Hochschule Flensburg
University of Applied Sciences

Abschlussbericht

Projekt »GitClassrooms«

Fachbereich:

Information und Kommunikation

Studiengang:

Angewandte Informatik (Master)

Prüfungsordnung:

PSO M AI 2022

Modul (Semester):

781011/781012 Projekt (WiSe 2023/2024–SoSe 2024)

Dozent: Prof. Dr. Niklas Klein

Abgabedatum: 27.09.2024

Studierende:

Borucki, Philipp
Mat.-Nr.: 690114

Friedrichsen, Pascal
Mat.-Nr.: 670188

Heckner, Dominik
Mat.-Nr.: 670005

Ingwersen, Hauke
Mat.-Nr.: 670177

Klein, Claas Thore
Mat.-Nr.: 670594

Nebendahl, Jannes
Mat.-Nr.: 750594

Neufeldt, Stefan
Mat.-Nr.: 710372

Petersen, Fabian
Mat.-Nr.: 670111

Porsche, Nils
Mat.-Nr.: 690870

Inhaltsverzeichnis

1 Einleitung

Dieser Abschlussbericht wurde im Kontext des zweisemestrigen Projektes »Git-Classrooms« in Rahmen des Studiengangs »M. Sc. Angewandte Informatik« verfasst und dient sowohl dem Zweck der Dokumentierung des Projekts bzw. Teilen der Anwendung als auch einer weitergehenden Betrachtung mit Ausblick auf mögliche zukünftige Weiterentwicklungen.

Beim konkreten Projekt handelt sich in diesem Fall um die Spezifizierung, Strukturierung, Umsetzung und Dokumentation eines Unterrichtswerkzeugs für die Software-Versionsverwaltung »GitLab«, mit dem Dozierende digitale Klassenräume und Aufgaben verteilen, verwalten und benoten können.

Einleitend wird das Projektziel detaillierter vorgestellt, näher auf die zu erwartenden Herausforderungen eingegangen und eine grundsätzliche Softwaredokumentation auch zum Zwecke einer Weiterentwicklung gegeben.

1.1 Projektidee und -ziel

Digitale Lernmanagementsysteme wie Moodle, Stud.IP und andere sind in vielen Bildungseinrichtungen etabliert und bieten grundlegende Funktionen zur Verwaltung und auch Bewertung von Kursen und Lerninhalten. Diese Plattformen stoßen jedoch bei der Durchführung von Aufgaben, die das Einreichen und Bewerten von Quellcode erfordern, an funktionale Grenzen. Ein wesentlicher Grund hierfür ist die fehlende Integration von Versionskontrollsystemen, die in der Softwareentwicklung eine zentrale Rolle spielen.

Versionskontrollsysteme, insbesondere Git-basierte Plattformen wie GitHub, GitLab und Gitea, bieten Werkzeuge zur kollaborativen Arbeit an Quellcode, zur Nachverfolgbarkeit von Änderungen und zur Verwaltung von Softwareprojekten. Die Anforderungen im Bildungsbereich gehen jedoch über die reine Versionskontrolle hinaus. Insbesondere das Verteilen von Aufgaben, das Einsammeln und Bewerten von Studierendenarbeiten, sowie die Unterstützung durch automatische Bewertungsmechanismen sind zentrale Anforderungen, die für eine Verwaltung von Programmierprojekten im Lehrbetrieb hilfreich sind.

GitHub Classroom ist eine Lösung, die zusammen mit GitHub, als separate Anwendung, zusätzliche Funktionen für den Bildungssektor bereitstellt, wie etwa die Erstellung und Verwaltung von Klassenräumen, die Verteilung von Aufgaben und die automatisierte Bewertung von Programmierlösungen. Diese Funktionen erlauben eine bessere Organisation und Durchführung von Lehrveranstaltungen mit einem Fokus auf Programmierprojekte.

Bei dieser Plattform handelt es sich jedoch um ein proprietäres Produkt, das von Microsoft betrieben wird, was bei Bildungseinrichtungen, die auf Datenschutz und die Kontrolle über eigene Daten besonderen Wert legen oder müssen, auf Bedenken stoßen kann. In solchen Fällen wird häufig auf selbstver-

waltete Lösungen wie GitLab zurückgegriffen, welche eine höhere Autonomie und Datensouveränität bieten. GitLab verfügt jedoch selbst über keine spezifischen Funktionen zur Verwaltung von digitalen Klassenzimmern oder zur automatisierten Aufgabenverteilung und -bewertung, was den Einsatz im Lehrbetrieb einschränkt.

Dieses Projekt setzt sich daher zum Ziel, eine Anwendung zu entwickeln, die Lehrenden bei der Verwaltung digitaler Klassenzimmer und deren Aufgaben unterstützt und gleichzeitig eine Integration in bestehende GitLab-Instanzen ermöglicht. Die Anwendung soll die Verwaltung von Kursen mit Programmieraufgaben erleichtern, indem sie Lehrenden Funktionen zur Organisation, Aufgabenverteilung und Bewertung von Quellcode zur Verfügung stellt. Dabei sollen Lehrende durch die Definition von eigenen Bewertungskriterien und auch teilweise automatisierte Bewertungen durch Tests unterstützt werden, wodurch Lehrende einen Überblick über die Projekte der Teilnehmenden bzw. der durchgeführten Bewertungen erhalten und so auch ihren eigenen Verwaltungsaufwand reduzieren können.

1.2 Herausforderungen

Im Rahmen der Entwicklung einer solchen Anwendung gibt es bereits vorab einige Herausforderungen. Diese betreffen nicht nur die technische Umsetzung der Integration mit GitLab, sondern auch die funktionalen Anforderungen, die an eine solche Lösung gestellt werden. Zentrale Herausforderungen waren dabei unter anderem die Architektur bzw. Art der Integration mit GitLab, die Realisierung der dazugehörigen Strukturen bzw. der Klassenraumumgebung in GitLab und die Implementierung eines Bewertungssystems.

Integration mit GitLab: Möglichkeiten und Anforderungen

Eine wichtige Herausforderung bei der Entwicklung der Lernplattform bestand darin, die passende Architektur zu wählen, um eine Integration mit GitLab zu gewährleisten. Dabei gab es verschiedene Möglichkeiten, wie beispielsweise eine direkte Integration durch Erweiterung von GitLab selbst oder eine Standalone-Anwendung, welche über geeignete Schnittstellen arbeitet.

Direkte Integration in GitLab Bei diesem Ansatz würde die Anwendung als Teil von GitLab direkt in die Benutzeroberfläche und das darunterliegende System integriert werden. Hierbei müssten Funktionen wie das Anlegen von Kursen und Aufgaben sowie die Bewertung von Abgaben nahtlos in die bestehenden Workflows von GitLab eingegliedert werden. Eine besondere Herausforderung stellt die Integration in GitLab dar, insbesondere da es für diesen Zweck nicht entwickelt wurde, da es kein dediziertes Plugin-System gibt. Daher müsste die Codebasis von GitLab direkt erweitert werden und bei eventuelle Änderungen

in zukünftigen GitLab-Versionen müssten Anpassungen vorgenommen werden, um mit dieser kompatibel bleiben.

Standalone Lösung über die API Eine Alternative zur direkten Integration ist die Entwicklung einer Standalone-Anwendung, die über die Schnittstellen mit GitLab kommuniziert. Dieser Ansatz bietet mehr Flexibilität, da GitLab selbst nicht modifiziert werden muss und die Anwendung unabhängig von GitLab-Updates weiterentwickelt werden kann. Dieser Ansatz erfordert eine umfassende Nutzung der GitLab-API, um Funktionen wie Gruppen-Management, Projekt-Management, sowie Funktionen für die Bewertung der Studierendenarbeiten zu ermöglichen. Bei diesem Ansatz musste insbesondere geprüft werden, ob die vorhandenen Strukturen in GitLab - insbesondere das Gruppen- und Rechtesystem - geeignet sind, um das Szenario eines Klassenraums sowie die Bearbeitung von Aufgaben abzubilden.

Prüfung notwendiger Features und deren Umsetzbarkeit

Ein weiterer wichtiger Punkt besteht darin, vor dem Projektstart abzugrenzen, welche Features tatsächlich erforderlich sind und welche in der gegebenen Zeit in Abhängigkeit von der gewählten Architektur realisiert werden können. Nicht alle wünschenswerten Funktionen sind zwangsläufig umsetzbar, und es muss ein Kompromiss zwischen Funktionalität, technischer Machbarkeit, verfügbarer Ressourcen und dem zur Verfügung stehenden Zeitrahmen gefunden werden. Dies erfordert eine Festsetzung von Mindestanforderungen bei den zu entwickelnden Features, um sicherzustellen, dass die wesentlichen Elemente der Anwendung implementiert werden, ohne dabei den Projektzeitrahmen zu überschreiten.

Abbildung in GitLab: Rollen und Rechte

Die Schaffung einer kontrollierten Arbeitsumgebung (bzw. Abbildung des Klassenraums in GitLab), die den Anforderungen des Lehrbetriebs entspricht, stellt damit auch eine der weiteren Herausforderungen dar.

In GitLab gibt es bereits ein differenziertes Rollen- und Rechtesystem sowie Gruppensystem, welche grundlegend geeignet ist eine Klassenraumstruktur abzubilden. Ein besonderes Augenmerk bei der Konzeption lag darauf, dass Lehrende die vollständige Kontrolle über die Klassenraumstruktur haben, während Studierende eingeschränkten Zugriff erhalten. So sollten Lehrende zum Beispiel in der Lage sein, Aufgaben zu erstellen, Quellcode einzusehen und Bewertungen durchzuführen, während Studierende nur Zugriff auf die ihnen zugewiesenen Aufgaben und eigene Projekte haben. Hierbei müssen die Rollen in GitLab klar definiert werden, um diesen Anforderungen gerecht zu werden. Dabei sollte ein wenig Flexibilität möglich sein, sodass Studierende auch wahlweise in Teams arbeiten können oder optional den Quellcode anderer Studierende für dieselbe Aufgabe einsehen können.

Bewertung von Aufgaben bzw. Abgaben

Die Umsetzung einer manuellen Bewertung von Programmieraufgaben ist relativ simpel, da Lehrende eigene Kriterien festlegen und diese auf die Klassenräume bzw. Aufgaben anwenden können. Eine komplexere Herausforderung stellt die automatisierte Bewertung dar, bei der ein flexibles Konzept erforderlich ist, das verschiedene Programmiersprachen und Projektarten unterstützt. Um diese Vielfalt abzudecken, muss die automatisierte Bewertung als Basis möglichst auf bereits vorhandene Auswertungskonzepte oder bereits integrierte Mechanismen in GitLab zurückgreifen – wie zum Beispiel CI/CD-Pipelines, die Tests ausführen und Ergebnisse generieren können. Dadurch lässt sich ein robuster Bewertungsprozess aufbauen, der bereits eine Vielzahl von Szenarien abbilden kann.

1.3 Bestehende Ansätze und Lösungen

In der anfänglichen Recherche wurden einige Ansätze und Lösungen gefunden. Ziel war es, bestehende Tools oder Plattformen zu finden, die bereits ähnliche Funktionalitäten bieten, um diese als Grundlage oder Inspiration für die eigene Entwicklung zu nutzen. Insbesondere bestand die Frage, ob es bereits ausgereifte und aktiv gepflegte Open-Source-Projekte gibt, die entweder direkt in die bestehende GitLab-Infrastruktur integriert werden können oder eine einfache Anpassung ermöglichen.

