_{Balken}=h_{DokumentWrapper} - h_{IconLeiste} - h_{Spaltenbeschriftung} \tag{4.1}$$

Für die Höhe des *DokumentWrapper*, der die Textanzeige umschließt, lässt sich dessen *clientHeight* verwenden, die exakt der Höhe des angezeigten Textes entspricht. Die Höhe der Icon Leiste und der Spaltenbeschriftung sind konstante Werte. Um zu erkennen, welche Absätze aus der *Agreement Map* gerade zum Abschnitt des Textes, der im Ansichtsfenster des Nutzers angezeigt wird, gehört, wird der Teil grau schattiert. Für diese Visualisierung muss auch die Höhe des Ansichtsfensters (engl. Viewport) berechnet werden:

$$h_{Ansichtsfenster}=\frac{h_{DokumentWrapper}}{h_{Dokument}} \cdot h_{Balken} \tag{4.2}$$

Es wird also erst der Anteil vom Text, der im Wrapper gerade sichtbar ist berechnet und dann auf die Höhe skaliert, die für die Balken zur Verfügung steht.
Zusätzlich muss noch die Position des Schattens berechnet werden, da sich dieser beim Scrollen des Textes mit verschieben soll. Somit wird gewährleistet, dass immer der sichtbare Teil des Textes markiert ist. Die Position der Schattierung wird mit folgender Formel berechnet:

$$p_{Schattierung}=\frac{h_{Scrollen}}{h_{Dokument}} \cdot h_{Balken} + h_{Spaltenbeschriftung} \tag{4.3}$$

Es wird also die Anzahl der Pixel, um die nach unten gescrollt worden ist durch die Pixelanzahl des gesamten Dokumentes dividiert, um mit diesem Faktor die Höhe der Balken der *Agreement Maps* zu skalieren. Da die Schattierung dann aber mit der Tabelle beginnen würde, wird diese noch um die Höhe der Spaltenbeschriftung nach unten verschoben.

Agreement pro Code

Neben der Analyse der Ähnlichkeit der Codierung einzelner Absätze mit der Musterlösung, kann der Kursteilnehmer auch zur Ansicht *Agreement pro Code* wechseln.

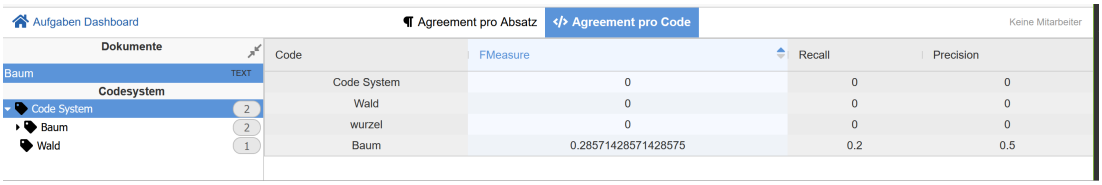

Abbildung 4.4: Ansicht Agreement pro Code

Auf dieser Seite sieht der Teilnehmer, welche Codes besonders ähnlich zur Musterlösung kodiert wurden und welche mehr Probleme bereitet haben. Dazu werden ihm zu jedem Code F-Maß, Trefferquote und die Genauigkeit aus dem berechneten Intercoder Agreement angezeigt. Die Tabelle ist aufsteigend sortiert nach den F-Maßen, sodass der Teilnehmer im oberen Bereich der Tabelle die problematischen Codes sieht. Für die Berechnung werden die *CodeResults* zum zugehörigen Projekt und Report vom Backend durch einen API-Aufruf angefordert. Da diese jedoch im *CodeResult* für jedes Dokument nur einzeln vorhanden sind, muss noch der Durchschnitt für alle vom Kursteilnehmer kodierten Dokumente ausgerechnet werden. Diese Durchschnittswerte werden dann tabellarisch zur Verfügung gestellt.

4.2.2 Verdeutlichung der Bewertung der Leistung im Vergleich zu den anderen Kursteilnehmern

Nachdem im Kapitel 4.2.1 aufgezeigt wurde, wie es dem Kursteilnehmer ermöglicht wurde, seine eigenen Schwächen der Abgabe einzusehen, wird nun in diesem Kapitel beschrieben wie eine Einordnung innerhalb des Kurses geschaffen wird, um die eigene Leistung besser einschätzen zu können und etwaige Verunsicherungen, die durch ein niedriges F-Maß kommen können, zu beseitigen.
Wie in Kapitel 3.4 erklärt, werden dafür bereits die F-Maße eines Reports sortiert gespeichert. Dieses Array wird im folgenden *sortierteFMaße* genannt und hat eine Länge l. Dadurch wird die Berechnung der Perzentile in 10% Schritten möglich. Die Entscheidung, diese Berechnung im Frontend zu tätigen, basiert auf der Überlegung in Zukunft vielleicht auch klein- oder grobgranularer die Perzentile auszurechnen. Aktuell wurden 10% Schritte realisiert. Für jeden dieser Schritte t wird das Perzentil mit folgender bekannter Formeln ausgerechnet:

Für $np{=}l \cdot t$ ganzzahlig:

$$Perzentil{=}0.5 \cdot (sortierteFMaße[np] + sortierteFMaße[np+1]) \tag{4.4}$$

sonst:

$$Perzentil{=}sortierteFMaße[\lceil np \rceil] \tag{4.5}$$

Wenn alle Perzentil-Werte ausgerechnet wurden, kann das F-Maß der Abga-

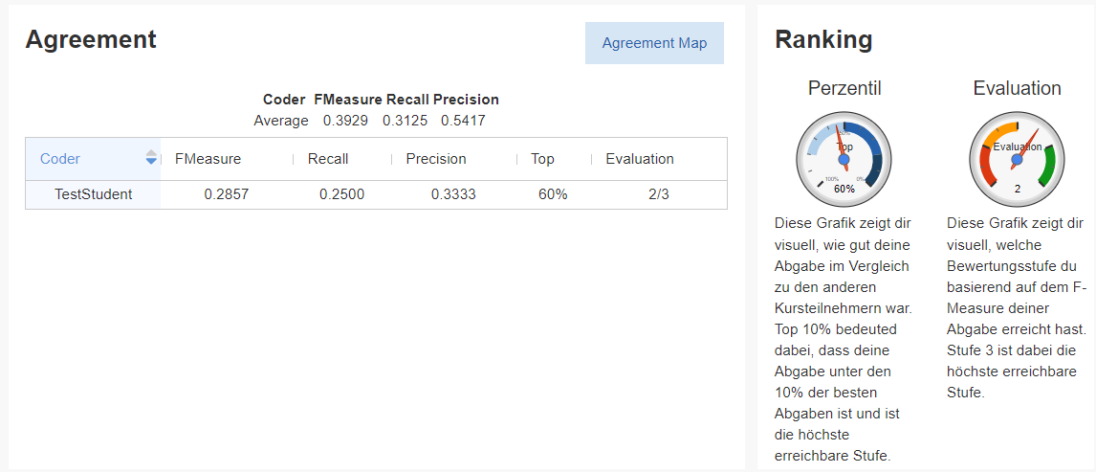

Abbildung 4.5: Übersicht mit Angabe des erreichten Perzentils

be eingeordnet und in der Tabelle hinzugefügt werden. Damit der Wert nicht in der Tabelle übersehen werden kann, wird dieser dem Teilnehmer zusätzlich mit einer Grafik, die in Abbildung 4.5 dargestellt ist, visualisiert. Dazu wird von der Google Charts Bibliothek eine Anzeige[1] implementiert, die den Wert mit einer analogen Anzeigenadel verdeutlicht. Mit den blauen Schattierungen aus dem QDAcity Farbschema hinterlegt, vergrößert sich der Anzeigestand mit Erreichen höheren Perzentils. Unter der Anzeige wird auch erklärt, was das erreichte Perzentil aussagt. 10% bedeutet, dass die eigene Abgabe unter den besten 10% aller Abgaben liegt. Dies ist die höchste erreichbare Stufe.

4.2.3 Verdeutlichung der Bewertung der Leistung im Vergleich zum Bewertungsschema

Um das automatisierte Feedback abzurunden, fehlt noch eine Einordnung der Abgabe in das Bewertungsschema. Bisher haben Teilnehmer des Kurses NYT eine

[1] https://developers.google.com/chart/interactive/docs/gallery/gauge?hl=de

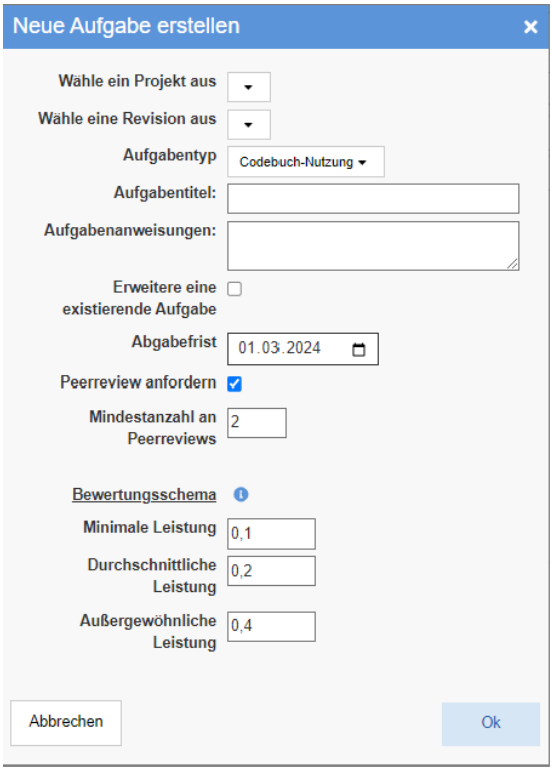

Abbildung 4.6: Dialog zum Erstellen neuer Aufgaben mit Bewertungsschema

Tabelle bekommen, die die F-Maße einer Punktzahl zuordnet. Diese Zuordnung soll nun automatisiert werden und wie das Perzentil mit einer Google Anzeige[1] visualisiert werden. Dazu hat der Kursleiter die Möglichkeit das Bewertungsschema individuell beim Erstellen einer neuen Aufgabe im Dialog (siehe Abbildung 4.6) zu konfigurieren. Für das Bewertungsschema stehen für die F-Maß Werte drei Felder zur Verfügung einer minimalen, einer durchschnittlichen und einer außergewöhnlichen Leistung. Initial sind diese Felder mit den Werten 0.1, 0.2 und 0.4 gefüllt. Dieses Bewertungsschema folgt der Empfehlung aus dem Paper: *A Solution for Automated Grading of QDA Homework* von Kaufmann et al. (2023). Da das Bewertungsschema, als neues Datenfeld *evaluationScheme* in der Tabelle *Exercise* gespeichert wird, kann dem Teilnehmer auf der Reportseite problemlos seine erreichte Punktzahl angezeigt werden. Das erreichte F-Maß muss nur in das Evaluationsschema eingeordnet werden. Wie das berechnete Perzentil wird auch die Punktzahl in einer Google Anzeige[2] visualisiert. Die einzelnen Punktzahlen sind dabei mit rot für 0 Punkte, orange für 1, weiß für 2 und grün für die höchste zu erreichende Punktzahl 3 farblich hinterlegt. Analog zur Anzeige des Perzentils wird auch im Infotext die Anzeige und dessen Bedeutung genauer erläutert. Die Ansicht mit Anzeige der Bewertungsstufe ist in Abbildung 4.5 zu sehen.

[2]https://developers.google.com/chart/interactive/docs/gallery/gauge?hl=de

4.3 Peerreview

Neben dem automatisierten Feedback wurde zusätzlich die Funktion zur Durchführung von Peerreviews implementiert, sodass Kursteilnehmer gegenseitig ihre Abgaben bewerten können. Der dieser neuen Funktionalität zugrunde liegende Prozess wird in den folgenden drei Unterkapiteln beleuchtet, nachdem zunächst die neuen Endpunkte vorgestellt werden.

4.3.1 Die neuen Endpunkte für das Review

Für die Funktionalität des Peerreviews sind die in Tabelle 4.1 dargestellten neuen Endpunkte in der Klasse *ReviewEndpoint* implementiert.