Bestrebungen, eine Lösung für den Bildungssektor intern im GitLab-Team umzusetzen, sind seit sieben Jahren ohne Resonanz geblieben (siehe Issue[1]).

Die anfängliche Recherche ergab einige Projekte, die grundsätzlich den gewünschten Ansatz verfolgten. Allerdings wurde bei einer tiefergehenden Untersuchung klar, dass keine der Lösungen ausreichend fortgeschritten, funktionsfähig oder gepflegt war. Die folgenden Abschnitte geben einen kurzen Überblick über die untersuchten Projekte.

- GitLab Classrooms von C.Joly[2]
- classrooms.gitlab.io[3]
- GitHub Repository for Gitlab Classrooms[4]

[1] @motatoes, *GitLab Classrooms Feature Proposal*.
[2] Joly, *Telecomnancy Web: Github Education-like dashboard for GitLab*.
[3] Benante, *Classroom GitLab Pages Repository*.
[4] @motatoes, *GitLab Classrooms Feature Proposal*.

1.3.1 »Gitlab Classrooms von C. Joly«

Das Projekt »Gitlab Classrooms von C. Joly« wurde als ein Schulprojekt entwickelt und bietet eine Webschnittstelle zur Automatisierung der Erstellung von Repositories und zur Verteilung von Aufgaben auf GitLab. Es wurde mit Python, Flask und SQLite implementiert. Die Integration erfolgt über die GitLab-API, was es ermöglicht, Repositories zu verwalten. Das Projekt wurde jedoch seit 2021 nicht mehr weiterentwickelt und ist archiviert. Die Funktionalitäten sind recht begrenzt und beschränken sich hauptsächlich auf die Grundverwaltung von GitLab-Repositories.

1.3.2 »github.com/gitlab-classroom«

Das Projekt »gitlab-classroom« wurde entwickelt, um die Verteilung von Aufgaben und die Verwaltung von Repositories auf GitLab zu automatisieren. Es besteht aus zwei Hauptkomponenten: einem Frontend »classroom-web« und einem Backend »classroom-server«, beide in JavaScript geschrieben. Das Frontend ermöglicht Lehrenden über eine Webschnittstelle das Erstellen und Verwalten von Aufgaben, während das Backend die Geschäftslogik und die Kommunikation mit der GitLab-API abwickelt.

Die Architektur ist gut strukturiert, da das Backend und das Frontend unabhängig voneinander funktionieren. Allerdings gibt es mehrere Gründe, das Projekt nicht als Basis zu verwenden.

- **Kein Fortschritt:** Der Funktionsumfang ist reduziert.
- **Veraltete Codebasis:** Seit 2016 wurden keine nennenswerten Updates vorgenommen, was die Weiterentwicklung für moderne Anforderungen erschwert.
- **Mangelnde Pflege:** Das Projekt wird nicht aktiv betreut, was bedeutet, dass eine Kompatibilität mit GitLab nicht mehr sichergestellt ist.
- **Fehlende Dokumentation:** Die vorhandene Dokumentation ist unvollständig, und es gibt nur wenige Tests, was die Integration und Erweiterung riskant macht.

Zusammenfassend lässt sich sagen, dass die Architektur zwar grundsätzlich passend ist, die fehlende Pflege, fehlende Funktionalität und die veraltete Codebasis es aber zu einer ungeeigneten Grundlage für die Weiterentwicklung oder für ein neues Projekt macht. Eine Adaptierung wäre zeitlich ineffizient.

1.3.3 »classroom.gitlab.io«

Das Projekt »classroom.gitlab.io« stellt einen Fork des ursprünglichen GitHub-Classroom-Projekts dar,[5] dessen Weiterentwicklung inzwischen nicht mehr öffentlich zugänglich ist. Ziel dieses Forks war es, die Funktionalität des ursprünglichen Projekts für die Nutzung mit GitLab zu adaptieren. Seit der Erstellung des Forks wurden insgesamt vier Commits durchgeführt. Diese umfassen hauptsächlich Änderungen des Namens in der README.md sowie auf der 404-Fehlerseite von GitLab-Classroom und dem Hinzufügung einer .gitlab-ci.yml-Datei. Diese Änderungen hatten jedoch bislang keine wesentlichen Auswirkungen auf die Funktionsweise, sodass seit der Duplikation keine substantiellen Entwicklungen zu verzeichnen sind.

1.3.4 Ergebnis

Zusammenfassend lässt sich feststellen, dass die durchgeführte Überprüfung bestehender Ansätze zwar mehrere Projekte identifiziert hat, die ähnliche Zielsetzungen verfolgen, jedoch keines der untersuchten Systeme den Anforderungen als Grundlage gerecht wird. Die mangelnde Pflege, der begrenzte Funktionsumfang sowie veraltete Codebasen führen dazu, dass diese Lösungen für eine nachhaltige Weiterentwicklung ungeeignet erscheinen. Daher wird eine Neuentwicklung verfolgt, um den im nächsten Kapitel beschriebenen funktionalen Anforderungen gerecht zu werden.

[5] al., *GitHub Classroom*.

1.4 Funktionale und nicht-funktionale Anforderungen

1.4.1 Funktionale Anforderungen

Im folgenden Kapitel werden die funktionalen Anforderungen an die Anwendung beschrieben, die mindestens implementiert werden müssen. Dabei werden die Anforderungen den jeweiligen Akteuren zugeordnet. Die Anwendung soll so gestaltet sein, dass Rollen flexibel bleiben, sodass jeder Benutzer sowohl Klassenräume verwalten als auch ihnen beitreten kann.

Authentifizierung

|A01| Auth über OAuth mit GitLab

- **Akteure:** Studierende, Dozierende
- **Beschreibung:** Benutzer melden sich mit ihren GitLab-Anmeldedaten in der »GitClassrooms«-Anwendung an. GitLab übernimmt die Authentifizierung. Dies hat den Vorteil, dass keine eigene Benutzerverwaltung entwickelt und betrieben werden muss. Durch die Nutzung von GitLab für die Authentifizierung können Benutzer sich sicher mit ihren bestehenden GitLab-Anmeldeinformationen anmelden. Dies reduziert den Aufwand für die Verwaltung von Benutzerkonten und erhöht die Sicherheit, da die Authentifizierung über ein bereits bestehendes System erfolgt.

Übersichtsfunktionen

|A02| Übersicht über verwaltete und beigetretene Klassenräume

- **Akteure:** Studierende, Dozierende
- **Beschreibung:** Benutzende sehen eine Übersicht über die Klassenräume, denen sie beigetreten sind oder die sie verwalten.

|A03| Übersicht über anstehende Aufgaben

- **Akteure:** Studierende, Dozierende
- **Beschreibung:** Studierende und Dozierende erhalten eine Übersicht über alle anstehenden Aufgaben, den dazugehörigen Klassenraum und deren Abgabefristen.

Verwaltung von Klassenräumen

|A04| Erstellung von Klassenräumen

- **Akteure:** Dozierende
- **Beschreibung:** Dozierende haben die Möglichkeit, neue Klassenräume zu erstellen und individuell zu konfigurieren. Dabei können sie den Namen des Klassenraums festlegen, eine kurze Beschreibung hinzufügen und entscheiden, ob Aufgaben in Teams bearbeitet werden sollen. Falls dies der Fall ist, können sie die maximale Anzahl der Teams sowie die maximale Anzahl der Studierenden pro Team festlegen. Auch soll festlegbar sein, ob Studierende eigene Teams erstellen dürfen. Der Klassenraum soll in GitLab als Gruppe repräsentiert werden.

|A05| Editierung von Klassenräumen

- **Akteure:** Dozierende
- **Beschreibung:** Dozierende können bestehende Klassenräume ändern. Darunter fallen mindestens der Name und die Beschreibung. Die Änderungen sollen auch in GitLab übernommen werden.

|A06| Archivierung von Klassenräumen

- **Akteure:** Dozierende, Anwendung
- **Beschreibung:** Klassenräume können nach Ablauf des Kurses vom Dozierenden archiviert werden. Studierende verlieren eventuell noch vorhandene Schreibrechte in den zugehörigen Projekten (Repositories) des Klassenraums. Dieser Vorgang kann auch alternativ durch die Anwendung ausgelöst werden, sollte die Verwaltung eines Klassenraums in GitLab nicht mehr möglich sein.

|A07| Einladen von Personen in einen Klassenraum

- **Akteure:** Dozierende, Moderierende
- **Beschreibung:** Dozierende und Moderierende können Studierende über E-Mail-Adressen in einen Klassenraum einladen. Dabei sollen die Studierenden per E-Mail benachrichtigt werden und einen personalisierten Link erhalten, über den sie dem Klassenraum beitreten können. Personalisierte Links können ablaufen.

|A08| Beitreten zu einem Klassenraum

- **Akteure:** Studierende
- **Beschreibung:** Studierende treten einem Klassenraum bei, nachdem sie eine Einladung per E-Mail erhalten haben. Sie authentifizieren sich über ihren GitLab-Account. Sofern der Klassenraum Teams vorsieht, müssen Studierende einem Team beitreten. Optional kann ein Team erstellt werden, sofern dies für Studierende freigegeben wurde (siehe |A16| Erstellung von Teams). Wenn keine Teams für den Klassenraum vorgesehen sind, dann werden die Aufgaben-Repositories für die einzelnen Studierenden in einer Untergruppe in GitLab zusammengefasst.

|A09| Übersicht über den Klassenraum

- **Akteure:** Dozierende, Moderierende, Studierende
- **Beschreibung:** Übersicht über den gesamten Klassenraum mit Informationen zu zugewiesenen Aufgaben, Mitgliedern und Teams. Die Übersicht soll an die jeweilige Rolle angepasst werden und weiterführende Optionen für diesen anzeigen.

Verwaltung von Aufgaben

|A10| Übersicht einer Aufgabe

- **Akteure:** Dozierende, Moderierende, Studierende
- **Beschreibung:** Übersicht einer Aufgabe mit der Auflistung der Projekt-Repositories der einzelnen Teams bzw. Studierenden. Insbesondere soll der Status des Teams für die Aufgabe angezeigt werden (beispielsweise »Eingeladen«, »Wird erstellt«, »Fehler«, »Akzeptiert« oder ähnliches) und das dazugehörige Projekt in GitLab verknüpft werden.

|A11| Erstellung von Aufgaben

- **Akteure:** Dozierende
- **Beschreibung:** Aufgaben können auf der Basis eines bestehenden öffentlichen GitLab-Repositories als Vorlage erstellt werden. Es kann ein Name, eine Beschreibung und ein Fälligkeitsdatum gesetzt werden. Das Fälligkeitsdatum soll später den Zeitpunkt markieren, an welchem Studierende im Repository nicht mehr weiterarbeiten können bzw. Schreibrechte verlieren.

|A12| Editierung von Aufgaben

- **Akteure:** Dozierende
- **Beschreibung:** Bestehende Aufgaben können geändert werden, sofern noch kein Studierender mit der Bearbeitung begonnen hat. Das Fälligkeitsdatum kann jederzeit geändert werden.

|A13| Einladen von Studierenden zu einer Aufgabe

- **Akteure:** Dozierende
- **Beschreibung:** Dozierende können alle Studierenden eines Klassenraums zu einer neuen Aufgabe einladen. Studierende erhalten eine E-Mail über die Einladung.

|A14| Akzeptieren von Aufgaben

- **Akteure:** Studierende
- **Beschreibung:** Studierende können eine Aufgabe annehmen und beginnen, daran zu arbeiten. Für die Bearbeitung wird ein Fork des Vorlagen-Repositorys innerhalb einer Untergruppe des Klassenraums erstellt. Das Annehmen kann über einen Link in der E-Mail-Einladung erfolgen, oder alternativ können die Aufgabe auch über die Klassenraum-Übersicht (siehe A10) angenommen werden. Die Studierenden erhalten Schreibrechte, solange die Fälligkeit der Aufgabe noch nicht erreicht wurde.

|A15| Einsehen und Bearbeiten von Projekten

- **Akteure:** Dozierende, Moderierende, Studierende
- **Beschreibung:** Studierende können den Code ihrer eigenen Projekte jederzeit einsehen und innerhalb der Fälligkeit auch bearbeiten. Optional dürfen Studierende auch die Projekte von anderen Teams oder Studierenden einsehen. Dozierende haben auf allen Projekten Lese- und Schreibrechte und Moderierende haben ausschließlich Leserechte.

Team- und Rollenverwaltung

|A16| Erstellung von Teams

- **Akteure:** Dozierende, Moderierende, Studierende
- **Beschreibung:** Teams können vom Dozierenden oder wahlweise auch von den Studierenden erstellt werden, um die Zusammenarbeit bei Aufgaben zu ermöglichen. Teams werden in GitLab als Untergruppe des Klassenraums repräsentiert.

|A17| Editierung von Teams

- **Akteure:** Dozierende, Moderierende
- **Beschreibung:** Bestehende Teams können umbenannt werden.

|A18| Beitreten zu einem Team

- **Akteure:** Studierende
- **Beschreibung:** Nach dem Beitritt zu einem Klassenraum kann der Studierende, sofern Teams aktiviert sind, sich einem Team zuweisen.

|A19| Zuweisen von Teams

- **Akteure:** Dozierende, Moderierende
- **Beschreibung:** Studierende können aus Teams entfernt oder in andere Teams verschoben werden.

|A20| Zuweisen von Rollen

- **Akteure:** Dozierende
- **Beschreibung:** Mitglieder des Klassenraums können andere Rollen im Klassenraum zugewiesen werden, sodass diese auch Moderationsaufgaben wahrnehmen können.

1.4.2 Optionale funktionale Anforderungen

Einige Anforderungen sind nicht für die minimale Umsetzung notwendig, aber dennoch ein wichtiger Bestandteil des Gesamtkonzeptes. Diese werden nach der Implementierung der funktionalen Mindestanforderungen umgesetzt.

Bewertungssystem

|K01| Übersicht über Bewertung aller Projekte einer Aufgabe

- **Akteure:** Dozierende, Moderierende
- **Beschreibung:** Dozierende oder Moderierende sehen die Bewertung der eingereichten Projekte einer Aufgabe. Die Bewertungen können maschinenlesbar exportiert werden.

|K02| Erstellen von manuellen Bewertungskriterien

- **Akteure:** Dozierende
- **Beschreibung:** Dozierende können eigene Bewertungskriterien mit einer maximalen Punktzahl für einen Klassenraum definieren und für verschiedene Aufgaben verwenden.

|K03| Editieren von manuellen Bewertungskriterien

- **Akteure:** Dozierende
- **Beschreibung:** Dozierende können die Bewertungskriterien für einen Klassenraum bearbeiten oder löschen.

|K04| Manuelle Kriterien für Aufgaben aktivieren

- **Akteure:** Dozierende
- **Beschreibung:** Definierte manuelle Bewertungskriterien eines Klassenraums können für Aufgaben aktiviert werden.

|K05| Manuelle Bewertung einer Aufgabe

- **Akteure:** Dozierende, Moderierende
- **Beschreibung:** Dozierende oder Moderierende können die Lösungen, jedes einzelnen teilnehmenden Teams beziehungsweise Studierenden einer Aufgabe manuell anhand der vorher festgelegten Kriterien mit Punkten bewerten.

|K06| Automatische Bewertung einer Aufgabe mit Tests

- **Akteure:** Dozierende
- **Beschreibung:** Automatische Tests für Aufgaben können aktiviert und konfiguriert werden. Als Vorlage für die möglichen Tests dient das Vorlage-Repository. Die ausgewerteten Tests sollen als Grundlage zur Bewertung mit Punkten verwendet werden. Der Dozent soll festlegen können, wie viele Punkte ein bestimmter erfolgreicher Test in der Auswertung ergeben soll.

|K07| Feedback-Branch zum Kommentieren vom Quellcode

- **Akteure:** Dozierende, Moderierende, Studierende
- **Beschreibung:** Ein Feedback-Branch mit erstelltem Merge-Request kann genutzt werden, damit Dozierende oder Moderierende, Kommentare direkt am Quellcode des Studierenden hinterlassen können.

1.4.3 Nicht-funktionale Anforderungen

Benutzbarkeit Die Benutzeroberfläche soll möglichst leicht verständlich sein, sodass Dozierende ohne technische Vorkenntnisse Klassenräume und Aufgaben erstellen und verwalten können. Außerdem soll eine konsistente Navigation und visuelle Struktur sicherstellen, dass Studierende und Dozierende gut mit dem System interagieren können.

Portabilität und Bereitstellung Da die Anwendung nicht ausschließlich für die Hochschule Flensburg entwickelt wird, sondern potenziell auch anderen Bildungseinrichtungen oder Entitäten als Open-Source-Lösung zur Verfügung gestellt werden soll, ist eine einfache Installation erstrebenswert.
Um dies zu erreichen, soll idealerweise ein Docker-Image bereitgestellt werden, um eine leichte und plattformunabhängige Bereitstellung der Anwendung zu ermöglichen.

1.4.4 Zusammenfassung

Erstellung von Klassenräumen Ein Dozierender soll die Möglichkeit haben, neue Klassenzimmer zu erstellen und Studierende zu diesem hinzuzufügen. Dabei sollen die Studierenden per E-Mail benachrichtigt werden und einen personalisierten Link erhalten, über den sie dem Klassenzimmer beitreten können. Nach dem Beitritt der Studierenden zum Klassenzimmer soll automatisch eine Gruppe in GitLab erstellt werden, in die die Studierenden hinzugefügt werden.

Erstellung von Teams Die Studierenden sollen die Möglichkeit erhalten, entweder den vom Dozierenden erstellten Teams beizutreten oder eigene Teams zu erstellen und diesen beizutreten, um die zukünftigen Aufgaben gemeinsam im Team zu bearbeiten. Darüber hinaus soll es auch möglich sein, in Einzelteams zu arbeiten. In GitLab soll währenddessen eine Untergruppe für jedes Team bereitgestellt und die entsprechenden Berechtigungen vergeben werden.

Erstellung von Aufgaben Der Dozierende soll die Möglichkeit haben, innerhalb eines Klassenzimmers eine beliebige Anzahl an Aufgaben zu erstellen und zu verwalten. Zudem soll es optional möglich sein, für jede Aufgabe einen Abgabetermin festzulegen.

Optional: Bewertungssystem Es sollen sowohl die Möglichkeit für ein automatisches als auch ein manuelles Bewertungssystem implementiert werden. Das automatische Bewertungssystem soll die automatische Überprüfung und Bewertung von eingereichtem Quellcode durch Tests ermöglichen. Parallel dazu soll das manuelle Bewertungssystem den Dozierenden die Flexibilität bieten, individuell die studentischen Leistungen zu bewerten und den Studierenden Feedback zu geben.

2 Methoden und Technologien

2.1 Definition der Architektur

Bereits in den ersten Planungen wurde diskutiert, in welcher Form und mit welcher Architektur die Anwendung realisiert werden sollte. Der Fokus lag dabei auf der Frage, ob eine direkte GitLab-Integration oder eine Standalone-Anwendung die geeignetere Lösung wäre. In diesem Abschnitt wird die letztlich getroffene Entscheidung für eine Standalone-Anwendung erläutert. Es werden die Struktur der Anwendung, die Interaktion über die REST-API sowie die Anbindung an eine Datenbank beschrieben.

2.1.1 Direkte Integration in GitLab: Vor- und Nachteile

Im Zuge der Entscheidungsfindung gab es viele Argumente für die Realisierung als Standalone-Variante, insbesondere im Vergleich zur direkten GitLab-Integration. Eine direkte Integration in GitLab hätte bedeutet, dass die Anwendung tief in die bestehende GitLab-Plattform eingebettet worden wäre, um Funktionen wie die Verwaltung von Kursen und Aufgaben sowie die Bewertung von Abgaben direkt in GitLab bzw. der GitLab-Oberfläche verfügbar zu machen. Diese Lösung hätte den Vorteil gehabt, eine nahtlose Benutzererfahrung in einer einzigen Plattform zu bieten, bei der alle notwendigen Funktionalitäten innerhalb von GitLab zugänglich wären.

Allerdings bietet GitLab nur begrenzte Erweiterungsmöglichkeiten. Eine tiefgreifende Integration hätte erhebliche Herausforderungen mit sich gebracht, da ein umfangreiches Verständnis des GitLab-Quellcodes, Anpassungen an der GitLab-Benutzeroberfläche und den internen Workflows notwendig gewesen wären. Darüber hinaus besteht das Risiko, dass zukünftige GitLab-Updates die integrierte Anwendung negativ beeinflussen oder gar unbrauchbar machen könnten, was zusätzliche Ressourcen für die Wartung erfordert hätte. Alternativ hätte man sich auf eine ältere GitLab-Version festlegen müssen, was zukünftige Updates erschwert hätte.

2.1.2 Standalone-Anwendung über die GitLab-API

Angesichts dieser Nachteile wurde die Entscheidung getroffen, die Anwendung als eigenständige Plattform zu konzipieren, welche über die GitLab-API mit GitLab kommuniziert und interagiert. Die Anwendung kann dabei über die gut dokumentierte REST-API von GitLab (Version 4) realisiert werden und ermöglicht gleichzeitig eine flexible und unabhängige Weiterentwicklung. Die GitLab-API ist versioniert und wird langfristig unterstützt (siehe REST-Kompatibilität), sodass die Anwendung unabhängig von GitLab-Updates in einem eigenen Release-Zyklus weiterentwickelt werden kann. Dadurch lassen sich potenzielle Inkompatibilitäten bei künftigen GitLab-Versionen minimieren, sodass die Anwendung auch bei Änderungen von GitLab überwiegend anpassungsfrei weiterverwendet werden kann.

2.1.3 Entscheidungsgründe für eine Standalone-Anwendung

Die Wahl der Umsetzung als Standalone-Anwendung basiert nicht nur auf den technischen Herausforderungen einer direkten Integration. Die Trennung von GitLab ermöglicht eine größere Flexibilität bei der Erweiterung der Funktionalität und erleichtert die langfristige Wartung und Weiterentwicklung. Des Weiteren ermöglicht diese Entscheidung langfristig, dass die Anwendung nicht nur mit GitLab, sondern auch bei Bedarf mit anderen Versionskontrollsystemen verwendet werden könnte.

2.1.4 Monolithische Architektur

Zur Vereinfachung der Bereitstellung, Entwicklung und Wartung wurde für die Standalone-Anwendung eine monolithische Architektur gewählt. Frontend und Backend werden logisch voneinander getrennt. Das Backend hat die Aufgabe, die eigentliche Geschäftslogik zu realisieren und die Kommunikation mit dem Frontend erfolgt über eine eigene REST-Schnittstelle (mit OpenAPI-Spezifikation). Das Frontend wird direkt vom Backend-Prozess bereitgestellt, was das Auftreten von CORS-Problemen verhindert und sicherstellt, dass die Version des Frontends stets kompatibel mit der aktuellen Backend-Version ist.