Tabelle 4.1: Neue Endpunkte in der Klasse *ReviewEndpoint*

Name	Kurze Beschreibung
getReviewFeedbackByIdIfFinished	Rückgabe des abgegebenen ReviewFeedbacks
getReviewFeedbackById	Rückgabe des ReviewFeedbacks
updateReviewFeedback updateReviewFeedbackCodebookCreation updateReviewFeedbackCodebookUtilization	Abspeichern des Reviewfeedbacks
submitReview submitReviewCodebookUtilization submitReviewFeedbackCodebookCreation	Abgeben und Speichern des Reviews
getSnapshotForReviewId	Rückgabe der zum Review gehörenden Abgabe
getDocumentsForReview	Rückgabe der zum Review gehörenden Dokumente
getSignedDownloadUrlsForDocumentIdForReview	Rückgabe der Url für das Download der Texte des Dokuments
getResultToReview	Anfordern eines neuen Reviews, siehe Kapitel 3.7
getResultWithNewInstructorReview	Anfordern eines Betreuerfeedbacks zu spezifizierten Abgabe

4.3.2 Konfiguration des Peerreviews bei Aufgabenerstellung

Der Kursleiter kann sich beim Erstellen einer neuen Aufgabe entscheiden, ob für diese ein Peerreview durchgeführt werden soll. Im positiven Fall kann er im Dialog zum Erstellen neuer Aufgaben, siehe Abbildung 4.6, den Haken bei Peerreviews setzen und die Anzahl an Peerreview, die jeder Teilnehmer mindestens machen soll, einstellen. Initial ist die Mindestanzahl auf 2 gesetzt, welches aber durch das Eingabefeld individuell angepasst werden kann. Für die Anzahl der Peerreviews gibt es keine Obergrenze, da Teilnehmer, die mehr Abgaben bewerten wollen, nicht eingeschränkt werden sollen. Die Mindestanzahl wird beim Erzeugen einer neuen Aufgabe in der Datenbank als neues Feld zum *Exercise-Object* gespeichert.

4.3.3 Verteilung der Peerreviews

Wenn die Abgabefrist einer Aufgabe, bei der Peerreviews gefordert sind, abgelaufen ist und der Kursleiter einen Report erstellt hat, findet der Teilnehmer auf der Übersichtseite des Reports zwei zusätzliche in Abbildung 4.7 illustrierte Kacheln. Die linke Kachel enthält eine Liste an Reviews, die der Teilnehmer erhalten hat,

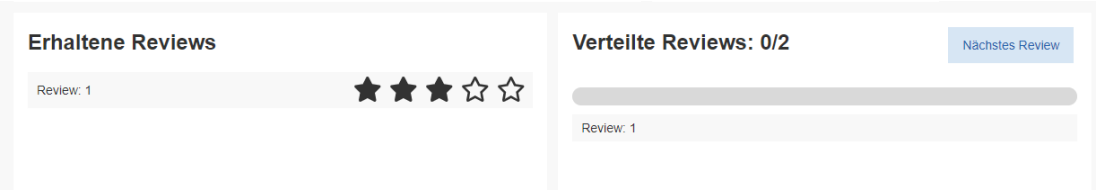

Abbildung 4.7: Kacheln mit erhaltenen und verteilten Reviews

und die rechte Kachel zeigt die von ihm erstellten und noch zu erstellenden Reviews an. Wie in der Abbildung 4.7 zu sehen, gibt es oben rechts in der Kachel *Verteilte Reviews* einen Button mit *Nächstes Review*. Beim Klicken dieses Buttons öffnet sich die Abgabe eines anderen Kursteilnehmers im *Coding Editor* mit einer Seitenleiste auf der rechten Seite, die Fragen über die Qualität der Abgabe enthält. Dazu wird ein *ExerciseResult* eines anderen Kursteilnehmers mit dem API-Aufruf an das Backend *getResultToReview* ausgewählt und beim Öffnen des Reviews dessen *Feedback*-Objekt (*getReviewFeedbackById*), die zugehörige Abgabe (*getSnapshotForReviewId*) und das Textdokument mit *getDocumentsForReview* und *getSignedDownloadUrlsForDocumentIdForReview* angefordert. Das geöffnete Feedback ist aus drei unterschiedlichen Fragetypen aufgebaut. Oben werden spezifisch auf den Aufgabentyp abgestimmte Fragen angezeigt, die mit einer Likert-Skala beantwortet werden. Beim Aufgabentyp *Codebuch Verwendung*, welches in Abbildung 4.8 zu sehen ist, wird gefragt, ob die vordefinierten Labels angemessen verwendet worden sind und beim Aufgabentyp *Codebuch Erstellung*, ob die selbst angelegten Labels sinnvoll gewählt und angewendet worden sind.

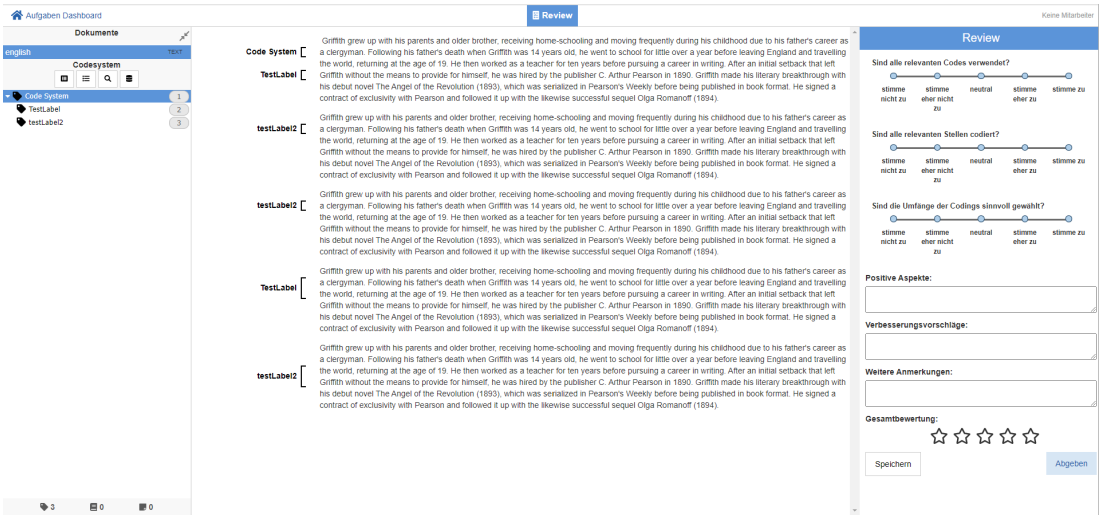

Abbildung 4.8: Review für Aufgabentyp *Codebuch Verwendung*

Zur Visualisierung der Bewertungsskala wird die Bibliothek *react-likert-scale*[3] verwendet, die in das Projekt importiert wurde. Die Bibliothek bildet die visuelle Grundlage und wurde hinsichtlich QDAcity Farben, Skalierung und Auswahlmöglichkeiten an das Projekt, wie in NFA 4.2 gefordert, angepasst. Nach den spezifischen Fragen folgen drei Freitexteingaben zu positiven Aspekten, Verbesserungsvorschlägen und weiteren Anmerkungen. Hier können die Teilnehmer Punkte nennen, die ihnen beim Betrachten der Abgabe aufgefallen sind. Als Letztes wird eine Gesamtbewertung in Form einer Sternbewertung gefordert, die als eigene Komponente designt und implementiert wurde. Der Beurteilende hat durch den Speicher-Button die Möglichkeit das angefangene Review zwischenzuspeichern und anschließend fortzuführen. Mit dem Backendaufruf *updateReviewFeedback* wird der aktuelle Stand der Bewertung in der Datenbank gespeichert. Der Reviewer kann die Bewertung nach Fertigstellung abgeben. Dabei wird zuerst geprüft, ob alle Fragen beantwortet und alle Felder gefüllt wurden. Dann wird mit einem Dialog nach einer Bestätigung zur Abgabe gefragt. Hierbei wird zusätzlich der Hinweis (siehe Abbildung 4.9), dass die Abgabe im Nachhinein nicht mehr verändert werden kann, angezeigt. Mit dem Aufruf *submitReview*, der durch das Bestätigen ausgelöst wird, wird dann der endgültige Stand des *Reviewfeedbacks* in der Datenbank gespeichert und das Review mit *isFinished* als fertig markiert. Nach der Abgabe kann der Beurteilende das Review zwar noch anschauen, indem er es in der Kachel *Verteilte Reviews* auf der Übersichtsseite das Review öffnet, aber nicht mehr bearbeiten. Für diese Leseansicht wurde das Bearbeiten aller Felder und Skalen im CSS deaktiviert und gleichzeitig im Backend bei Reviews, die bereits das Attribut *isFinished* haben, sichergestellt, dass das Feedback nicht mehr veränderbar ist. Der Kursteilnehmer erkennt den Schreibschutz

[3] https://github.com/Craig-Creeger/react-likert-scale

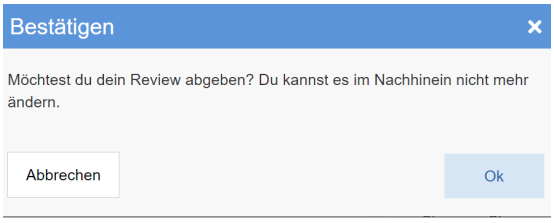

Abbildung 4.9: Dialog zum Bestätigen der Abgabe eines Reviews

gleichzeitig auch am Häkchen, das bei abgegebenen Reviews angezeigt wird. Zusätzlich visualisiert eine in Abbildung 4.10 dargestellte Fortschrittsanzeige in der Übersichtseite des Reports, wie viele von den mindestens geforderten Reviews bereits abgegeben worden sind. Eine neue Abgabe kann ein Teilnehmer nur dann

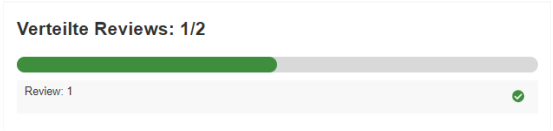

Abbildung 4.10: Fortschrittsanzeige der verteilten Reviews

zum Review anfordern, wenn er das vorherige Review abgegeben hat, wie dies bereits in Kapitel 3.7 erläutert wurde. Deshalb öffnet sich, falls ein Review bereits angefangen aber noch nicht fertig ist, mit dem Button *Nächstes Review*, das zwischengespeicherte Review. Ansonsten kann damit ein neues Review gestartet werden. Nur in der Situation, in der kein neues Review mehr möglich ist, weil alle anderen Kursteilnehmer schon von einem bewertet worden sind, verschwindet der Button.

4.3.4 Erhaltene Peerreviews

Sobald das Review abgegeben wurde, erscheint in der Übersichtsseite des Reports beim Autor der Abgabe das Review als neuer Listeneintrag in der Kachel *Erhaltene Reviews.* Wie in Abbildung 4.7 zu sehen, wird direkt die Gesamtbewertung des Feedbacks angezeigt. Dazu wird das Feedback mit dem Backendaufruf *getReviewFeedbackByIdIfFinshed* geladen. Durch einen Klick auf den Listeneintrag öffnet sich das Feedback parallel zur eigenen Abgabe, siehe Abbildung 4.11. Auch an dieser Stelle sind alle Felder nicht bearbeitbar, damit das Review nicht noch durch den Autor selbst verändert werden kann. Unten rechts in der Abbildung 4.11 ist der Button *Bewerte das Feedback* zu sehen. Durch einen Klick auf diesen öffnet sich in der Seitenleiste *Review* eine neue Seite, die in Abbildung 4.12 zu sehen ist. Hier kann der Autor der Abgabe das erhaltene Feedback zum einen mit der Sterneskala bewerten und einen Kommentar hinterlassen. Mit dem Button *Abgeben* kann er dieses Feedback abgeben und dann nicht mehr verändern. Der

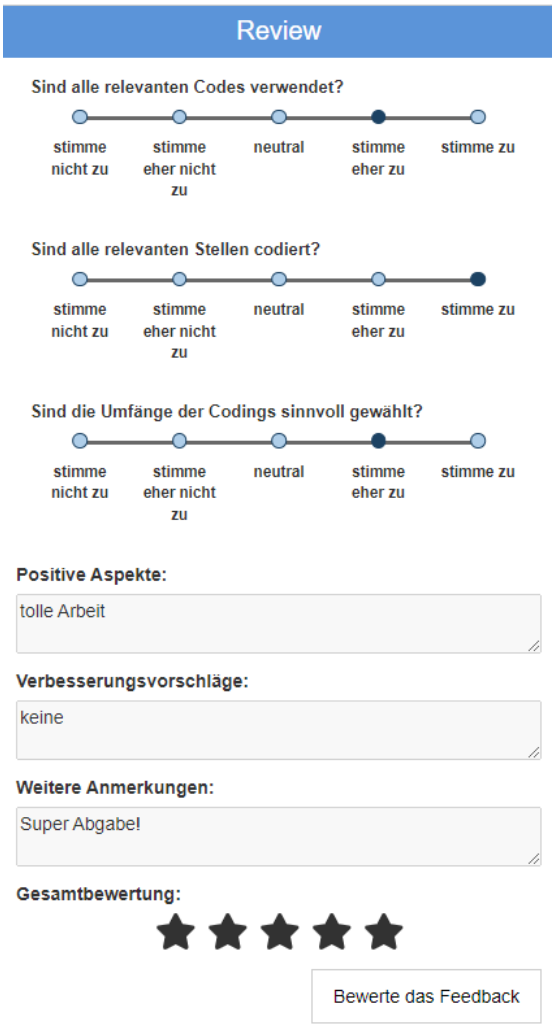

Abbildung 4.11: Erhaltenes Review

Autor des Reviews erhält die Bewertung, wobei ihm die Sternbewertung direkt im Listeneintrag des verteilten Reviews angezeigt wird und der Kommentar sichtbar wird, sobald er das Review öffnet.