Dieser Ansatz ermöglicht eine konsistente und integrierte Entwicklungsumgebung, in der sämtliche Änderungen am Backend und Frontend synchronisiert sind und in einem einzigen Vorgang bereitgestellt werden können. Dadurch wird die Komplexität reduziert und auch der Wartungsaufwand für die Verwender minimiert.

2.1.5 REST-API und GitLab-Integration

Die REST-API der Anwendung ist vielseitig einsetzbar und kann unabhängig vom Frontend verwendet werden, um Änderungen oder Abfragen durchzuführen. Die Integration des Backends mit GitLab erfolgt über die GitLab-API und die Authentifizierung erfolgt über eine GitLab-OAuth-Schnittstelle. Dadurch ist es möglich, im Namen der Benutzer Aktionen durchzuführen, wie etwa das Erstellen von Gruppen, um das Konzept der »Klassenräume« in GitLab abzubilden.

2.1.6 Datenbankanbindung

Zusätzlich zur GitLab-Anbindung benötigt die Anwendung eine eigenständige Datenbank. Diese ermöglicht die Speicherung von anwendungsspezifischen Daten, die nicht direkt in GitLab hinterlegt werden können, aber für die Umsetzung der Anwendung notwendig sind.

2.2 Technologieauswahl

Bei der Technologiewahl wurden die Anforderungen an Performance und Teamfähigkeit berücksichtigt. Durch die Überprüfung verschiedener Möglichkeiten und eine demokratische Abstimmung im Team wurde die geeignete Technologie für das Backend und das Frontend ausgewählt. Dabei lag der Fokus auf etablierten und weit verbreiteten Technologien, die sich durch ihre Flexibilität, Erweiterbarkeit und Einsteigerfreundlichkeit auszeichnen.

2.2.1 Backend

Für die Implementierung des Backends wurde sich für die Programmiersprache Go (Golang) entschieden. Go hat sich in den letzten Jahren in der Webentwicklung etabliert und ist besonders bekannt für seine einfache Syntax, hohe Performance und parallele Ausführbarkeit. Im Vergleich zu Alternativen wie Node.js oder Python verfügt Go über eine effiziente Speicherverwaltung und ermöglicht es, hochperformante Anwendungen zu entwickeln, die auch bei hoher Last stabil bleiben.

Die Standardbibliothek von Go ist sehr umfangreich und bietet integrierte Lösungen für viele gängige Anwendungsfälle, wie z.B. die Implementierung von OAuth2-Authentifizierung. Besonders hervorzuheben ist die Fähigkeit von Go, plattformübergreifende, statisch gelinkte Binaries zu erstellen, die ohne Abhängigkeiten auf verschiedenen Betriebssystemen laufen können. Dies reduziert den Aufwand für die Bereitstellung und macht Go zu einer idealen Wahl für die Entwicklung einer skalierbaren und robusten Backend-Architektur.

Als Webframework wurde `GoFiber` ausgewählt, welches Ähnlichkeiten zum bekannten `Express.js`-Framework in Node.js aufweist, jedoch die Performance von Go optimal nutzt. Es bietet eine Vielzahl an nützlichen Middlewares, die die Entwicklung beschleunigen und standardisieren.

Für die Datenbankanbindung wurde das ORM-Framework `GORM` verwendet, das eine einfache Abstraktion über SQL-Datenbanken bietet. Die Verwendung von `GORM` ermöglicht es, Go-Structs direkt mit Datenbanktabellen zu verknüpfen und Änderungen an diesen Strukturen effizient zu synchronisieren. Diese Integration erleichtert den Umgang mit der Persistenzschicht und beschleunigt die Entwicklung.

2.2.2 Frontend

Im Bereich des Frontends wurde `React` ausgewählt, ein von Facebook bzw. Meta Platforms entwickeltes Framework, das sich als eine der führenden Lösungen für die Entwicklung dynamischer, komponentenbasierter Webanwendungen etabliert hat. `React` ist nicht nur flexibel und leistungsstark, sondern bietet auch eine große Gemeinschaft und zahlreiche Lernressourcen, die es dem Team ermöglichen, schnell produktiv zu werden. Die große Verbreitung von `React` sorgt

zudem für eine ständige Weiterentwicklung und eine Fülle an verfügbaren Bibliotheken und Erweiterungen.

Für die Client-Side-Routing-Lösung wurde `TanStack-Router` verwendet, da dieser vollständig in TypeScript integriert ist und im Vergleich zum herkömmlichen `React`-Router eine verbesserte Typensicherheit bietet. Dies trägt zur Vermeidung von Fehlern bei, da etwa fehlende Parameter oder falsche Links bereits zur Entwicklungszeit erkannt werden.

Die Kommunikation zwischen Frontend und Backend erfolgt asynchron über `TanStack-Query`, in Kombination mit `Swagger-Codegen`, welches auf Basis einer OpenAPI-2.0-Spezifikation einen JavaScript-Client generiert. Dies reduziert potenzielle Kommunikationsfehler und sorgt für Konsistenz zwischen den Schnittstellen.

Für das Styling wurde die Komponentensammlung von `shadcn/ui` in Kombination mit dem `Tailwind`-CSS-Framework verwendet. Tailwind ermöglicht ein hochgradig anpassbares Design durch vordefinierte Utility-Klassen, während `shadcn/ui` eine moderne und kohärente visuelle Sprache bietet.

2.3 Entwicklungsumgebung

Die Entwicklungsumgebung ist ein zentraler Bestandteil des Softwareentwicklungsprozesses, da sie den Rahmen für die Implementierung, das Testen und das Deployment der Anwendung vorgibt. Für dieses Projekt wurde eine moderne, containerbasierte Entwicklungsumgebung gewählt, die durch die Verwendung von Docker und einer CI-/CD-Pipeline die Effizienz und Zuverlässigkeit der Entwicklungsprozesse sicherstellt. Durch den Einsatz dieser Technologien kann das Team sicherstellen, dass die Anwendung konsistent über verschiedene Umgebungen hinweg entwickelt, getestet und bereitgestellt wird.

2.3.1 Prototyping

Für die Entwicklung der Benutzeroberfläche (UI) wurde das Prototyping-Tool Figma verwendet. Figma diente am Anfang des Projektes für die Planung der Benutzererfahrung (UX) und im späteren Verlauf auch als grobe Vorlage für das Design der UI.

Im Verlauf des Projektes wurden zwei Prototypen erstellt:

1. **Konzeptprototyp**
   Ein erster LoFi-Prototyp für das Konzept wurde erstellt.

2. **UI-Prototyp**
   Ein weiterer Prototyp wurde als grobe Vorlage für ein Design auf der `React`-Komponentenbibliothek `shadcn/ui` erstellt.

2.3.2 Docker

Docker ist eine Containerisierungsplattform, die es ermöglicht, Anwendungen zusammen mit ihren Abhängigkeiten in isolierten Umgebungen zu betreiben. In diesem Projekt wird Docker verwendet, um sowohl das Frontend als auch das Backend in einem einzigen Container-Image zu bündeln. Dies erleichtert die Verteilung und das Deployment der Anwendung erheblich, da das Image auf jeder Maschine, die Docker unterstützt, unabhängig von der spezifischen Systemkonfiguration ausgeführt werden kann.

Ein besonderer Vorteil von Docker liegt in der Isolierung der Anwendungsumgebung. Dies bedeutet, dass alle notwendigen Abhängigkeiten, Bibliotheken und Konfigurationen innerhalb des Containers definiert sind. Dadurch wird die Wahrscheinlichkeit von »works on my machine«-Problemen minimiert, welche oft durch Unterschiede in den Entwicklungs- oder Ausführungsumgebung verursacht werden.

2.3.3 Konfiguration und E-Mail

Die Konfiguration der Anwendung erfolgt hauptsächlich über `.env`-Dateien, die für die Festlegung von Umgebungsvariablen genutzt werden. Diese Dateien enthalten spezifische Konfigurationsparameter für die verschiedenen Aspekte der Anwendung und ermöglichen eine einfache Anpassung der Umgebung, ohne den Quellcode ändern zu müssen. In der Entwicklungsumgebung können beispielsweise unterschiedliche Datenbankverbindungen, E-Mail-Konfiguration, API-Schlüssel, URL-Konfigurationen oder andere Konfigurationsparameter darüber definiert werden.

In der Entwicklungsumgebung wird außerdem `Mailpit` verwendet, ein Werkzeug, das in der Entwicklungsphase alle vom System gesendeten E-Mails sammelt. Dies ermöglicht es den Entwicklern, E-Mail-Workflows zu testen, ohne dass echte E-Mails gesendet werden müssen. `Mailpit` fungiert dabei als interner E-Mail-Server, der alle in der Entwicklungsumgebung generierten E-Mails speichert und darstellt.

2.3.4 CI-/CD-Pipeline

Um eine kontinuierliche Integration und Bereitstellung (CI/CD) zu ermöglichen, wurde eine CI-/CD-Pipeline implementiert, die den gesamten Entwicklungsprozess automatisiert und die Qualität der Software durch regelmäßige Tests und Linting verbessert. Diese Pipeline umfasst insgesamt sechs Jobs, die in vier Stufen organisiert sind. Die ersten beiden Stufen, Test und Lint, werden bei jedem Commit auf den GitLab-Server ausgeführt, während die vollständige Pipeline nur nach einem erfolgreichen Merge in den `Development`-Branch durchgeführt wird. Die Stufen sind wie folgt strukturiert:

- **Test** In dieser Phase werden automatisierte Softwaretests sowohl für das Frontend als auch für das Backend durchgeführt. Diese Tests sollen sicherstellen, dass alle Funktionen der Anwendung wie erwartet arbeiten und dass keine neuen Fehler durch Änderungen im Code eingeführt wurden. Darüber hinaus wird die Testabdeckung (Coverage) ermittelt und in GitLab zur Verfügung gestellt. Dies bietet dem Team wertvolle Einblicke in die Bereiche des Codes, die getestet wurden, und jene, die möglicherweise noch ungetestet sind.

- **Lint** Während der Lint-Phase wird der Code auf mögliche Fehler und Verstöße gegen definierte Stilkonventionen überprüft. Das Frontend wird mit TypeScript kompiliert, um sicherzustellen, dass es fehlerfrei ist, und anschließend mit `ESLint` auf Stilkonformität überprüft. Für das Backend wird das Tool `golangci-lint` verwendet, um den Go-Code zu analysieren. Die Ergebnisse der Lint-Phase werden als Artefakte gespeichert und in GitLab visualisiert. Dies ermöglicht es den Entwicklern, den Code auf potenzielle Schwächen zu prüfen und gegebenenfalls Korrekturen vorzunehmen.

- **Build** In dieser Phase wird das Docker-Image der Anwendung erstellt. Dabei kommt eine Docker-in-Docker-Umgebung zum Einsatz, um das Image innerhalb eines Containers zu bauen. Bei jedem Build-Prozess wird eine neue App-Version erstellt, die auf der aktuellen Commit-Referenz basiert. Diese Version wird anschließend in eine Docker-Registry hochgeladen, damit sie für das Deployment zur Verfügung steht. Diese Strategie stellt sicher, dass die neueste Version der Anwendung immer bereit zur Auslieferung ist.