4.4 Betreuerfeedback

Neben dem erhalten Peerreviews erscheinen in der Kachel *Erhaltene Reviews* auch Betreuerfeedbacks, falls ein Betreuer die Abgabe bewertet hat. Diese können die Kursteilnehmer daran erkennen, dass der Listeneintrag des Reviews *Betreuer Review* lautet - im Gegensatz zu *Review* bei erhaltenen Reviews von anderen Kursteilnehmern. Analog zu einem Peerreview kann dieses Review durch einen Klick

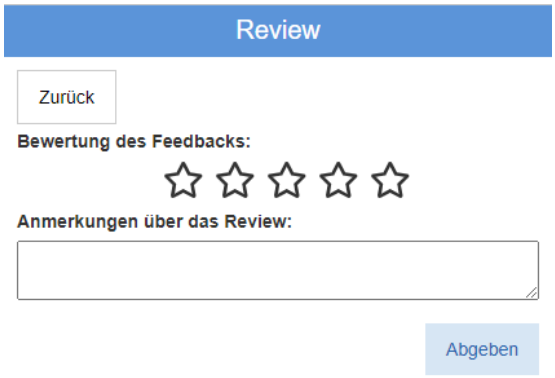

Abbildung 4.12: Bewertung des Reviews

auf den Eintrag geöffnet werden.
Als Kursleiter erscheint beim Öffnen eines Reports auf der Aufgabenkonfigurationsseite eine Übersichtsseite, die in Abbildung 4.13 zu sehen ist. Auf dieser Seite

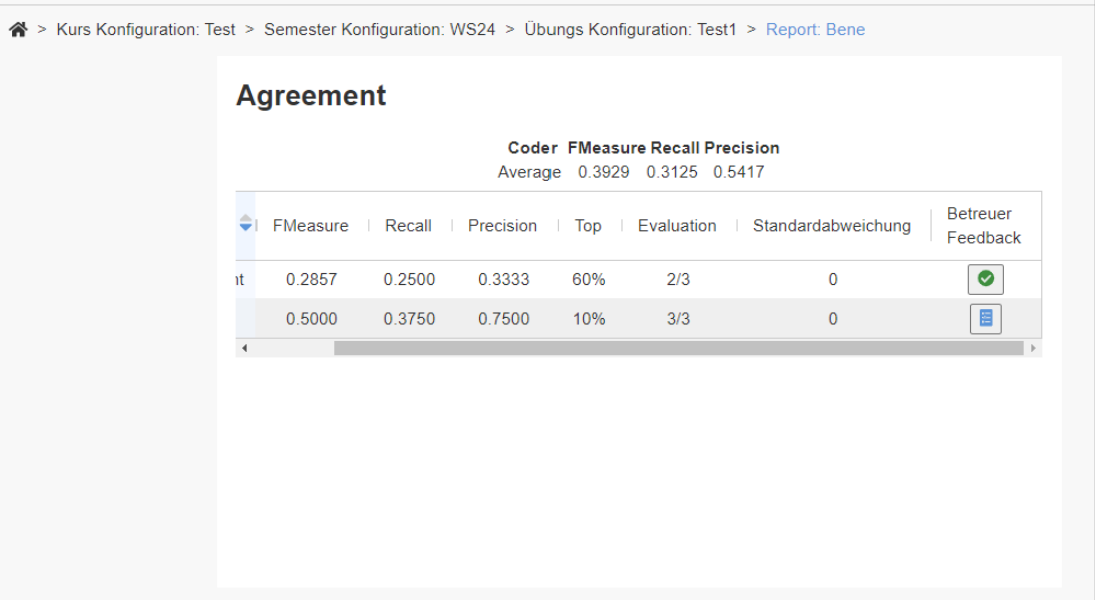

Abbildung 4.13: Übersichtsseite eines Reports aus Sicht eines Kursleiters

befindet sich die Kachel *Agreement* auf der die erreichten Werte des F-Maßes, Genauigkeit und Trefferquote aller Abgaben dieser Aufgabe zu finden sind. Außerdem gibt es eine Spalte mit der in Kapitel 3.10 erklärten Standardabweichung als Indiz dafür, ob ein Betreuerfeedback nötig ist. In der Spalte *Betreuerfeedback* befindet sich entweder ein Button mit einem Reviewsymbol, wenn die Abgabe noch nicht von einem Betreuer bewertet oder die Bewertung zwar angefangen aber noch nicht abgegeben worden ist, oder ein Button mit einem grünen Haken,

wenn die Abgabe bereits ein Betreuerfeedback erhalten hat. Durch das Klicken des Buttons öffnet sich dann dementsprechend ein neues oder ein bereits bearbeitetes Review. Ein neues Review wird dabei mit dem API-Aufruf *getResultWithNewInstructorReview* erstellt, der ein neues *Review*-Objekt erzeugt, bei dem das Attribut *isInstructorReview* gesetzt ist, und der die Id dieses Objekts in dem bewerteten *ExerciseResult* im Attribut *instructorReviewsReceived* speichert. Das Review an sich ist dabei genauso aufgebaut, wie das in Abbildung 4.8 dargestellte Peerreview.

5 Evaluation

Die Evaluation besteht aus zwei Teilen. Als Erstes wird im Kapitel 5.1 untersucht, ob die in Kapitel 2 definierten Anforderungen, erfüllt worden sind. Danach wird im Kapitel 5.2 die Benutzeroberfläche und Bedienbarkeit der neuen Funktionen mithilfe einer heuristischen Evaluation getestet.

5.1 Anforderungsüberprüfung

In diesem Kapitel wird die in Kapitel 2 definierte Liste der Anforderungen evaluiert, inwiefern die einzelnen Punkte erfüllt worden sind. Dazu werden zunächst die funktionalen Anforderungen betrachtet und im Anschluss die nichtfunktionalen.

5.1.1 Funktionale Anforderungen

Abgabe einer Hausaufgabe

FA 1.1: *Der Aufgabenbereich muss dem Kursteilnehmer die Möglichkeit geben, die Codings und bei Aufgabentypen mit Codebuch Erstellung auch das Codebuch abzugeben.*
Über den *Abgeben*-Button, der in Abbildung 4.1 zu sehen ist, kann der Kursteilnehmer seine Aufgabenbearbeitung abgeben. Der aktuelle Stand wird dann in der Datenbank in der Tabelle *Snapshot* gespeichert (siehe Kapitel 3.2)

FA 1.1 ist erfüllt

FA 1.2: *Der Aufgabenbereich sollte dem Kursteilnehmer die Möglichkeit geben, sich seine Abgaben jederzeit im Coding Editor anzuschauen.*
Auf der in Abbildung 4.1 sichtbaren Aufgabenübersichtsseite befindet sich eine Liste mit den Zeitpunkten der jeweiligen Abgaben. Diese lassen sich durch einen Klick auf den entsprechenden Listeneintrag öffnen. Die Abgabe wird dann im *Coding Editor* angezeigt.

FA 1.2 ist erfüllt

Automatisiertes Feedback

FA 2.1.1*Das automatisierte Feedback sollte aus der letzten Abgabe des Kursteilnehmers erstellt werden. Falls jedoch keine Abgabe getätigt worden ist, soll eine Abgabe vom aktuellen Bearbeitungsstand automatisiert erstellt werden.*
Bei der Reporterstellung, die beim Aufgabentyp *Codebuch Verwendung* manuell durch den Kursleiter und bei *Codebuch Erstellung* automatisch beim Überschreiten des Abgabezeitpunkts gestartet wird, wird das *Intercoder Agreement* basierend auf einem *Snapshot* berechnet. Falls noch keine Abgabe gemacht wurde, wird zu diesem Zeitpunkt vom aktuellen Bearbeitungsstand der Aufgabe ein *Snapshot* erzeugt, das dann dem automatisierten Feedback zugrunde liegt.

FA 2.1.1 ist erfüllt

FA 2.1.2: *Die Agreement Maps müssen den Grad an Abweichung zur Musterlösung visualisieren.*
In den *Agreement Maps* werden die Anzahl der RP, FP und FN dargestellt - also sowohl die Anzahl der Codes, die mit der Musterlösung übereinstimmen als auch diejenigen, die abweichen. Diese werden in einer Seitenleiste in Form von drei Balken dargestellt, in denen die einzelnen Paragrafen farblich markiert sind. Je dunkler der Teil des Balkens, umso größer die Abweichung. Die Abbildung 4.3 zeigt diese Seitenleiste.

FA 2.1.2 ist erfüllt

FA 2.1.3: *Die Einfärbungen der Agreement Maps sollte dem Kursteilnehmer die Möglichkeit geben, die Absätze seiner Abgaben mit den größten Abweichungen herauszufinden und die Einfärbungen sollen relativ zur Leistung innerhalb der eigenen Abgabe sein.*
Die Absätze mit der größten Abweichung sind in der *Agreement Map* dunkelrot gefärbt. Die Einteilung der Absätze zu einer der fünf roten Farbe wird dabei linear anhand der Anzahl der Fehler innerhalb der betreffenden Abgabe vorgenommen und ist deshalb nur relativ zu seiner eigenen Abgabe.

FA 2.1.3 ist erfüllt

FA 2.1.4: *Die Nutzeroberfläche sollte dem Kursteilnehmer eine Hilfestellung zur Terminologie bieten.*
Die Seitenleiste hat einen Info Icon, der in Abbildung 4.3 zu sehen ist. Beim Klick auf diesen öffnet sich ein Informationstext, der die Einfärbungen erklärt.

FA 2.1.4 ist erfüllt

FA 2.1.5: *Die Agreement Maps sollten dem Kursteilnehmer die Möglichkeit geben, mit einer Mehrfachauswahl die RP, FP und die FN nebeneinander zu sehen.* Neben dem Info Icon befindet sich in der Seitenleiste ein DropDown Menü, in dem die einzelnen *Agreement Maps* mit einem Haken ausgewählt werden können. Alle angewählten Spalten werden dann parallel in der Seitenleiste angezeigt.

FA 2.1.5 ist erfüllt

FA 2.1.6: *Eine Übersichtsseite muss dem Kursteilnehmer die Möglichkeit geben, einen Überblick zu bekommen, welche Codes in der Abgabe abweichend von der Musterlösung verwendet wurden. Dazu soll es einen tabellarischen Überblick mit den F-Maß der einzelnen Codes über alle kodierten Dokumente der Abgabe geben.* Zusätzlich zum Reiter *Agreement pro Absatz* mit den *Agreement Maps* gibt es den Reiter *Agreement pro Code*, der in Abbildung 4.4 dargestellt ist. Hier werden die Werte für F-Maß, Genauigkeit und Trefferquote einzelner Codes über alle kodierten Dokumente aggregiert in einer Tabelle dargestellt.

FA 2.1.6 ist erfüllt

FA 2.2.1 *Die Übersichtsseite der Aufgabenbewertung sollte dem Kursteilnehmer die Möglichkeit geben, zu sehen, in welchem Quantil innerhalb des Kurses seine Abgabe war. Dabei soll die Auswertung auf Zehntel genau ausgegeben werden. Das Ergebnis sollte dem Kursteilnehmer grafisch durch eine Anzeige visualisiert werden.*
Diese Anforderung wird mithilfe einer Google Charts Anzeige[1] umgesetzt. Diese zeigt das erreichte Perzentil mit einer analogen Anzeigenadel an (zu sehen in Abbildung 4.5). Dazu werden alle Zehntel-Perzentil Stufen der erreichten F-Maße über den Kurs hinweg kalkuliert und den erreichten Wert der Abgabe in diese einsortiert.

FA 2.2.1 ist erfüllt

FA 2.2.2 *Der Dialog zum Erstellen einer neuen Aufgabe muss dem Kursleiter die Möglichkeit geben, ein Bewertungsschema festzulegen. Dazu legt er die Grenzen des F-Maßes für eine minimale, durchschnittliche und außergewöhnliche Leistung fest.*
Die geforderten Felder wurden dem Dialog, der in der Abbildung 4.6 dargestellt ist, hinzugefügt. Zwar ist das Bewertungsschema initial mit den Werten 0.1, 0.2 und 0.4 gefüllt, kann aber auch individuell vom Kursleiter angepasst werden.

FA 2.2.2 ist erfüllt

[1] https://developers.google.com/chart/interactive/docs/gallery/gauge?hl=de

FA 2.2.3 *Die Felder zum Anlegen eines Bewertungsschemas sollten durch ein Info Icon mit aufklappbaren Informationen erklärt werden.*
Der Info Icon wurde dem Dialog hinzugefügt. Beim Klick auf diesen öffnet sich ein Informationstext, der auf den Artikel von Kaufmann et al. (2023) verweist.

FA 2.2.3 ist erfüllt

FA 2.2.4 *Die Übersichtsseite der Aufgabenbewertung muss dem Kursteilnehmer die Möglichkeit geben, grafisch mit einer Anzeige die Einordnung in das Bewertungsschema nachzuvollziehen.*
Analog zur Visualisierung des erreichten Perzentils wird auch die Einordnung des erreichten F-Maßes mit einer Google Charts Anzeige[2] dargestellt (siehe Abbildung 4.5).

FA 2.2.4 ist erfüllt

Peerreview

FA 3.1 *Der Dialog zum Erstellen einer neuen Aufgabe muss dem Kursleiter die Möglichkeit geben, anzugeben, ob ein Peerreview durchgeführt werden soll und wenn ja wie viele Reviews mindestens pro Teilnehmer durchgeführt werden müssen.*
Im Dialog (Abbildung 4.6) wurde eine Checkbox für das Peerreview hinzugefügt, bei dessen Auswahl ein zusätzliches Feld erscheint, in dem die Mindestanzahl festgelegt werden kann.

FA 3.1 ist erfüllt

FA 3.2 *Die Übersichtsseite der Aufgabenbewertung muss dem Kursteilnehmer die Möglichkeit geben, eine fremde Abgabe zum Reviewen anzufordern.*
Durch den Button *Nächstes Review*, der auf der Übersichtseite des Reports in der Kachel *Verteilte Reviews* zu finden ist, wird die Abgabe eines anderen Kursteilnehmers im *Coding Editor* mit einer Seitenleiste zum Ausfüllen des Reviews geöffnet. Die Zuordnung von Peerreviews wird in Kapitel 3.7 erläutert.

FA 3.2 ist erfüllt

FA 3.3 *Die Reviewseite sollte dem Reviewer die Möglichkeit geben, mit Likertskalen, offenen Fragen und einer Gesamtbewertungsskala die zu beurteilende Abgabe zu bewerten.*
Wie in Abbildung 4.8 zu sehen, ist die Eingabe-Maske zum Review wie gefordert aufgebaut. Am Anfang gibt es für den Aufgabentyp spezifische Likertfragen, dann offene Fragen zu positiven Aspekten, Verbesserungsvorschlägen und weiteren Anmerkungen und zum Schluss eine Gesamtbewertung mit einer Sterneskala.

[2] https://developers.google.com/chart/interactive/docs/gallery/gauge?hl=de

FA 3.3 ist erfüllt

FA 3.4 *Die Reviewseite sollte dem Reviewer die Möglichkeit geben, parallel zum Review die Abgabe im Coding Editor einsehen zu können.*
Das Review ist in einer Seitenleiste rechts platziert, die parallel zur Abgabe im *Coding Editor* angezeigt wird (siehe Abbildung 4.8).

FA 3.4 ist erfüllt

FA 3.5 *Die Reviewseite sollte dem Reviewer die Möglichkeit geben, das Review zwischenzuspeichern und später fortzuführen. Solange ein Review nicht final abgegeben worden ist, ist es für den Autor der Abgabe nicht einsichtig. Gleichzeitig kann der Reviewer auch noch kein neues Review anfangen.*
Diese Anforderung wurde zum einen durch den Speicher Button unten in der Seitenleiste des Reviews sichergestellt, mit dem der aktuelle Stand des Reviews zwischengespeichert wird und zum anderen durch die Logik hinter dem Button *Nächstes Review*. Wenn ein noch nicht abgeschlossenes Review vorhanden ist, wird dieses geöffnet. Eine neue Abgabe wird nur angefordert, wenn alle offenen Reviews abgegeben worden sind.

FA 3.5 ist erfüllt

FA 3.6 *Die Reviewseite sollte dem Reviewer die Möglichkeit geben, das Review abzugeben und dies durch einen zusätzlichen Dialog zu bestätigen.*
Neben dem Speichern Button befindet sich der Button *Abgeben* bei dem sich, wenn alle Felder des Reviews ausgefüllt sind, ein Dialog zum Bestätigen öffnet. Das Review wird erst nach der Bestätigung als beendet markiert. Der Dialog ist in Abbildung 4.9 dargestellt.

FA 3.6 ist erfüllt

FA 3.7 *Die Übersichtsseite der Aufgabenbewertung muss dem Reviewer die Möglichkeit geben, die von ihm gegebenen Urteile zu seinen Reviews anzuschauen. Dabei muss aber ein Bearbeiten nach Abgabe verhindert werden und der User darüber aufgeklärt werden, dass das Review bereits abgegeben worden ist.*
Bereits verteilte Reviews findet der Benutzer in der zugehörigen Kachel auf der Übersichtsseite des Reports. Ein grüner Haken am Ende des Eintrags visualisiert, dass das Review bereits abgegeben worden ist. Durch einen Klick auf diesen öffnet sich das ausgefüllte Review im *Coding Editor*. Anstelle des Speichern und Abgeben Buttons befindet sich jetzt jedoch ein Haken mit dem Text *Review ist bereits abgegeben* am unteren Ende des Reviews. Auch das Backend stellt sicher, dass ein bereits abgegebenes Review nicht weiter bearbeitet werden kann.

FA 3.7 ist erfüllt

FA 3.8 *Die Übersichtsseite der Aufgabenbewertung muss dem Kursteilnehmer die Möglichkeit geben, Reviews zu seinen Abgaben anzuschauen.*
Die erhaltenen Reviews findet der Kursteilnehmer in der zugehörigen Kachel auf der Übersichtsseite des Reports, welche in Abbildung 4.7 abgebildet ist. Hier wird ihm direkt die Gesamtbewertung angezeigt. Durch Anklicken des Reviews öffnet sich dieses im *Coding Editor* parallel zur eigenen Abgabe, sodass dann das ausgefüllte Review angeschaut werden kann.

FA 3.8 ist erfüllt

FA 3.9 *Die Reviewseite sollte dem Kursteilnehmer, der das Review erhalten hat, die Möglichkeit geben, das erhaltene Review zu bewerten.*
Beim Öffnen eines erhaltenen Reviews kann der Autor der Abgabe das Review bewerten. Er kann sowohl eine Sternbewertung als auch einen Kommentar zum Review abgeben, wie in Abbildung 4.12 zu sehen ist.

FA 3.9 ist erfüllt

Feedback Kursleiter

FA 4.1 *Der Dialog zum Erstellen einer neuen Aufgabe muss dem Kursersteller die Möglichkeit geben, ein Bewertungsschema für das Betreuerfeedback festzulegen.*
Aufgrund mangelnder Zeit und niedrigere Priorität konnte die Anforderung der unterschiedlich konfigurierbaren Bewertungsschemen nicht mehr umgesetzt werden. Der Betreuer kann die Abgaben nur mit dem Schema des Peerreviews bewerten.

FA 4.1 ist nicht erfüllt

FA 4.2 *Die Reportseite sollte dem Kursleiter die Möglichkeit geben, Feedback zu einzelnen Abgaben zu geben.*
Die Anforderung wurde erfüllt, da man als Kursleiter bei Öffnung des Reports eine Liste aller Abgaben findet. In der letzten Spalte *Betreuerfeedback*, zu sehen in Abbildung 4.13, befindet sich ein Button, mit dem sich bei Klick die Abgabe mitsamt des Reviews öffnet.

FA 4.2 ist erfüllt

FA 4.3 *Die Reportseite sollte dem Kursleiter die Möglichkeit geben, die Standardabweichung der Gesamtbewertung erhaltener Peerreviews von einzelnen Abgaben zu sehen.*
Wie in Kapitel 3.10 erläutert, wird die Standardabweichung bei jeder Abgabe eines Peerreviews berechnet und im *ExerciseResult* gespeichert. Der Kursleiter sieht die Abweichung dann in der Tabelle aller Abgaben eines Reports als neue Spalte.

FA 4.3 ist erfüllt

FA 4.4 *Die Übersichtsseite der Aufgabenbewertung sollte den Kursteilnehmern die Möglichkeit geben, das Feedback des Kursleiters einzusehen.* Die erhaltenen Betreuerfeedbacks finden die Kursteilnehmer in der Kachel *Erhaltene Reviews* in der Übersichtsseite eines Reports.

FA 4.4 ist erfüllt

5.1.2 Nichtfunktionale Anforderungen

Stabilität

NFA 1.1 *Die Implementierung darf keine vorhandenen Funktionalitäten beeinträchtigen.*
Diese Anforderung ist erfüllt, da, wie bisher, der Kursleiter einen Report zu einer Aufgabe erstellen kann. Auch das bereits vorher existierende Feedback mit den erreichten Werten für F-Maß, Genauigkeit und Trefferquote wird dem Kursteilnehmer weiterhin angezeigt.

NFA 1.1 ist erfüllt

NFA 1.2 *Der Rollout einer neuen Funktion sollte über ein Feature Flag für einen Nutzer verborgen bleiben, bis die Funktionalität angemessen umgesetzt ist.*
Bis die *Agreement Map* vollständig funktionsfähig war, wurde im *Coding Editor* das Flag *showSideBarAgreement* auf *false* gesetzt, um den Nutzer die Seitenleiste nicht anzuzeigen.

NFA 1.2 ist erfüllt

Performanz

NFA 2.1 *Längere und aufwändigere Rechenaufwände sollen in das Backend verlagert werden, um die Performance im Frontend zu beschleunigen.*
Um diese Anforderung zu erfüllen, wurde versucht möglichst viele Aufgaben bereits im Backend abzuarbeiten. Dies ist unter anderem daran zu erkennen, dass sowohl die Berechnung der Standardabweichung als auch die Sortierung der F-Maße eines Reports im Backend implementiert wurden.

NFA 2.1 ist erfüllt

Kompatibilität

NFA 3.1 *Die Implementierung muss kompatibel mit den bereits verwendeten Technologien sein.*

Die geforderten Implementierungen wurden im Frontend mit dem bereits verwendeten Technologien React und Javascript und im Backend Java und der Google App Engine vorgenommen.

NFA 3.1 ist erfüllt

Benutzbarkeit

NFA 4.1 *Die Texte in der Benutzeroberfläche müssen alle eine deutsche Übersetzung haben.*
Mithilfe der Bibliothek *react-intl*[3] können Texte mit *FormattedMessage* so implementiert werden, dass sie sowohl in Englisch als auch in Deutsch existieren. Die Abbildungen innerhalb der Bachelorarbeit wurden von der deutschen Website gemacht, sodass man in ihnen die Übersetzungen sieht.

NFA 4.1 ist erfüllt

NFA 4.2 *Das Design muss das vorhandene Farbschema von QDAcity übernehmen.*
Für die Implementierung steht eine vorausgewählte Farbpalette in der Klasse *Theme* zur Verfügung. Für die Umsetzung der Anforderungen wurden ausschließlich die Farben dieser Palette verwendet. Auch die Anzeigenadeln und Likert-Skalen wurden mit diesen Farben konfiguriert.