- **Deploy** In der letzten Stufe der Pipeline wird das neu erstellte Docker-Image auf dem Staging-Server bereitgestellt. Dazu wird eine Verbindung über SSH zum Staging-Server hergestellt, das neue Image aus der Registry geladen und der laufende Docker-Container mit der neuesten Version der Anwendung neu gestartet. Diese Phase ermöglicht es, den aktuellen Stand der Entwicklung in einer simulierten Produktionsumgebung zu testen und sicherzustellen, dass die Anwendung auch in einem realen Szenario einwandfrei funktioniert.

3 Architektur und Implementierung

3.1 Konzeptionierung und Systemzerlegung

In diesem Abschnitt erfolgt eine grobe Betrachtung der einzelnen Anwendungskomponenten. Die vollständige Implementierung kann dem beigefügten Quellcode entnommen werden.

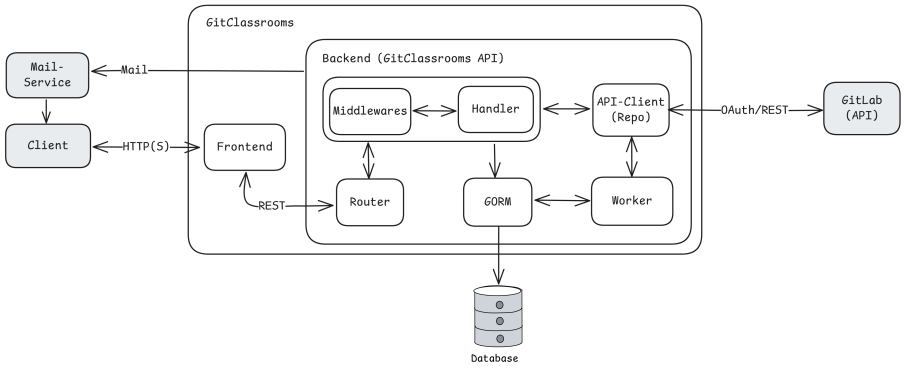

Abbildung 1: Systemzerlegung der Anwendung »GitClassrooms«

In der »GitClassrooms«-Architektur gibt es mehrere zentrale Komponenten: das Frontend, das Backend und die GitLab bzw. die GitLab-API. Die Kommunikation zwischen diesen Komponenten erfolgt größtenteils über standardisierte Schnittstellen wie REST-APIs und asynchrone Aufrufe.

Kommunikation zwischen Client, Frontend, Backend und GitLab

Client-Frontend-Kommunikation Der Client (Nutzer) interagiert über das Frontend mit der Anwendung. Hierbei werden HTTP(S)-Anfragen gesendet, die auf dem Backend REST-Endpoints aufrufen.

Frontend-Backend-Kommunikation Die Datenübertragung zwischen Frontend und Backend erfolgt über REST-API-Aufrufe. Die APIs sind `OpenAPI-2.0`-kompatibel, was eine standardisierte und dokumentierte Schnittstelle sicherstellt. Diese Schnittstelle erlaubt es, verschiedene Endpunkte des Backends anzusprechen, z. B. um Daten zu Klassenräumen oder Aufgaben zu laden.

Backend-GitLab-Kommunikation Das Backend kommuniziert mit GitLab über die GitLab-API. Hierbei werden als Authentifizierungsinformationen OAuth-Tokens genutzt, um über den GitLab Account des Nutzers Aktionen durchzuführen. So z.B. das Erstellen von Gruppen (Klassenräumen), Repositories (Aufgaben) und die Verwaltung von Rechten und Rollen. Für die Verwaltung

von GitLab-Gruppen werden zur Authentifizierung »Group Access Tokens« (siehe Seite 29) verwendet.

Austauschbarkeit des Frontends Durch den Einsatz einer klar strukturierten API und eines getrennten Frontends ist es möglich, das Frontend bei Bedarf auszutauschen. Dies wird durch den Einsatz von OpenAPI-Spezifikationen vereinfacht, da jede Änderung an der API leicht in ein neues Frontend integriert werden kann. So könnte beispielsweise ein neues Web- oder Mobile-Frontend entwickelt werden, ohne die bestehende Backend-Logik verändern zu müssen.

3.2 Klassenräume mit GitLab

Dieses Kapitel beschreibt die Implementierung der »GitClassrooms«-Struktur in GitLab. Es wird erläutert, wie Klassenräume, Aufgaben, Teams und Rechte innerhalb von GitLab abgebildet werden. Darüber hinaus wird der grobe Ablauf bei der Erstellung von Klassenräumen dargestellt.

3.2.1 Struktur

Innerhalb dieser Anwendung werden die folgenden Strukturen verwendet:

- Nutzende können beliebig viele Klassenräume (`Classrooms`) erstellen bzw. Mitglied von beliebig vielen Klassenräumen sein.
- Klassenräume können beliebig viele Aufgaben (`Assignments`) beinhalten. Eine Aufgabe repräsentiert ein bestimmtes öffentliches Vorlage-Repository, welches vorher in GitLab vom Erstellenden vorbereitet werden muss.
- `Teams` können in einem Klassenraum definiert werden, damit mehrere Studierende Aufgaben in einem gemeinsamen Projekt bearbeiten können.
- Für jede vom Team oder Studierenden akzeptierte Aufgabe wird ein Projekt (`Project`) angelegt, welches die Repositories in GitLab repräsentiert.
- Jedes dieser Projekte kann einzeln bewertet werden, wobei die Bewertung dann für alle Team-Mitglieder gültig ist.

3.2.2 Rollen in der Anwendung

- `Creator/Owners` Erstellende des Klassenraums sind die höchste Rolle des Klassenraums. Sie dürfen alle Einstellungen des Klassenraums verwalten, Mitglieder einladen, Mitglieder verwalten, Teams verwalten, Aufgaben verwalten, Mitglieder zu Aufgaben einladen, alle Projekte einsehen und Bewertungen durchführen. Sie haben immer vollständigen Zugriff auf den Quellcode aller Mitglieder.
- `Moderators` Moderatoren sind eine unterstützende Rolle. Sie können Mitglieder einladen, Teams verwalten, alle Projekte einsehen und Bewertungen durchführen.

- `Students` Studierende sind die Standardrolle für Mitglieder. Diese können Aufgaben einsehen, Einladungen zu Aufgaben akzeptieren, Teams erstellen und beitreten. Außerdem können diese, falls aktiviert, Projekte von anderen Studierenden einsehen. Innerhalb der Bearbeitungszeit können diese an Aufgaben arbeiten und jederzeit ihren Quellcode einsehen.

3.2.3 Umsetzung in GitLab

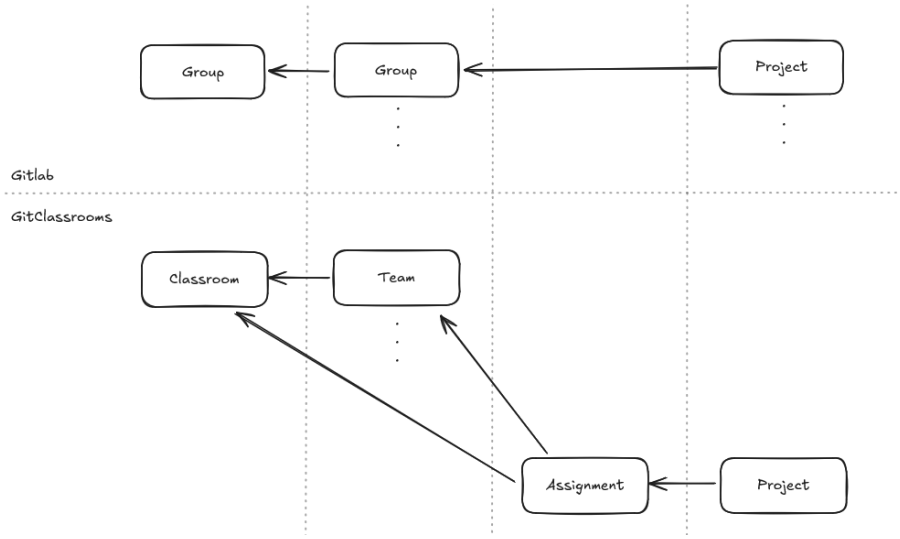

Abbildung 2: Abbildung der »GitClassroom«-Entitäten auf GitLab-Entitäten

Klassenräume (`Classrooms`) sowie Teams werden in GitLab als Gruppen abgebildet. Der `Creator` des Klassenraums ist auch der Besitzer der Klassenraum-Gruppe in GitLab. Teams werden als Untergruppen der Klassenraum-Gruppen erstellt. Sofern keine Teams aus mehreren Personen gewünscht sind, wird pro `Student` eine Team-Untergruppe mit dem Namen des `Student` in der Klassenraum-Gruppe angelegt. `Students` können von `Owners` bzw. `Moderators` des Klassenraums in diesen eingeladen werden. Diese Einladung erfolgt per E-Mail. Über einen Einladungslink kann in Kombination mit einem gültigen Account der GitLab-Instanz dem Klassenraum beigetreten werden.

Aufgaben (`Assignments`) referenzieren jeweils ein öffentliches Projekt-Repository (des `Creators`) als Vorlage. Für jede Aufgabe werden die `Students` bzw. `Teams` separat eingeladen. Sofern diese Einladung akzeptiert wird, werden für jedes `Team` oder jeden `Student` ein Projekt-Repository als Fork des Vorlage-Repositories erstellt. Dieses Projekt-Repository wird unterhalb der jeweiligen Untergruppe des `Teams` oder `Student` angelegt.

Abbildung der Rechte in GitLab `Students` eines Klassenraums sind Mitglieder der jeweiligen GitLab-Gruppe. In GitLab werden Rechte an Untergruppen vererbt, daher erhalten `Students` auf der Ebene der Klassenraum-Gruppe nur `Guest`-Berechtigungen, damit sie standardmäßig nicht die Untergruppen bzw. Projekte anderer Teams einsehen können. Wenn den `Students` dies möglich sein soll, muss dies bei der Erstellung des Klassenraumes entsprechend eingestellt werden. In diesem Fall erhalten sie `Reporter`-Berechtigungen und können dadurch den Code aller anderen Untergruppen einsehen. Der `Creator` eines Klassenraums erhält immer `Owner`-Berechtigungen für die entsprechende GitLab-Gruppe.

Einzelne `Students` können zu `Moderators` befördert werden, wodurch sie ebenfalls `Reporter`-Berechtigungen innerhalb der Klassenraum-Gruppe erhalten und dadurch Projekte aller einsehen können. Diese Funktionalität soll beispielsweise die Unterstützung durch Tutoren innerhalb eines Klassenraums ermöglichen. `Moderators` können nicht selbst an der Bearbeitung von Aufgaben teilnehmen und können auch nicht Teil einer `Teams`-Gruppe sein.

In den `Teams`-Gruppen sind jeweils nur die entsprechenden Mitglieder enthalten. Innerhalb eines Projekts erhalten alle Team-Mitglieder zunächst `Developer`-Berechtigungen, damit sie die Aufgabe bearbeiten können. Sobald die Deadline der Aufgabe erreicht wird, werden die Studierende auf `Reporter`-Berechtigungen herabgestuft, sodass sie keine Änderungen mehr am Projekt vornehmen können.