NFA 4.2 ist erfüllt

NFA 4.3 *Das System muss sicherstellen, dass der Reviewer das Review ausgefüllt hat, bevor er es abgeben kann.*
Diese Anforderung wird mithilfe des *formValidator* sichergestellt, der überprüft, dass alle Felder ausgefüllt wurden.

NFA 4.3 ist erfüllt

NFA 4.4 *Die Benutzeroberfläche muss den Web Content Accessibility Guidelines (WCAG) 2.1 entsprechen.*
Um die Erfüllung dieser Anforderung zu überprüfen, wurden die entsprechenden Seiten mithilfe der Chrome-Erweiterung *Lighthouse*[4] auf die Barrierefreiheit getestet. Dabei sind unter anderem die Verletzungen der Regeln nach dem WCAG 2.1 (A) Standard zum Buttonname[5], Linkname[6] und der Verwendung von Listen[7] festgestellt worden. Diese Fehler konnten auf allen Seiten inklusive der Kopfzeile von QDAcity gefunden werden, sodass die Website nicht dem *Web Content Accessibility Guidelines* entspricht.

[3] https://www.npmjs.com/package/react-intl?activeTab=readme
[4] https://chromewebstore.google.com/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk
[5] https://dequeuniversity.com/rules/axe/4.8/button-name
[6] https://dequeuniversity.com/rules/axe/4.8/link-name
[7] https://dequeuniversity.com/rules/axe/4.8/list

NFA 4.4 ist nicht erfüllt

NFA 4.5 *Das System muss sicherstellen, dass das eingegebene Bewertungsschema geeignet ist.*
Mithilfe des in QDAcity implementierten *formValidator* wird sichergestellt, dass das eingegebene F-Maß für eine minimale Leistung größer als 0 und kleiner als der Wert für eine durchschnittliche Leistung ist, dass der Wert für eine durchschnittliche Leistung zwischen dem Wert für eine minimale und einer außergewöhnlichen liegt und, dass der Wert für eine außergewöhnliche Leistung größer als der einer durchschnittlichen aber kleiner als 1 ist. Ein Evaluationsschema, das diese Vorgaben erfüllt, ist für die Bewertung geeignet.

NFA 4.5 ist erfüllt

Sicherheit

NFA 5.1 *Alle Endpunkte müssen die Autorisierung prüfen, bevor sie die Anfrage durchführen können.*
Die neuen Endpunkte in der Klasse *ReviewEndpoint* beinhalten eine Überprüfung der Autorisierung durch die neu implementierten Methoden *isAuthorizedToMakeInstructorReview*, *isAuthorizedToMakeNextReviewFeedback*, *isAuthorizedToUpdateReviewFeedback*, *isAuthorizedToSetFeedbackOfFeedback* und *isAuthorizedToGetReviewFeedback*. Der neue Endpunkt *addSnapshotToExerciseProject* im *ExerciseEndpoint* und die beiden Endpunkte *getAgreementMapsForProject* und *getAgreementPerCodeForProject* im *AgreementMapEndpoint* prüfen durch den Methodenaufruf *throwIfUserIsNotAuthorizedForTermCourse*, ob der Benutzer ein Kursteilnehmer und daher autorisiert ist.

NFA 5.1 ist erfüllt

NFA 5.2 *Das System muss sicherstellen, dass der Reviewer nicht die Identität des Autors der Abgabe herausfinden kann.*
NFA 5.3 *Das System muss sicherstellen, dass der Kursteilnehmer nicht die Identität des Reviewers herausfinden kann.*
Wie in Kapitel 3.5 erläutert, wurden im Hinblick auf diese Anforderungen die zwei separaten Klassen *Review* und *Reviewfeedback* eingeführt. Die Metadaten, also auch die Zuordnung des Reviews zu Autor und Reviewer, sind im *Review*-Objekt gespeichert, welches nicht im Frontend angefordert wird. Dadurch ist sichergestellt, dass das Review anonym durchgeführt wird. Einzig die Ids des *Review*-Objekts erhält sowohl der Autor der Abgabe als auch der Reviewer.

NFA 5.2 und 5.3 sind erfüllt

NFA 5.4 *Das System muss sicherstellen, dass der Kursteilnehmer nicht die Musterlösung rekonstruieren kann.*
Diese Anforderung wurde zum einen durch die Einführung der *Agreement Maps*

erfüllt. Diese aggregieren die Anzahl an RP, FP und FN über den Absatz hinweg, sodass keine genaue Position der Codes verraten wird. Auch wird durch die Balken nicht angezeigt, welche Codes genau richtig und welche falsch gesetzt werden. Des Weiteren hat auch die *Agreement pro Code* Seite zu wenig Aussagekraft, um eine Musterlösung rekonstruieren zu können, da dort zwar die Übereinstimmung der Codes dargestellt wird, aber nur über das ganze Dokument hinweg.

NFA 5.4 ist erfüllt

Wartbarkeit

NFA 6.1 *Alle neuen Endpunkte müssen eine Testabdeckung durch Unittests von mindestens 90% haben.*
Im Anhang ist dazu eine genaue Auflistung der einzelnen Methoden und ihrer Testabdeckung zu finden, die mit dem Tool *Jacoco*[8] gemessen wurden. Diese bestätigen eine Testabdeckung von 91% der neu hinzugefügten Endpunkte.

NFA 6.1 ist erfüllt

NFA 6.2 *Alle neuen Funktionen müssen mit Akzeptanztests geprüft werden.*
Aufgrund der zeitlichen Beschränkung einer Bachelorarbeit konnte diese Anforderung nicht erfüllt worden. Im Rahmen der Bachelorarbeit sind zwar einige neue Akzeptanztests entstanden, jedoch testen sie nicht alle neuen Funktionen ab.

NFA 6.2 ist nicht erfüllt

NFA 6.3 *Das System muss für neue Feedbackarten erweiterbar sein.*
Diese Anforderung wurde bei der Architekturentscheidung zu einer Vererbung des *ReviewFeedbacks*, welche in dem Klassendiagramm in der Abbildung 3.3 dargestellt ist, berücksichtigt. So kann diese leicht durch anders aufgebaute Feedbacks, die zum Beispiel für den Kursleiter entworfen werden, ergänzt werden.

NFA 6.3 ist erfüllt

NFA 6.4 *Das System muss sicherstellen, dass unterschiedliche Zuständigkeiten auch in getrennten Klassen implementiert werden.*
Die Umsetzung dieser Anforderung ist vorrangig beim Aufbau des *Backends* des Reviews zu sehen. Hier werden die einzelnen Zuständigkeiten in die Klassen *ReviewEndpoint*, *Authorization*, *ReviewController*, *ReviewDao* und *ReviewFeedbackDao* aufgeteilt, was in Kapitel 3.6 erläutert wird.

NFA 6.3 ist erfüllt

[8] https://www.jacoco.org/jacoco/trunk/doc/maven.html

5.2 Heuristische Evaluation

Um die neu implementierten Funktionen auch in der Praxis zu testen und um Probleme im Design zu finden, wurde eine heuristische Evaluation nach Nielsen (1994) durchgeführt. Demnach testen zwischen drei bis fünf Gutachter die neue Benutzeroberfläche anhand vorher festgelegten Heuristiken. (Nielsen, 1994)

5.2.1 Aufbau der heuristischen Evaluation

Um bestmögliche Ergebnisse zu erhalten, wurden die Evaluationen unabhängig voneinander durchgeführt, sodass die Gutachter sich nicht gegenseitig beeinflussen konnten. (Moran & Gordon, 2023) Zusätzlich wurden die Evaluationen von einem Beobachter durchgeführt und von diesem protokolliert, um zum einen die Handhabung der Nutzer zu sehen und um zum anderen Auffälligkeiten direkt besprechen zu können.

Gutachter

Für die Evaluation wurden drei Personen gesucht, die bereit waren, diese durchzuführen. Als besonders geeignet haben sich dafür Studenten herausgestellt, die den Kurs NYT bereits absolviert haben. Zum einen hat dies den Vorteil, dass sie sich besonders gut in die Situation der Studenten hineinversetzen können, in der sie das Feedback erhalten. Zum anderen sind die Studenten dadurch auch bereits vertraut mit der Webanwendung QDAcity und dem Kodieren von Texten. Ein weiterer Vorteil ist, dass sie das neue Feedback mit dem früheren vergleichen können. Für die Durchführung wurden drei Studenten gefunden, die vor zwei bzw. drei Jahren den Kurs belegt hatten. Bei der Auswahl wurde auf möglichst hohe Heterogenität geachtet, um ein möglichst breites Feedback zu erhalten. So befinden sich unter den Gutachtern sowohl eine weibliche als auch zwei männliche Personen, zwei im Bachelorstudiengang Informatik und eine Person im Masterstudiengang International Information System. Einer der Studenten hatte zwar den Kurs frühzeitig abgebrochen, da er jedoch eine Aufgabe in QDAcity bearbeitet hatte, war sein Feedback trotzdem sehr erkenntnisreich.

Liste an Heuristiken

Als Liste von Heuristiken wurden die Design-Prinzipien von Norman (2016) ausgewählt, die wie folgt lauten[9]:

- **Discoverability**. Dieses Prinzip schreibt vor, dass der Nutzer sowohl seine Handlungsmöglichkeiten als auch den aktuellen Status der Anwendung kennen muss.
- **Feedback**. Nach diesem Prinzip sollte es genügend Informationen bezüglich der Folgen von Aktionen und des aktuellen Status der Anwendung geben. Auch nach der Ausführung einer Aktion ist der neue Status leicht zu bestimmen.
- **Conceptual Model**. Das Layout sollte dem Nutzer alle notwendigen Information präsentieren, sodass das Produkt verstanden wird und ein Gefühl der Kontrolle entsteht.
- **Affordances**. Dieses Prinzip ist erfüllt, wenn dem Nutzer klar ist, welche Aktionen möglich sind.
- **Signifiers**. Durch eindeutige Kennzeichen weiß der Nutzer, wo und wie er die verfügbaren Aktionen auslösen kann.
- **Mapping**. Für dieses Prinzip muss die Beziehung zwischen den Steuerelementen und deren Funktion verständlich sein.
- **Constraints**. Durch physische, logische, semantische und kulturelle Einschränkungen werden die Handlungen vereinfacht und die Interaktion verbessert.

Die ausgewählten Studenten hatten diese Liste mit der Erklärung bereits vorab erhalten, damit sie sich damit vertraut machen konnten und bei der Durchführung auf diese Aspekte achten konnten.

Ablauf der heuristischen Evaluation

Die Ausgangssituation wurde für alle drei Studenten gleich vorbereitet. Sie sollten sich in die Situation hineinversetzen, dass sie eine Aufgabe bearbeitet hatten, der Kursleiter einen Report erstellt hat und sie nun ihr Feedback anschauen wollen. Dazu haben die Gutachter Login-Daten zu einem Profil in QDAcity bekommen, dessen Nutzer einen neuen Report zu einer Aufgabe bekommen hatte. Mit diesem Profil sollten sie die folgenden Aufgaben erfüllen:

[9] https://www.webdesign-essentials.ch/kategorien/human-centered-design

- den Report öffnen und das neue Dashboard begutachten
- das Agreement pro Absatz analysieren
- das Agreement pro Code anschauen
- das erhaltene Betreuerfeedback betrachten und bewerten
- ein Peerreview ausfüllen und abgeben

Zusätzlich sollten die Gutachter die folgenden Fragen beantworten:

- Wie hast du damals, als du NYT belegt hast, das Feedback wahrgenommen?
- Hätte dir das neue, erweiterte Feedback geholfen? Was ist aus deiner Sicht besonders hilfreich?
- Siehst du Probleme bei dem neuen Feedback?

Dabei wurde die erste Frage direkt am Anfang vor der Durchführung der Aufgaben gestellt, um die Gutachter in die Situation zurückzuversetzen, die sie bei Erhalt eines Reports hatten. Die anderen beiden Fragen wurden am Ende der Evaluation gestellt.

5.2.2 Ergebnis der heuristischen Evaluationen

In diesem Abschnitt werden zunächst die in den durchgeführten Evaluationen gefundenen Auffälligkeiten, die nicht den Heuristiken entsprechen, vorgestellt. Diese wurden über alle Gutachten hinweg aggregiert und den einzelnen Prinzipien zugeordnet. Im Anschluss werden die Antworten auf die gestellten Fragen begutachtet. Die einzelnen Protokolle der Evaluationen befinden sich im Anhang. Die folgenden Probleme wurden angesprochen:

- **Discoverability**:
  Im Hinblick auf diesen Punkt ist die Ansicht *Agreement pro Absatz* negativ aufgefallen. Beim Öffnen der Ansicht war einem Gutachter die Aussagekraft der *Agreement Map* nicht klar und er erkannte nicht, wie man damit umzugehen hatte.
- **Feedback**:
  Dieses Prinzip wurde nach Ansicht aller Gutachter vollständig erfüllt.
- **Conceptual Model**:
  Alle Gutachter waren sich hierbei einig, dass ihnen weitere Informationen zur *Agreement Map* fehlten. Sie hätten sich darüber gefreut, wenn der Infotext näher darauf eingegangen wäre, was die Balken über die Qualität ihrer Abgabe aussagen.

Außerdem ist aufgefallen, dass die Farbwahl ungünstig ist, da ein Student mit einer Rot-Grün-Schwäche nicht zwischen der Grünschattierung der RP und der Rotschattierung der FN und FP unterscheiden konnte. Dadurch war es für ihn nicht verständlich, dass bei RP diejenigen Absätze besser sind, die einen dunkleren Grünton haben, bei den anderen Balken aber ein dunklerer Rotton, schlechter kodierte Absätze markieren.
Zusätzlich hätte sich ein Gutachter Informationstexte über die Fachbegriffe *FMeasure, Recall* und *Precision* gewünscht.

- **Affordances**:
  Unter diesem Prinzip ist das Problem einzuordnen, dass der Info Icon in der *Agreement Map* nicht gefunden wurde und so nicht erkannt wurde, dass zusätzliche Informationen verfügbar sind.

- **Signifiers**:
  Zum einen ist von einem Studenten angemerkt worden, dass die Darstellung des *Agreements* als Tabelle sinnlos erscheint, da nur eine einzige Zeile mit den erreichten Werten angezeigt wird. Die Tabellendarstellung suggeriert jedoch, dass dort weitere Einträge sein sollten. Die enthaltenen Werte hätte man auch ähnlich wie das erreichte Perzentil und Einordnung in das Bewertungsschema grafisch visualisieren können. Zudem wurde auch kritisiert, dass die Zeile auswählbar ist, dies jedoch kein Effekt hat.

- **Mapping**:
  Hinsichtlich dieses Prinzips wurde auf der Seite *Agreement pro Absatz* angesprochen, dass man die Beziehung zwischen den Absätzen im Text und der Einfärbung der *Agreement Map* verbessern sollte. Dabei wurde vorgeschlagen, dass man die einzelnen Absätze in den Balken durch schwarze Striche unterteilt. Dadurch würde der Beginn eines neuen Absatzes auch dann erkenntlich sein, wenn diese die gleiche Einfärbung haben.

- **Constraints**:
  Hierzu wurde kritisiert, dass man die Einschränkung, dass alle Felder ausgefüllt sein müssen, um ein Peerreview abgeben zu können, besser visualisieren sollte. Dazu wurde vorgeschlagen, dass die Freitexteingaben als Pflichtfelder mit einem Stern markiert werden sollten.

Zusätzlich zu den oben genannten Auffälligkeiten, die bei der Bearbeitung der Aufgaben aufgefallen sind, waren auch die Antworten auf die gestellten Fragen sehr aufschlussreich.
Auf die Frage, wie sie bei der ursprünglichen Kursteilnahme das Feedback wahrgenommen haben, waren sich alle drei Gutachter einig, dass dieses nicht wirklich vorhanden war. Einer erzählte davon, dass er viel Zeit in den Kurs investiert hatte, dafür aber nie Rückmeldung bekommen hatte und sich so ständig Sorgen um die Note gemacht hatte. Dadurch ist der Kurs als sehr belastend wahrgenommen

worden. Außerdem wurde kritisiert, dass man sich nicht verbessern konnte, weil nie aufgezeigt wurde, wo genau die Fehler lagen, sodass der Fähigkeitserwerb innerhalb des Kurses insgesamt infrage gestellt wurde. Damit Bestätigen diese Aussagen die Zitate aus dem Evaluationsbögen, die in der Einleitung wiedergegeben wurden und die Motivation der Arbeit darstellen.
Die Antworten auf die zweite nach dem Testen der neuen Funktionen gestellte Frage, ob ihnen das neue, erweiterte Feedback geholfen hätte und was aus ihrer Sicht besonders hilfreich gewesen wäre, waren äußerst positiv. Alle Gutachter bestätigen, dass sie sich dieses Feedback schon bei ihrer ursprünglichen Teilnahme am Kurs NYT gewünscht hätten. Außerdem stellt sich heraus, dass besonders die Vielfalt der Feedbackarten von Vorteil ist. So war für einen Gutachter besonders die Einordnung in den Kurs und in das Evaluationsschema hilfreich, da er dadurch seine Leistung im Vergleich zu anderen hätte einordnen können und sich dadurch weniger Sorgen um die Note gemacht hätte. Ein anderer Gutachter fand die *Agreement Map* am besten, da er dadurch erfahren hätte, wo seine Fehler lagen und daraus für die nächste Abgabe hätte dazulernen können. Der dritte Gutachter lobte vor allem die Möglichkeit durch das Peerreview andere Abgabe einsehen zu können und dadurch herauszufinden, wie andere Kursteilnehmer die Aufgabe gelöst haben. Dieses Ergebnis zeigt, dass die Gutachter verschiedene Feedbackarten intensiv genutzt hätten und bestätigt auch, dass sie alle mit Hinblick auf die individuellen Bedürfnisse einzelner Kursteilnehmer sinnvoll sind.
Auf die letzte Frage, ob sie Probleme bei dem neuen Feedback sehen, wurde angesprochen, dass ein qualitatives Peerfeedback zusätzlichen Zeitaufwand bedeutet. Da der Aufwand für eine erfolgreiche Teilnahme am Kurs schon als hoch eingeschätzt wurde, liegt die Befürchtung nahe, dass sich kaum Zeit für die Reviews genommen wird und der Autor der Abgabe kein hilfreiches Review bekommt. Dies wurde auch mit der zweiten Befürchtung, dass die Reviews nur durchgeführt werden, um möglichst viele Abgaben zu sehen und nicht, um konstruktives Feedback abzugeben, bestärkt. Außerdem befürchtete ein Gutachter, dass die Einordnung ins Perzentil eventuell auch demotivierend sein könnte, wenn man zur unteren Hälfte des Kurses gehört. So könnte eine schlechte Einordnung innerhalb des Kurses trotz guter Bewertung dazu führen, dass Teilnehmer den Kurs abbrechen, da sie eine schlechte Note befürchten und sie deshalb keine Zeit mehr in den Kurs investieren wollen.

5.3 Fazit der Evaluation

Zusammenfassend hat die heuristische Evaluation gezeigt, dass der Mehrwert des neuen Feedbacksystems sehr groß ist und sie hat auch den Bedarf für zusätzliches Feedback bestätigt. Gleichzeitig hat sie aber auch aufgezeigt, dass an der Benutzeroberfläche weitere Verbesserungen durchgeführt werden können, um die Benutzerfreundlichkeit zu steigern. Zum größten Kritikpunkt, dass es schwierig ist aus der *Agreement Map* die einzelnen Absätze herauslesen zu können, ist zu ergänzen, dass das gewählte Projekt, mit dem die Evaluation durchgeführt worden ist, relativ kurz war. In einem solchen Fall wäre der Vorschlag, die Absätze mit Strichen zu unterteilen, sinnvoll. Im Kurs NYT werden aber oft Texte mit über hundert Absätzen kodiert, bei dem dann die vielen Striche zur Unterteilung der *Agreement Map* zu Unübersichtlichkeit führen würden. Bei längeren Texten ist die Handhabung der *Agreement Maps* in der gewählten Form vermutlich einfacher. Die Befürchtung, dass die Qualität der einzelnen Peerreviews niedrig sein wird, ist nachvollziehbar und sollte beim kommenden NYT Kurs überprüft werden. Eventuell kann man auch die Bewertungen des Peerreviews mit in die Notengebung einbeziehen, um dieses zu verhindern. Zusätzlich zeigte sich bei der Evaluation auch, dass die Feedbackarten sinnvoll gewählt waren und von verschiedenen Menschen unterschiedlich aufgenommen wurden. So hat jeder Gutachter eine andere Feedbackart als besonders hilfreich eingestuft. Dies unterstützt die ursprüngliche Entscheidung, mehrere statt nur eine Feedbackart zu ermöglichen. Die Anforderungsüberprüfung hat zudem bestätigt, dass der allergrößte Teil der geforderten Anforderungen erfüllt wurden. Ausgenommen davon sind nur die individuell generierbaren Evaluationsschemen für ein Betreuerfeedback des Kursleiters sowie die Anforderungen zur Barrierefreiheit und zu den Akzeptanztests. Die Erfüllung dieser drei noch fehlenden Anforderungen sowie die Beseitigung der Auffälligkeiten aus der heuristischen Evaluation war leider wegen dem festgesetzten zeitlichen Rahmen einer Bachelorarbeit nicht mehr möglich. Dennoch ist das Feedbacksystem so ausgereift, dass es innerhalb der nächsten Kursinstanz von NYT in der Praxis eingesetzt werden kann.

6 Fazit

Ziel der Arbeit war, ein Feedbacksystem innerhalb der Webanwendung QDAcity aufzubauen, dass aus automatisiertem Feedback, Peerreview und Betreuerfeedback besteht.
Ausgehend von der Motivation des Ziels durch Auswertung der Evaluationsbögen der letzten Kursinstanzen von NYT wurden nach der Betrachtung des Kursbereichs von QDAcity mit einem besonderen Augenmerk auf die Bewertung mittels Intercoder Agreement die Anforderungen für ein neues Feedbacksystem aufgestellt (Kapitel 1). Diese unterteilen sich in 25 funktionale und 17 nichtfunktionale Anforderungen (Kapitel 2). Auf Grundlage dieser Vorarbeiten wurden die notwendigen Anpassungen und Erweiterungen der Software-Architektur festgelegt (Kapitel 3), die als Basis für die Implementierung und das Design der neuen Funktionen der Webanwendung dienten (Kapitel 4). Die Umsetzung wurde zunächst anhand der Anforderungsliste rekapituliert. Dabei stelle sich heraus, dass mit 24 der 25 funktionalen Anforderung und 15 der 17 nichtfunktionalen Anforderungen fast alle gesetzten Ziele erreicht werden konnten (Kapitel 5). Darüber hinaus wurde zusätzlich eine heuristische Evaluation mit drei Gutachtern durchgeführt. Diese bestätigt einen erheblichen Mehrwert des Kursbereichs durch die neuen Feedback-Funktionen und zeigt gleichzeitig Verbesserungspotentiale auf.
Zusammenfassend lässt sich sagen, dass die Bachelorarbeit mit großem Erfolg abgeschlossen werden konnte. Der auf Grundlage der am Ende des Kurses NYT ausgefüllten Evaluationsbögen größte Kritikpunkt, der Mangel an Feedback, der auch durch die Aussagen der Gutachter in der heuristischen Evaluation gestützt wurde, konnte verbessert werden. Durch das im Rahmen der Bachelorarbeit entstandene Feedbacksystem wurden mehrere Optionen geschaffen, die den individuellen Lernprozess der Kursteilnehmer verbessern. Sie können aus den Ansichten der *Agreement pro Absatz* und *Agreement pro Code* die Fehlerquellen ihrer Abgaben analysieren und daraus Lehren für die nächste Abgabe ziehen. Die Anzeige der Einordnung der eigenen Leistung in die der anderen Kursteilnehmer und in das vom Kursleiter angelegte Bewertungsschema führt zu mehr Transparenz der Notengebung. Diese Indikatoren sollen zum einen Autoren mit guten Abgaben beruhigen und zum anderen Kursteilnehmer mit schlechteren Abgaben motivieren, mehr Zeit in ihre Abgaben zu investieren, um bessere Bewertungen zu erhalten.