3.3 Struktur des Projekts

Die Projektstruktur des Projekts ist in verschiedene Verzeichnisse und Dateien unterteilt, die die verschiedenen Komponenten und Funktionen der Anwendung widerspiegeln. Für den besseren Einstieg in das Projekt wird die Basisstruktur des Projekts beschrieben.

```
git-classrooms/
    code_gen/
    config/
    controller/api
    controller/auth
    docs/
    frontend/
    model/
    repository/
    router/
    script/
    utils/
    worker/
    wrapper/
```

code_gen/ Dies Verzeichnis enthält Hilfsskripte für das Backend, welche für die Codegenerierung benötigt werden. Aktuell befindet sich dort der Code, der typsichere Queries auf Basis von `GORM/Gen` generiert, welche für den Datenbankzugriff im Backend verwendet werden.

config/ Im Verzeichnis `config/` werden Strukturen definiert, die die Konfiguration verschiedener Komponenten des Backends repräsentieren. Dabei wird auf das Go-Paket `caarlos0/env` zurückgegriffen, welches die automatische Konvertierung von Environment-Variablen in Go-Strukturen ermöglicht. Diese Vorgehensweise erlaubt es, Konfigurationsparameter in der Anwendung zu verwenden, ohne explizite manuelle Zuordnungen zwischen Variablen und Strukturen vorzunehmen.

controller/api Dieses Verzeichnis enthält die API-Handler und Middlewares des Backends. Jeder Handler wird in einer dedizierten Datei definiert, um die Übersichtlichkeit zu gewährleisten.

controller/auth Das `/auth`-Verzeichnis enthält die Handler und Authentifizierungslogik des Backends und den Authentifizierungs-Controller für die OAuth2-Authentifizierung mit GitLab bereit.

docs/ Im `docs/`-Verzeichnis befinden sich unter anderem Anleitungen für das Development-Setup, um neue Entwickler in das Entwicklungs-Setup einzuführen. Darüber hinaus wird in diesem Verzeichnis die `swagger.yaml`-Datei

generiert, welche die OpenAPI-Spezifikation (Swagger) für die Backend-API-Dokumentation enthält. Diese Datei dient der automatisierten Generierung von API-Dokumentationen und ermöglicht die Visualisierung und Interaktion mit den API-Endpunkten.

frontend/ Im Verzeichnis `frontend/` befindet sich der Quellcode für die Frontend-Anwendung.

model/ Im Verzeichnis `model/` befinden sich die ORM-Structs (Object-Relational Mapping). Auf Basis dieser Structs erstellt `GORM` automatisch die entsprechenden Datenbanktabellen. Die hier enthaltenen Strukturen repräsentieren die Hauptentitäten der Applikation und dienen als zentrale Datenmodelle für alle Operationen, die Datenbankinteraktionen erfordern.

repository/ Im Verzeichnis `repository/` sind die Schnittstellen zu externen Systemen implementiert. Dazu gehören unter anderem das GitLab-Repository, welches die Interaktionen mit der GitLab-API handhabt, sowie ein Repository für den Versand von E-Mail-Benachrichtigungen. (Siehe unter anderem Seite 33).

script/ Das Verzeichnis `script/` enthält nützliche Shell- und PowerShell-Skripte. Diese Skripte erleichtern das Starten der lokalen Entwicklungs-Umgebung sowie die automatisierte Generierung des Swagger-Clients für das Frontend.

utils/ Das Verzeichnis `utils/` enthält eine Sammlung von Hilfsfunktionen, die in verschiedenen Teilen der Anwendung wiederverwendet werden. Diese Funktionen dienen der Abstraktion und Modularisierung häufiger Operationen, um den Code lesbarer und wartbarer zu gestalten. Zu den bereitgestellten Funktionen gehören unter anderem Operationen auf Slices, Caching-Mechanismen und die Erstellung von CSV-Reports für Bewertungen.

worker/ Im Verzeichnis `worker/` sind die Hintergrunddienste implementiert, die beim Start der Anwendung in Go-Routinen ausgeführt werden. Diese Worker führen in regelmäßigen Abständen bestimmte Aufgaben (Work-Funktionen) aus. Ein zentraler Worker ist dafür verantwortlich, abgelaufene Aufgaben automatisch zu schließen. Ein weiterer Worker sorgt für die Synchronisation zwischen GitLab und der Applikation, insbesondere für den Fall, dass Änderungen direkt in GitLab vorgenommen wurden und nicht über diese Anwendung.

wrapper/ Hier befinden sich Wrapper-Klassen für den Umgang mit Kontext- und Session-Management. Es werden zentrale Typen bereitgestellt, die die Fiber-Session und den Fiber-Context durch Struct Embedding umschließen. Diese Wrapper ermöglichen es, typsichere Funktionen bereitzustellen, wodurch die Nutzung von `Get` und `Set`, die sonst ein `interface` zurückgeben und eine Laufzeit-Typprüfung erfordern, vermieden wird.

3.4 Backend

Dieses Kapitel beschreibt einige wichtige Aspekte über das Backend von »GitClassrooms«. Für einen vollständigen Einblick sollte der Quellcode konsultiert werden.

3.4.1 Routing

Router (`router.go`) Die Routen-Konfiguration wird in der Datei `router.go` definiert, welche alle relevanten Endpunkte und deren zugehörige Controller bzw. Handler enthält.

Der Router nutzt verschiedene Middlewares, welche beispielsweise für den CSRF-Schutz und Logging verwendet werden. Es gibt separate Routen für authentifizierte Benutzer (unter `/api`), die auf bestimmte Ressourcen wie Klassenräume (`Classrooms`), Aufgaben (`Assignments`) und Projekte (`AssignmentProjects`) zugreifen können. Weitere Middlewares schränken den Zugang auf bestimmte Nutzergruppen ein und granulieren die Authentifizierung weiter.

Die wesentlichen Bestandteile des Routers umfassen:

- Setup der Middlewares: Globale Middlewares wie Session-Handling und CSRF-Schutz werden hier initialisiert.
- Die Definition der Routen für authentifizierte API-Zugriffe mit ihren zugeordneten Handlern/Controllern.
- Frontend-Handling: Statische Dateien werden über einen speziellen Endpunkt ausgeliefert, und alle nicht erkannten Routen leiten auf die Haupt-Frontend-Seite, eine Single Page Application, um.

Middlewares Middlewares fungieren als Zwischenschicht, die Anfragen abfängt und vor der eigentlichen Verarbeitung in den Controllern relevante Daten bereitstellt oder bestimmte Überprüfungen durchführt. Die Daten sind den Controllern über den Kontext der Anfrage zugänglich. Die Anwendung verfügt über mehrere Middlewares, die an spezifische Routen gebunden sind und den darunterliegenden Controllern die benötigten Informationen bereitstellen. Zum Beispiel löst die `Classroom Middleware` das URL-Segment der `ClassroomID` direkt in den entsprechenden Klassenraum auf und stellt diesen allen darunterliegenden Controllern bereit. Zudem übernehmen Middlewares wie die `Archived Middleware` Validierungen und gestatten so beispielsweise nur GET-Anfragen für archivierte Klassenräume.

Controller/Handler Sie sind für die Verarbeitung der Anfragen an die Backend-API verantwortlich. Über den Kontext können sie auf die für die Anfrage relevanten Daten sowie den GitLab-Client zugreifen. Innerhalb der Controller wird die Anfrageverarbeitung gesteuert, wobei sowohl `GORM` für die Datenbankinteraktionen als auch der GitLab-Client für die Kommunikation mit GitLab genutzt werden.

OpenAPI-Spezifikationen Um eine OpenAPI-Spezifikation zu erstellen, wird das Tool Swag genutzt. Dieses Tool erlaubt es, durch Kommentare direkt im Go-Code über jedem API-Handler den jeweiligen Endpunkt zu beschreiben. Diese Dokumentationskommentare enthalten Referenzen auf Go-Strukturen (Structs), die als Response- bzw. Requestschemas in der API-verwendet werden. Swag parst diese Informationen und generiert daraus automatisch eine OpenAPI-Dokumentation.

Für das Frontend wird auf Basis dieser Spezifikation mit dem Tool swagger-codegen ein Axios-Client generiert. Dieser Client enthält sowohl die Typen als auch die Routen, die zur Kommunikation mit dem Backend benötigt werden. Ein wesentlicher Vorteil dabei ist, dass Änderungen an Backend-Endpunkten automatisch in den Frontend-Client übernommen werden. Sollten sich zum Beispiel die Parameter eines Endpunkts ändern, wird dies direkt in den generierten Typen und Routen reflektiert. Das reduziert den manuellen Aufwand bei Anpassungen und stellt sicher, dass das Frontend stets konsistent mit der API bleibt.

3.4.2 Authentifizierung

Die Authentifizierung der in der Anwendung bzw. die Anmeldung der Benutzer erfolgt über das OAuth2-Protokoll, das GitLab als Authentifizierungs-Anbieter verwendet. Der `OAuthController` steuert den gesamten Prozess der Benutzeranmeldung, Token-Verwaltung und Sitzungsvalidierung.

OAuth2 eignet sich insbesondere für diese Anwendung, da es eine sichere und standardisierte Methode bietet, Benutzer zu authentifizieren, ohne dass die Anwendung selbst Passwörter verwalten muss. Durch die Integration mit GitLab ermöglicht OAuth2 eine nahtlose Anmeldung mit bereits bestehenden GitLab-Konten. Dies ist ideal, da die Anwendung nur von bereits vorhandenen Nutzern GitLab-Instanz verwendet wird und die Authentifizierungslogik größtenteils von GitLab übernommen wird. Für die Implementierung wird auf die GO-interne Bibliothek `golang.org/x/oauth2` zurückgegriffen.

Der Prozess beginnt, indem der Benutzer einen CSRF-Token über einen Form-Post an den Sign-In-Endpunkt sendet. Anschließend wird der Benutzer zur GitLab-Anmeldeseite weitergeleitet, wobei eine PKCE-Challenge (Proof Key for Code Exchange) verwendet wird, um die Sicherheit zu erhöhen und CSRF-Angriffe zu verhindern. Nach erfolgreicher Authentifizierung in GitLab wird der Benutzer zur Route `/api/v1/auth/gitlab/callback` zurückgeleitet. In diesem Schritt wird der übermittelte Authentifizierungscode an GitLab gesendet, um ein `AccessToken` sowie ein `RefreshToken` zu erhalten. Diese Tokens werden in der Sitzung des Benutzers gespeichert und die Benutzerdaten werden erstellt oder, falls bereits vorhanden, in der Datenbank aktualisiert. Bei erfolgreicher Authentifizierung werden nur grundsätzliche User-Informationen wie User-ID, Name, E-Mail-Adresse und Avatar-URL in der Datenbank hinterlegt.

Die `AuthMiddleware` stellt sicher, dass bei jeder REST-Anfrage die Sitzung validiert wird. Zuerst wird der aktuelle `AccessToken` aus der Sitzung geholt, und das Ablaufdatum wird überprüft. Falls der Token abgelaufen ist, wird ein neuer `AccessToken` mit dem gespeicherten `RefreshToken` von GitLab angefordert. Anschließend wird ein neues GitLab-Repository-Objekt erstellt, das mit dem `AccessToken` arbeitet und in den Kontext der aktuellen Anfrage geladen wird. Dies ermöglicht es der Anwendung, im Namen des authentifizierten Benutzers Abfragen gegenüber GitLab durchzuführen.

3.4.3 Datenmanagment

Für die Datenspeicherung wird eine PostgreSQL-Datenbank eingesetzt, die über das ORM-Framework `GORM` angesprochen wird.

`GORM` bietet mehrere wesentliche Vorteile:

- **Schema-Definition:** Durch die Verwendung von Struct-Tags in Go-Datentypen kann direkt die Struktur der Datenbanktabellen definiert werden. Dies ermöglicht eine präzise Zuordnung von Go-Feldern zu Datenbankspalten, einschließlich Datentypen, Indizes und Standardwerten.
- **Automatische Migrationen:** `GORM` kann Tabellen in der Datenbank automatisch erstellen und aktualisieren, basierend auf den Go-Strukturen.
- **Relationen und Verknüpfungen:** `GORM` unterstützt die Definition und Verwaltung von Relationen zwischen Entitäten, wie z.B. Eins-zu-Eins, Eins-zu-Viele und Viele-zu-Viele-Beziehungen, und übernimmt die erforderlichen Join-Operationen.
- **Generierung von SQL-Queries:** `GORM` ermöglicht es, Datenbankabfragen auf hohem Niveau zu formulieren, indem es die SQL-Queries intern generiert. Dies reduziert die Notwendigkeit, SQL-Code direkt zu schreiben, und erleichtert die Arbeit mit Datenbankoperationen in Go.
- **Löschvorgänge:** Wenn eine Entität gelöscht wird, können alle zugehörigen verknüpften Entitäten automatisch gelöscht werden. Dies erfolgt durch die Konfiguration von Kaskadierungsregeln in den Beziehungseinstellungen. So werden beispielsweise beim Löschen eines Classrooms gleichzeitig auch alle verknüpften Teams und Assignments gelöscht.

  Anstelle des Kaskadierens kann auch festgelegt werden, dass beim Löschen eines Datensatzes die entsprechenden Fremdschlüssel in den verknüpften Tabellen auf NULL gesetzt werden. Diese Option kommt beispielsweise zum Einsatz, wenn ein Nutzer aus einem Team austritt. In diesem Fall wird in der Verbindungstabelle `UsersClassroom` die TeamID auf NULL gesetzt.

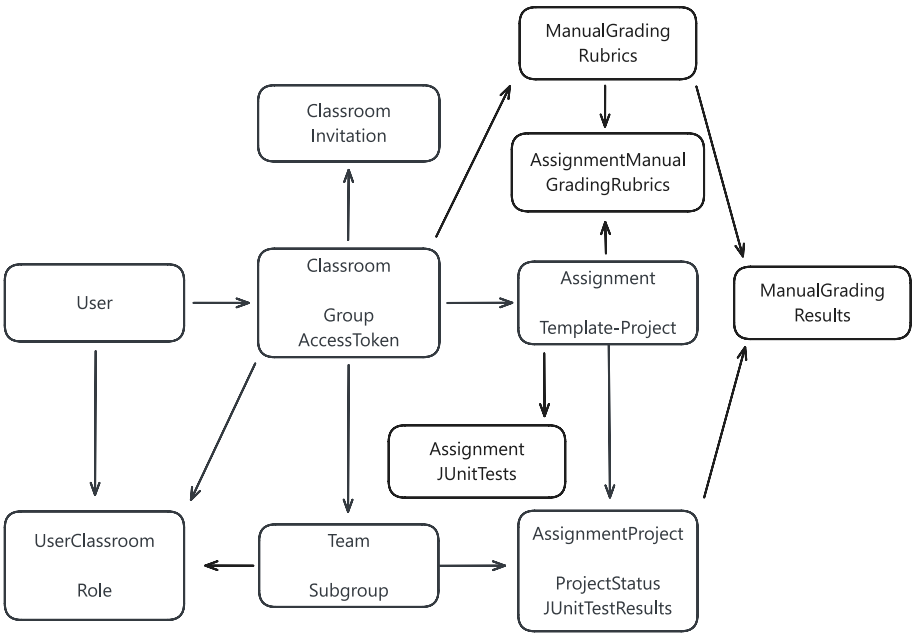

Abbildung 3: Abbildung der Datenbank Entitäten

In Abbildung 3 ist das grundlegende Datenbankschema dargestellt. Zur Wahrung der Übersichtlichkeit werden die Namen der Relationen genannt. Die genaue Struktur kann in den Go-Structs im Quellcode eingesehen werden (siehe `git-classrooms/model`).

Die `User`-Tabelle repräsentiert einen GitLab-Nutzer bzw. Nutzer dieser Anwendung, dessen Daten nach der Anmeldung mit GitLab erstellt und bei jeder weiteren Anmeldung aktualisiert werden, zudem wird ein Link zum GitLab-Avatar in der Tabelle `UserAvatar`. Das zentrale Element bildet die `Classrooms`-Tabelle, in der sich der GitLab-Group-Access-Token befindet. Dieser ermöglicht es, Aktionen wie das Erstellen von Subgruppen oder das Hinzufügen neuer Mitglieder durchzuführen, auch wenn der Owner nicht eingeloggt ist.

Die `ClassroomInvitations`-Tabelle speichert die Einladungen zu den Klassenräumen. Jede Einladung enthält Informationen über den eingeladenen Nutzer und den entsprechenden Klassenraum, für den die Einladung ausgestellt wurde. Zudem wird der Status der Einladung sowie die Gültigkeit festgehalten.

Die `UserClassroom`-Tabelle stellt die Verbindung zwischen `User` und `Classroom` dar und speichert zusätzlich die Rolle des Studierenden im entsprechenden Klassenraum. Die `Team`-Tabelle definiert die Teams innerhalb eines Klassenraums. Falls keine Teams aktiviert sind, wird dennoch für jeden Studierenden ein Team erstellt, jedoch ohne Änderungsmöglichkeiten über die API. Dies dient dazu,

unnötige Sonderfälle im Code zu vermeiden, wenn Teams deaktiviert sind.

Ein `Assignment` steht für eine Aufgabe innerhalb eines Klassenraums und enthält optional ein Ablaufdatum sowie die ID eines zugeordneten Template-Projekts. Das `AssignmentProject` stellt den Status einer Aufgabe für ein Team dar. Sobald eine Aufgabe angenommen wurde, wird die ID des geforkten Repositories gespeichert.

Bei automatisierten Tests innerhalb einer Aufgabe können diejenigen Tests ausgewählt werden, die Einfluss auf die Gesamtbewertung der Studierenden haben. Zudem können für jeden Klassenraum manuelle Bewertungskriterien erstellt und für bestimmte Aufgaben aktiviert werden. Die Ergebnisse dieser Bewertungen werden in der Tabelle `ManualGradingResults` gespeichert.

Migrationen Obwohl `GORM` den Vorteil automatischer Migrationen bietet, sind diese nicht immer ausreichend für den produktiven Einsatz. Insbesondere das Löschen von Feldern sowie Änderungen an Datenbank-Constraints werden nicht zuverlässig von der Auto-Migration abgedeckt. So können zum Beispiel Felder, die nicht mehr benötigt werden oder deren `NOT NULL`-Constraint entfernt werden soll, nicht automatisch gelöscht oder geändert werden, was manuelle Eingriffe in die Datenbank erforderlich macht.

Ein weiteres Problem der Auto-Migration besteht darin, dass es keine Funktionalität gibt, die SQL-Differenzen zwischen dem aktuellen Zustand der Datenbank und dem gewünschten Schema ausgibt. `GORM` führt sämtliche Änderungen direkt aus, ohne vorher eine Übersicht der geplanten Änderungen zu bieten.

Um dieses Problem zu bewältigen, wurde ein eigens entwickeltes Tool unter `code-gen/migrations` in die Anwendung integriert. Dieses Tool ermittelt die Differenzen zwischen dem Ist-Zustand der Datenbank und dem Soll-Zustand, der durch die `GORM`-Structs definiert wird, und erstellt darauf basierend SQL-Migrationsskripte.

Der Prozess nutzt Docker, um eine PostgreSQL-Instanz zu starten, in der sowohl die `GORM`-Migrationen als auch manuell durchgeführte Migrationen mit `Goose` verglichen werden. Anschließend wird das Docker-Image `supabase/migra` verwendet, um die notwendigen SQL-Befehle zu generieren, die erforderlich sind, um die Datenbank von den `Goose`-Migrationen auf den Stand der `GORM`-Migrationen zu aktualisieren. Diese Befehle werden im Standardausgang angezeigt. Sollte es keine Änderungen geben, wird das Tool mit Exit-Code 0 beendet, andernfalls mit Exit-Code 2. Dies erlaubt es, in zukünftigen CI-Prozessen zu prüfen, ob noch ausstehende Migrationen erstellt werden müssen.

3.4.4 GitLab-Repository

Für die Anbindung an GitLab wird ein Repository-Pattern in Kombination mit »Dependency Injection« als Abstraktionsschicht verwendet. Dies ermöglicht eine Austauschbarkeit der Implementierung, ohne dass eine direkte Abhängigkeit von der spezifischen API-Implementation besteht. Zur Anbindung der GitLab-API wird der GO-Gitlab Client verwendet, eine in GO geschriebene API-Bibliothek.

Für die Authentifizierung bei GitLab kommen sowohl OAuth-Tokens als auch Group-Access-Tokens zum Einsatz. Bei eingeloggten Nutzern wird zunächst der persönliche OAuth-Token verwendet, um mit GitLab zu kommunizieren. Sobald jedoch ein Nutzer einen Klassenraum erstellt, fordert die Anwendung im Namen des Erstellers von GitLab ein Group-Access-Token an. Mit diesem Token kann die Anwendung dann eigenständig als Bot in GitLab mit dem entsprechenden Klassenraum interagieren und somit auch Aktionen durchführen, ohne dass die Verfügbarkeit des ursprünglichen Erstellers erforderlich ist.

3.4.5 Bewertungssystem

In der Anwendung gibt es verschiedene Möglichkeiten zur Bewertung von Projekten, die sowohl manuelle als auch automatische Verfahren umfassen. Die folgenden Konzepte wurden in die Anwendung integriert:

Feedback-Branches Für jedes Projekt-Repository der Teams oder Studierenden wird automatisch ein Feedback-Branch erstellt, welcher auf dem ursprünglichen Code des Vorlage-Repositories basiert. Zudem wird eine Merge-Request vom Main-Branch zum Feedback-Branch erstellt, der ausschließlich von den Dozierenden oder Moderierenden genutzt werden kann, um direkt am eingereichten Quellcode Kommentare und Feedback zu hinterlassen. Dies erleichtert die Rückmeldung und ermöglicht eine effektive Kommunikation zwischen Lehrenden und Lernenden über den eingereichten Code.

Manuelle Bewertung Das manuelle Bewertungssystem erlaubt es Dozierenden, für jeden Klassenraum spezifische Bewertungskriterien, sogenannte Rubriken, festzulegen. Jede Rubrik kann mit einer maximalen Punktzahl versehen werden, die den Bewertungsrahmen für eine Aufgabe definiert. Diese Rubriken können/müssen für jede Aufgabe in einem Klassenraum aktiviert werden.

Bei der Bewertung der einzelnen Projekt-Repositories der teilnehmenden Teams oder Studierenden werden die für die Aufgabe aktivierten Rubriken angewendet. Für jedes Projekt-Repository kann eine individuelle Punktzahl pro Rubrik festgelegt werden. Die erzielten Punktestände werden in der Anwendungsdatenbank gespeichert und in der Anwendung für Dozierende oder Moderierende dargestellt. Dieses System ermöglicht eine detaillierte und flexible Bewertung, die auf den spezifischen Anforderungen der Aufgaben und den definierten Kriterien der Dozierenden basiert.