Generell soll durch das neue Feedbacksystem die individuelle Einschätzung der eigenen Leistung jedes einzelnen Kursteilnehmers bereits während der Laufzeit des Kurses gestärkt werden. Das Peerreview bietet den Teilnehmern zwei Vorteile. Zum einen haben sie die Möglichkeit andere Abgaben kennenzulernen. So erfahren sie, wie andere den Text kodiert haben, und können - besonders aus guten Abgaben - lernen, ihre eigenen Fähigkeiten zu verbessern. Zum anderen bekommen sie durch Peerreviews, die sie erhalten, individuelles Feedback zu ihren Abgaben. Eine weitere Option ist das Betreuerfeedback, dass die Betreuer eines Kurses für ausgewählte Abgaben ausfüllen können, um den Teilnehmern ein zusätzliches, konstruktives Feedback zu geben. Da die Teilnehmer erkennen, dass das Review von einem Betreuer kommt, können diese sich über die Korrektheit des Reviews sicher sein und größtmöglichen Nutzen aus dem Feedback ziehen.
Ob mit diesem System alle Kritik über den Mangel an Feedback beseitigt werden konnte, wird sich erst bei der nächsten Evaluationsumfrage des Kurses NYT zeigen. Die erfolgreich durchgeführte heuristische Evaluation ist aber ein positiver Indikator dafür, dass das neue Feedbacksystem auch von den zukünftigen Kursteilnehmern wertgeschätzt wird.

Appendices

A Heuristische Evaluation

In diesem Kapitel befinden sich die Protokolle der einzelnen Evaluationen.

A.1 Evaluation 1

Wie hast du damals, als du NYT belegt hast, dass Feedback wahrgenommen?
Bei dieser Frage wurde kritisiert, dass es zu diesem Zeitpunkt nur den Wert des F-Maßes als Feedback gab, mit dem die Studentin kaum etwas anfangen konnte. Dies hat dazu geführt, dass sie ihre Leistung nicht einordnen konnte und sich daher viel Stress gemacht hat. Rückblickend konnte sie dann aber aufgrund ihrer sehr guten Note im Modul feststellen, dass ihre vorherigen Sorgen unbegründet waren.
Erster Eindruck zum Dashboard eines Reports
Das Design des Dashboards hat ihr sehr gut gefallen. Mit den Anzeigenadeln konnte sie direkt feststellen, wie ihre Abgabe einzuordnen ist. Ihrer Meinung nach hätte sie mit diesen Anzeigen gleich einige Sorgen weniger gehabt.
Agreement pro Absatz
Mit der Seitenleiste, die die *Agreement Maps* beinhaltet, konnte die Studentin zunächst nicht wirklich etwas anfangen. Erst mit dem Hinweis auf den Info Icon hat sie die Funktionalität verstanden. Der angezeigte Infotext hat sich für sehr sinnvoll herausgestellt. Nach einer Erklärung, welche Informationen aus der Seitenleiste ablesbar sind, wurde sie als sehr hilfreich eingestuft.
Agreement pro Code
Die Tabelle fand sie verständlich und sehr hilfreich, um mehr über die Qualität ihrer Abgabe zu erfahren.
Betreuerfeedback und Bewertung des erhaltenen Feedbacks
Das erhaltene Betreuerfeedback hat sie selbständig erkannt und geöffnet. Das Review war für sie logisch aufgebaut und das Design der Likert-Skalen hat ihr gefallen. Auch die Bewertung des Feedbacks hat sie als sinnvoll erachtet und intuitiv abgegeben.
Review verteilen
Auch diese Aufgabe konnte ohne Hilfe durchgeführt werden. Die Fragen waren dabei verständlich.
Hätte dir das neue, erweiterte Feedback geholfen? Was ist aus deiner Sicht besonders hilfreich?
Besonders hat der Studentin die Einordnung in das Evaluationsschema und in die Bewertung anderer Teilnehmer des Kurses geholfen. Sie meinte, dass sie dadurch ein besseres Gefühl dafür bekommen hätte, dass sich der Aufwand für die Abgabe

gelohnt hat.

Siehst du Probleme bei dem neuen Feedback?

Auf diese Frage antwortete sie, dass sie Peerreview etwas problematisch sieht. Ihrer Einschätzung nach ist der Aufwand des Kurses NYT schon ohne Feedback relativ hoch. Ein qualitativ gutes Feedback würde diesen Aufwand noch erhöhen. Deshalb ist ihre Befürchtung, dass sich wenige Studenten Zeit für das Peerreview nehmen und der Autor der Abgabe kein aufschlussreiches Feedback bekommt.

A.2 Evaluation 2

Wie hast du damals, als du NYT belegt hast, dass Feedback wahrgenommen?

Auf diese Frage erzählte der Student, dass er den Zeitpunkt des Feedbacks zu spät fand, da in der Zwischenzeit bereits weitere Abgaben gemacht werden mussten. Außerdem hatte das F-Maß der Abgabe keine Aussagekraft über die Fehlerquelle. Seiner Meinung nach war es schwierig, sich innerhalb des Kurses zu verbessern, da man seine Fehler nicht erkennen konnte und sie deshalb somit in allen Abgaben in identischer Art und Weise gemacht hat.

Erster Eindruck zum Dashboard eines Reports

Beim Dashboard ist ihm zunächst die zweite Kachel, mit den visualisierten Perzentil Wert und der Einordnung in das Bewertungsschema positiv aufgefallen. Bei der ersten Kachel mit dem *Agreement* kritisierte er die Tabellendarstellung. Studenten sehen an dieser Stelle nur eine einzige Zeile mit der eigenen Abgabe. Er hätte dort, wie in der zweiten Kachel, die Werte lieber grafisch visualisiert gesehen. Außerdem konnte er nicht nachvollziehen, warum die Zeile auswählbar ist.

Agreement pro Absatz

Der Student hat direkt die Aussagekraft der *Agreement Maps* verstanden. Der Info Icon ist ihm allerdings nicht aufgefallen. Er hätte sich dabei auch gewünscht, dass beim Klick auf den Info Icon, die ausgewählten *Agreement Maps* geöffnet bleiben würde, und nicht danach wieder neu ausgewählt werden müsste. Außerdem wünschte er sich eine Verbesserung der Zuordnung zwischen den Absätzen und den zugehörigen Farbmarkierung. Ein Vorschlag von ihm war, die Absätze in der *Agreement Map* mit schwarzen Strichen zu unterteilen. So erkennt man die unterschiedlichen Absätze auch bei gleicher Farbschattierung.

Agreement pro Code

Die Tabelle wurde als sehr hilfreich, verständlich und übersichtlich bewertet.

Betreuerfeedback und Bewertung des erhaltenen Feedbacks

Das erhaltene Betreuerfeedback wurde selbständig gesichtet und bewertet. Die Funktionalität und das Design wurde dabei gelobt.

Review verteilen

Auch diese Aufgabe wurde ohne Hilfe durchgeführt. Die Fragen wurden dabei als verständlich und einfach auszufüllen bewertet.

Hätte dir das neue, erweiterte Feedback geholfen? Was ist aus deiner Sicht besonders hilfreich?
Ihm hätte das Feedback sehr geholfen. Vor allem hätte er sich das *Agreement pro Absatz* und *pro Code* gewünscht, da er dadurch hätte lernen können, wo seine Fehler waren und wie diese in der nächsten Aufgabe hätten vermieden werden können.
Siehst du Probleme bei dem neuen Feedback?
Problematisch hat er die Perzentil-Bewertung eingestuft. Er vermutet, dass er bei einer schlechteren Bewertung als 50% den Kurs wahrscheinlich verlassen hätte, da ihn eine solche Bewertung deprimiert hätte. Dies wäre auch dann der Fall gewesen, wenn die Einstufung in das Bewertungsschema gut gewesen wäre.

A.3 Evaluation 3

Wie hast du damals, als du NYT belegt hast, dass Feedback wahrgenommen?
Zu dieser Frage konnte der Student keine Aussage treffen, da er zwar Aufgaben in QDAcity bearbeitet hatte, aber den Kurs abgebrochen hat, bevor die erste Aufgabe bewertet worden ist.
Erster Eindruck zum Dashboard eines Reports
Das Dashboard fand er schön übersichtlich. Ihm hat das Layout mit den unterschiedlichen Kacheln gefallen.
Agreement pro Absatz
Da der Student eine Rot-Grün-Sehschwäche hat sahen die Farbschattierungen der roten Färbung der FN und FP und der grünen Färbung der RP für ihn gleich aus. Deshalb fand er es irritierend, dass die dunklere Färbung bei RP positiv zu bewerten ist und bei den anderen zwei Balken aber negativ. Zusätzlich fand er die Zuordnung zwischen Absatz und zugehöriger Einfärbung in der *Agreement Map* schwierig.
Agreement pro Code
Hier hätte sich der Student einen Info Icon hinter den Begriffen *FMeasure*, *Recall* und *Precision* gewünscht, der die Bedeutung der Begriffe erklärt. Zudem hat er vorgeschlagen, dass die Werte als Prozentzahlen angegeben werden.
Betreuerfeedback und Bewertung des erhaltenen Feedbacks
Die Funktionalität des Anschauens und des Bewertens eines erhaltenen Reviews hat er als sehr nutzerfreundlich eingestuft.
Review verteilen
Beim Verteilen des Feedbacks hat er sich gewünscht, dass die offenen Fragen mit einem Sternchen als Pflichtfelder markiert wären. Er hatte zunächst versucht, diese leer zu lassen und wurde erst beim Abgeben darauf hingewiesen, dass diese auch ausgefüllt werden müssen. Die Fragen fand er gut ausgewählt.
Hätte dir das neue, erweiterte Feedback geholfen? Was ist aus deiner Sicht besonders hilfreich?

Zum einen hat er hier die *Agreement Map* positiv erwähnt, die ihm ermöglichen würde transparenter die Bewertung seiner Abgabe einsehen zu können. Zum anderen fand er vor allem die Möglichkeit, durch die Peerfeedbacks Abgaben anderer Kursteilnehmer einzusehen, gut. So könne man sehen, wie andere bei der Aufgabenbearbeitung vorgegangen sind.

Siehst du Probleme bei dem neuen Feedback?

Bezüglich Feedback befürchtet er, dass die Qualität der Peerreviews gering sei, da man eventuell nur möglichst viele Reviews macht, um sich andere Abgaben anzuschauen, aber nicht, um diese sinnvoll zu bewerten.

B Testabdeckung durch JUnit Tests

Im Folgenden wird die genaue Testabdeckung, die mithilfe von JaCoCo gemessen wird, aufgegliedert. Die Klasse *ExerciseEndpoint* wurde um den neuen Endpunkt *addSnapshotToExerciseProject* erweitert. Diese Methode hat eine Testabdeckung von 100%, die gesamte Klasse hat eine Testabdeckung von 90%, wie in Abbildung B.1 zu sehen ist.
In der Klasse *ExerciseController* wurden die Methoden *createReportForCodebookCreationExercise*, *checkAndCreateExerciseProjectSnapshotsAndReportIfNeeded* und *createSnapshotsForExpiredExerciseProjects* mit jeweils einer Testabdeckung von 100% und *addSnapshotToExerciseProject* mit 84% implementiert. Die genaue Übersicht über die Abdeckung innerhalb dieser Klasse ist in der Abbildung B.2 zu sehen. Insgesamt werden 93% der Klasse getestet.
Für die Daten zur Darstellung der *Agreement pro Absatz* und *Agreement pro Code* wurde die Klasse *AgreementMapEndpoint* um die beiden neuen Methoden *getAgreementMapsForProject* und *getAgreementPerCodeForProject* erweitert, die eine Testabdeckung von jeweils 100% haben. In der Abbildung B.3 sind diese dargestellt und auch, dass die Testabdeckung der ganzen Klasse bei 90% liegt.
Der *ReviewEndpoint* ist im Rahmen der Bachelorarbeit neu dazugekommen und hat, wie in Abbildung B.4 zu sehen, eine Testabdeckung von 92%.
Für die Autorisierung der im *ReviewEndpoint* neu entstandenen Endpunkte wurden die neuen Methoden *isAuthorizedToMakeInstructorReview*, *isAuthorizedToMakeNextReviewFeedback* und *isAuthorizedToUpdateReviewFeedback* mit jeweils einer Testabdeckung von 100%, *isAuthorizedToSetFeedbackOfFeedback* mit 86% und *isAuthorizedToGetReviewFeedback* mit 67% hinzugefügt. Insgesamt haben die fünf neuen Methoden, die auch in der Abbildung B.5 zu sehen ist, eine Testabdeckung von 91%.
Auch die Klassen im Java Paket *Review* sind neu implementiert worden. Die Testabdeckung darin enthaltener Klassen ist insgesamt bei 90%. Dies wird in Abbildung 6 gezeigt.

ExerciseEndpoint

Element	Missed Instructions	Cov.
getExerciseProjectByRevisionID(Long, Long, User)		100%
getExercisesOfExerciseGroup(Long, User)		100%
deleteExerciseProjectReport(Long, User)		100%
getExerciseProjectsByExerciseID(Long, User)		100%
listExerciseReportsByRevisionID(Long, Long, User)		100%
listExerciseResultsForProject(Long, Long, Long, User)		100%
removeExercise(Long, User)		100%
listTermCourseExercises(Long, User)		100%
listTermCourseExerciseGroups(Long, User)		100%
insertExerciseGroupForNewExercise(Exercise, Long, String, User)		100%
createExerciseReport(EvaluateReport, User)		100%
getExercisesByProjectRevisionID(Long, User)		100%
getExerciseGroupsByProjectRevisionID(Long, User)		100%
createAndInsertExerciseToExerciseGroup(Exercise, Long, User)		100%
getExerciseGroupByID(Long, User)		100%
getExerciseByID(Long, User)		100%
lambda$createExerciseProjectIfNeeded$2(Long, User, Long, Context)		100%
lambda$createExerciseProject$1(Long, User, Context)		100%
initializeExerciseGroup(String, Long, Long, List)		100%
lambda$addSnapshotToExerciseProject$3(Long, User, Long, Context)		100%
persistExerciseGroup(ExerciseGroup)		100%
lambda$insertExercise$0(Exercise, User, Context)		100%
createExerciseProjectIfNeeded(Long, Long, User)		100%
addSnapshotToExerciseProject(Long, Long, User)		100%
insertExercise(Exercise, User)		100%
createExerciseProject(Long, Long, User)		100%
ExerciseEndpoint()		100%
getPersistenceManager()		100%
listExerciseResults(Long, Long, User)		98%
listExerciseReports(Long, Long, User)		92%
throwIfUserIsNotAuthorizedForTermCourse(Long, User)		70%
insertExerciseGroup(ExerciseGroup, User)		66%
sendNotificationEmailExercise(Long, User)		0%
containsExerciseGroup(ExerciseGroup)		0%
Total	75 of 995	92%

Abbildung B.1: Testabdeckung *ExerciseEndpoint*

ExerciseController

Element	Missed Instructions	Cov.
createReportForCodebookCreationExercise(Exercise)		100%
checkAndCreateExerciseProjectSnapshotsAndReportIfNeeded()		100%
createSnapshotsForExpiredExerciseProjects(Exercise)		100%
createExerciseProjectIfNeeded(Exercise, Long, Long)		100%
initializeExerciseProject(Exercise, ProjectRevision)		100%
listCodingsForExerciseProject(Long)		100%
insertExercise(Exercise)		100%
createNewCodeSystemIfNeeded(ProjectRevision, ExerciseProject, ExerciseType)		100%
createExerciseProjectAndCloneDocuments(Exercise)		100%
insertExerciseWithCodebookCreation(Exercise)		100%
insertExerciseWithCodebookUtilization(Exercise)		100%
getProjectRevisionById(Long)		100%
ExerciseController(Context)		100%
createEmptyCodeSystem(ExerciseProject)		100%
with(Context)		100%
persistExerciseProject(Exercise, ProjectRevision)		100%
createExerciseProject(Exercise)		100%
addSnapshotToExerciseProject(Exercise, Long)		84%
throwIfExerciseDoesAlreadyExist(Exercise)		30%
containsExercise(Exercise)		0%
Total	41 of 612	93%

Abbildung B.2: Testabdeckung *ExerciseController*

AgreementMapEndpoint

Element	Missed Instructions	Cov.
getAgreementMapsForProject(Long, ProjectType, Long, User)		100%
getAgreementPerCodeForProject(Long, ProjectType, Long, Long, User)		100%
AgreementMapEndpoint()		100%
getPersistenceManager()		100%
getAgreementMaps(Long, ProjectType, User)		76%
throwIfUserIsNotAuthorizedForTermCourse(Long, User)		70%
Total	25 of 251	90%

Abbildung B.3: Testabdeckung *AgreementMapEndpoint*

ReviewEndpoint

Element	Missed Instructions	Cov.
lambda$getSignedDownloadUrlsForDocumentIdForReview$7(Long, Long, Long, User, Long, Context)		83%
lambda$getDocumentsForReview$6(Long, Long, Long, User, Context)		80%
lambda$submitReviewContextCall$5(Long, ReviewFeedback, User, Context)		79%
lambda$updateReviewFeedbackContextCall$4(Long, ReviewFeedback, User, Context)		79%
lambda$getSnapshotForReviewId$3(Long, Long, Long, User, Context)		78%
lambda$getReviewFeedbackByIdIfFinishedAndInstructor$1(Long, Long, Long, User, Context)		100%
lambda$getReviewFeedbackByIdIfFinshed$0(Long, Long, Long, User, Context)		100%
lambda$getReviewFeedbackById$2(Long, Long, Long, User, Context)		100%
lambda$getResultToReview$8(Long, User, Long, Long, ExerciseType, Context)		100%
lambda$getReviewInstructor$9(Long, User, Long, ExerciseType, Context)		100%
getSignedDownloadUrlsForDocumentIdForReview(Long, Long, Long, Long, User)		100%
getResultToReview(Long, Long, Long, ExerciseType, User)		100%
getReviewFeedbackByIdIfFinshed(Long, Long, Long, User)		100%
getReviewFeedbackByIdIfFinishedAndInstructor(Long, Long, Long, User)		100%
getReviewFeedbackById(Long, Long, Long, User)		100%
getSnapshotForReviewId(Long, Long, Long, User)		100%
getDocumentsForReview(Long, Long, Long, User)		100%
getReviewInstructor(Long, Long, ExerciseType, User)		100%
updateReviewFeedbackContextCall(Long, ReviewFeedback, User)		100%
submitReviewContextCall(Long, ReviewFeedback, User)		100%
updateReviewFeedbackCodebookUtilization(Long, ReviewFeedbackCodebookUtilization, User)		100%
updateReviewFeedbackCodebookCreation(Long, ReviewFeedbackCodebookCreation, User)		100%
updateReviewFeedback(Long, ReviewFeedback, User)		100%
submitReview(Long, ReviewFeedback, User)		100%
submitReviewCodebookUtilization(Long, ReviewFeedbackCodebookUtilization, User)		100%
submitReviewCodebookCreation(Long, ReviewFeedbackCodebookCreation, User)		100%
ReviewEndpoint()		100%
Total	25 of 324	92%

Abbildung B.4: Testabdeckung *ReviewEndpoint*

isAuthorizedToGetReviewFeedback(Review, Long, Long, User)		67%
isAuthorizedToMakeInstructorReview(Long, User)		100%
isAuthorizedToMakeNextReviewFeedback(Long, User)		100%
isAuthorizedToRemoveParticipant(TermCourse, String, User)		0%
isAuthorizedToSetFeedbackOfFeedback(Review, Long, User)		86%
isAuthorizedToUpdateReviewFeedback(Review, Long, User)		100%

Abbildung B.5: Testabdeckung der neuen Methoden in der Klasse *Authorization*

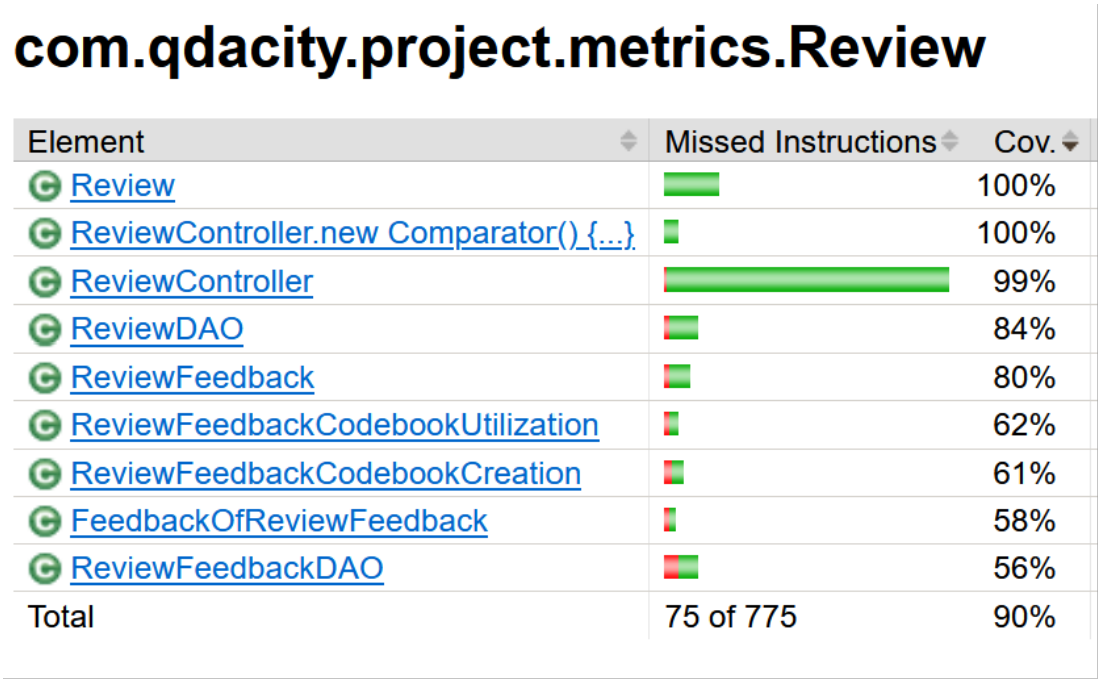

Abbildung B.6: Testabdeckung des Pakets *Review*

Literaturverzeichnis

Ackermann, P. (2021). *Webentwicklung Das Handbuch für Fullstack-Entwickler*. Rheinwerk-Verlag.

Benning, V. (2023a). Arithmetisches Mittel verstehen und berechnen - mit Beispielen. https://www.scribbr.de/statistik/arithmetisches-mittel/

Benning, V. (2023b). Standardabweichung verstehen und berechnen + Rechner. https://www.scribbr.de/statistik/standardabweichung/

Campbell, J. L., Quincy, C., Osserman, J., & Pedersen, O. K. (2013). Coding In-depth Semistructured Interviews: Problems of Unitization and Intercoder Reliability and Agreement. *Sociological Methods & Research*, (42).

Cliodhna, O., & Helene, J. (2020). Intercoder Reliability in Qualitative Research: Debates and Practical Guidelines. *International Journal of Qualitative Methods*, *19*.

DeLyser, D. (2008). Teaching Qualitative Research. *Journal of Geography in Higher Education*, *32*(2).

Harrison, M. (2019). *Machine Learning - Die Referenz*. O'Reilly Media, Inc.

Jiusto, S., & DiBiasio, D. (2006). Experiential learning environments: Do they prepare our students to be self-directed, life-long learners? *Journal of Engineering Education*, *95*(3), 195–204.

Kaufmann, A., Barcomb, A., & Riehle, D. (2016). Using Students as a Distributed Coding Team for Validation through Intercoder Agreement. https://open.fau.de/server/api/core/bitstreams/12ab69f2-8fc0-4760-8f51-c19b840326da/content

Kaufmann, A., Riehle, D., Krause, J., & Harutyunyan, N. (2023). A Solution for Automated Grading of QDA Homework. *Proceedings of the 56th Hawaii International Conference on System Sciences*, 44–53.

Kolb, D. (1984). *Experiential learning: experience as the source of learning and development*. Prentice Hall.

Lowe, M. S. (1992). Safety in Numbers? How to Teach Qualitative Geography. *Jounal of Geography in Higher Education*, *16*(2).

MacQueen, E., Kathleen M.and McLellan, Kay, K., & Milstein, A. (1998). Codebook development for team-based qualitative analysis. *Cultural Anthropology Methods*, *10*(2).

Moran, K., & Gordon, K. (2023). How to Conduct a Heuristic Evaluation. https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/

Narayn, H. (2022). *Just React!: Learn React the React Way*. Apress.

Nielsen, J. (1994). The Theory Behind Heuristic Evaluations. https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/theory-heuristic-evaluations/

Norman, D. (2016). *The Design of everyday things*. Verlag Franz Vahlen München.

SOPHISTen. (2016). Schablonen für alle Fälle. (3). https://www.sophist.de/fileadmin/user_upload/Bilder_zu_Seiten/Publikationen/Wissen_for_free/MASTeR_Broschuere_3-Auflage_interaktiv.pdf

Springer, S. (2023). *React Das umfassende Handbuch*. Rheinwerk-Verlag.