Automatisierte testgetriebene Bewertung Die automatisierte Bewertung basiert auf der CI/CD-Pipeline von GitLab und nutzt die Unit Test Reports als zentrales Element. Diese Reports werden als JUnit-XML-Formate generiert, welches ursprünglich aus dem Java-Testframework `JUnit` stammt, sich jedoch inzwischen als Standard für viele andere Test-Frameworks etabliert hat. GitLab wertet diese Reports automatisch aus, wenn sie als Report-Artefakte nach der Ausführung der Pipeline bereitgestellt werden.

Die Anwendung greift auf die Testergebnisse im Main-Branch des jeweiligen Projekt-Repositories zu. Dies ist über die Pipeline-API von GitLab möglich. Die Tests im Vorlage-Repository einer Aufgabe dienen als Grundlage und können pro Test in der Aufgaben-Konfiguration mit einer Punktzahl versehen werden. Später wird die Bewertung jedes Abgabe-Repositories automatisch auf Basis des letzten CI-/CD-Pipelines durchgeführt auf und die konfigurierten Punkte werden anhand der erfolgreichen Tests ermittelt.

Für die Nutzung muss ein Runner zur Ausführung der CI/CD-Pipeline zur Verfügung stehen, dabei handelt es sich idealerweise um einen Instanz-Runner oder alternativ um einen Group-Runner, welcher in GitLab für die Klassenraum-Gruppe definiert ist. Dieser muss in der Lage sein, die notwendigen Report-Artefakte zu erzeugen.

Ergebnissexport Die Anwendung bietet die Möglichkeit, die kombinierte manuellen und automatischen Bewertungen eines gesamten Klassenraums, einer spezifischen Aufgabe oder eines Teams als CSV-Datei zu exportieren. Darüber hinaus kann ein generiertes Shell-Script verwendet werden, um alle Projekt-Repositories für eine Aufgabe zu klonen. Diese Funktionen bieten den Dozierenden einen umfassenden Überblick über den Bewertungsprozess und erleichtern das Verwalten der Resultate.

3.4.6 Worker und Synchronisation

Um bestimmte Aufgaben automatisch im Backend ausführen zu lassen, wurden sogenannte Worker implementiert. Ein Worker ist eine eigenständige Go-Routine, die in regelmäßigen Abständen eine definierte Aufgabe ausführt. Folgende Worker wurden für die Anwendung entwickelt:

Fällige Aufgaben Worker (`DueAssignmentWork`) Dieser Worker prüft jede Minute die Fälligkeitsdaten aller offenen Aufgaben. Erreicht eine Aufgabe ihr Fälligkeitsdatum, wird sie automatisch geschlossen. Dabei werden die Schreibrechte der Studierenden in den entsprechenden GitLab-Projekten von `Developer` auf `Reporter` herabgestuft, um weitere Änderungen zu verhindern. Die Studierenden behalten jedoch weiterhin Lesezugriff, sodass die Projekte für Bewertungszwecke und die Einsicht in den eigenen Code zugänglich bleiben.

GitLab-Synchronisations-Worker (`SyncGitlabDBWork`) Dieser Worker führt einen regelmäßigen Datenabgleich zwischen der Anwendung und GitLab durch. Er erkennt beispielweise, wenn der Name eines Projekts in GitLab geändert wurde und nicht mehr mit dem in der Anwendung gespeicherten Namen übereinstimmt.

Die Synchronisation erfolgt über Polling, da GitLab die benötigten Webhooks für Gruppen nur in der Premium-Version anbietet (siehe Webhooks in GitLab-Dokumentation). Das Polling-Intervall wird über die Umgebungsvariable `GITLAB_SYNC_INTERVAL` festgelegt. Dabei muss darauf geachtet werden, dass die API-Ratenbeschränkungen von GitLab bei hoher Nutzung nicht überschritten werden (siehe API-Ratenbeschränkungen).

Im Rahmen der Synchronisierung werden alle unarchivierten Klassenräume sowie deren Teams und Aufgaben geprüft. Es wird überprüft, ob Namen, Beschreibungen und Mitglieder mit den Daten in GitLab übereinstimmen und ob entsprechende Objekte in GitLab gelöscht wurden.

Falls ein Klassenraum oder eine GitLab-Gruppe gelöscht wurde, wird dies in der Anwendung als potenziell gelöscht markiert. Wenn ein Nutzer mit einem als gelöscht markierten Klassenraum interagiert, überprüft eine Middleware, ob der Klassenraum tatsächlich gelöscht wurde. Falls nur der Zugriffstoken der Anwendung entfernt wurde, archiviert die Anwendung den Klassenraum. Ist der Klassenraum auch mit dem Token des Nutzers nicht zugänglich, wird er vollständig gelöscht, inklusive aller damit verbundenen Teams und Projekte. Entitäten, die nicht mehr erreichbar sind (z. B. Teams oder Projekte), werden direkt entfernt, da ein fehlerhafter Zugriffstoken in solchen Fällen ausgeschlossen werden kann. Alle Mitglieder, die ein Team oder einen Klassenraum in GitLab verlassen haben, werden auch in der Anwendung entfernt. Mitglieder können jedoch nur über die Anwendung hinzugefügt werden; alle Mitglieder, die direkt über GitLab hinzugefügt wurden, werden vom Worker wieder entfernt.

3.4.7 Tests

Bei der Entwicklung der Anwendung wurde darauf geachtet, zentrale Funktionalitäten durch Tests abzusichern. Viele der Controller sind durch zugehörige Integrationstests abgedeckt. Der GitLab-Client wird dabei mit `Mockery` gemockt, sodass er die für den jeweiligen Test die erforderlichen Antworten liefert. Während der Integrationstests wird ein Test-Container mit einer Datenbank gestartet, von der zu Beginn ein Snapshot erstellt wird. Dieser Snapshot kann nach jedem Test wiederhergestellt werden, um eine saubere Datenbank für jeden Testlauf sicherzustellen. Dies ermöglicht eine realistische und zuverlässige Prüfung der Anwendung, ohne Seiteneffekte durch bereits vorhandene Daten.

3.5 Frontend

Im Folgenden wird auf die Architektur des Frontends eingegangen. Zunächst wird erläutert, welche Frontend-Werkzeuge genutzt wurden. Des Weiteren werden Screenshots aus dem Frontend der entwickelten Anwendung vorgestellt. Die Auswahl der Screenshots konzentriert sich auf jene, die die zentralen Aspekte und Funktionen der Anwendung passend repräsentieren und veranschaulichen.

3.5.1 Routing

Für die Navigation innerhalb der `React`-basierten Anwendung wird der `TanStack` Router verwendet, der eine dateibasierte Routing-Architektur implementiert. Dabei werden aus den im Verzeichnis `frontend/routes` definierten `tsx`-Dateien automatisch Routing-Definitionen generiert. Diese Routen werden in der Datei `routeTree.gen.ts` zusammengefasst und repräsentieren den gesamten Routing-Baum der Anwendung. Der `TanStack` Router setzt zudem stark auf die Nutzung von `TypeScript`, wodurch alle Routen typsicher definiert werden müssen. Bei Änderungen an den Routen werden in allen betroffenen Bereichen der Anwendung statische Typprüfungen durchgeführt, wodurch potenzielle Fehler frühzeitig erkannt werden.

3.5.2 Backend-Kommunikation

swagger-client Wie in Unterkapitel Backend (siehe Seite 29) erläutert, wird im Backend eine OpenAPI-2.0-Spezifikation generiert. Mithilfe des Werkzeuges `swagger-codegen` wird daraus ein JavaScript-Client erstellt, der die notwendigen Routen, Typen und Parameter bereitstellt, um eine standardisierte und korrekte Kommunikation mit den API-Endpunkten sicherzustellen.

Tanstack Query `TanStack Query` ist eine leistungsfähige Bibliothek für das Management von asynchronem State-Management in TypeScript- und JavaScript-Anwendungen. Die Bibliothek bietet unter anderem Mechanismen für Caching, Cache-Invalidation, Lade- und Fehlerzustände sowie Prefetching. In Kombination mit dem Swagger-Client wird `TanStack Query` verwendet, um mit der Backend-API zu kommunizieren und dessen Daten in die verschiedenen Komponenten der Anwendung zu integrieren und deren Status zu verwalten.

3.5.3 UI-Frameworks

Zur Optimierung der Frontend-Entwicklung und zur Sicherstellung einer robusten Benutzeroberfläche wurden verschiedene UI-Frameworks verwendet, die im Folgenden beschrieben werden.

Tailwind CSS `Tailwind CSS` ist ein Utility-First CSS-Framework, das eine Vielzahl von vordefinierten Klassen für häufig verwendete Stilelemente bereitstellt. Im Vergleich zur herkömmlichen Erstellung von CSS-Dateien ermöglicht

es eine schnellere und effizientere Implementierung benutzerdefinierter Designs und fördert die Konsistenz innerhalb der Anwendung.

Radix UI `Radix UI` stellt eine Reihe von grundlegend wichtigen UI-Komponenten bereit, die individuell angepasst und gestylt werden können. Diese Komponenten bieten eine solide Basis für die Entwicklung wiederverwendbarer und anpassungsfähiger Benutzeroberflächen.

shadcn/ui `Shadcn/ui` ist eine auf `Radix UI` und `Tailwind CSS` basierende Komponentenbibliothek für `React`. Die Integration der Komponenten erfolgt über ein CLI-Tool, das die entsprechenden Dateien in das Verzeichnis `components/ui` des Projekts speichert. Im Gegensatz zu den reinen `Radix UI`-Komponenten verfügen die `shadcn/ui`-Komponenten über vordefinierte Stile, was eine schnellere Implementierung einer funktionalen Benutzeroberfläche ermöglicht.

3.5.4 Weitere verwendete Werkzeuge

Lucide `Lucide` ist eine Open-Source-Icon-Bibliothek, die eine Vielzahl von SVG-Vektorgrafiken zur Verfügung stellt. Die einfache Integration in Projekte sowie die Möglichkeit, die Symbole an spezifische Designanforderungen anzupassen, machen `Lucide` zu einer flexiblen Lösung für die Gestaltung von Icons in der Benutzeroberfläche.

React Hook Form + Zod `React Hook Form` erleichtert den Umgang mit Formularen in React-Anwendungen, indem es eine einfache und effiziente Verwaltung von Formularzuständen bietet. In Kombination mit der Validierungsbibliothek für Schemas `Zod` wird sichergestellt, dass Formulareingaben kontinuierlich gegen ein definiertes Schema validiert werden, bevor die Daten an den Server gesendet werden. Dies gewährleistet eine typsichere Überprüfung der Benutzereingaben und erhöht die Konsistenz und Integrität der übermittelten Daten.

3.5.5 Screens

Mit Ausnahme der Login-Seite sind sämtliche Seiten der Anwendung durch Zugriffskontrollen gesichert, welche eine erfolgreiche Authentifizierung erfordern. Die Benutzeroberfläche ist durchgehend in englischer Sprache gehalten, um eine international einheitliche Nutzbarkeit zu gewährleisten. Zur Förderung einer benutzerfreundlichen Anpassung der visuellen Darstellung wurde ein `Theme Provider` implementiert. Dieser ermöglicht es den Nutzenden, entweder das systemweit definierte Theme (Light- oder Dark-Mode) zu übernehmen oder eine individuelle Anpassung des Themes vorzunehmen, wodurch eine höhere Flexibilität hinsichtlich der Benutzerpräferenzen gewährleistet wird.

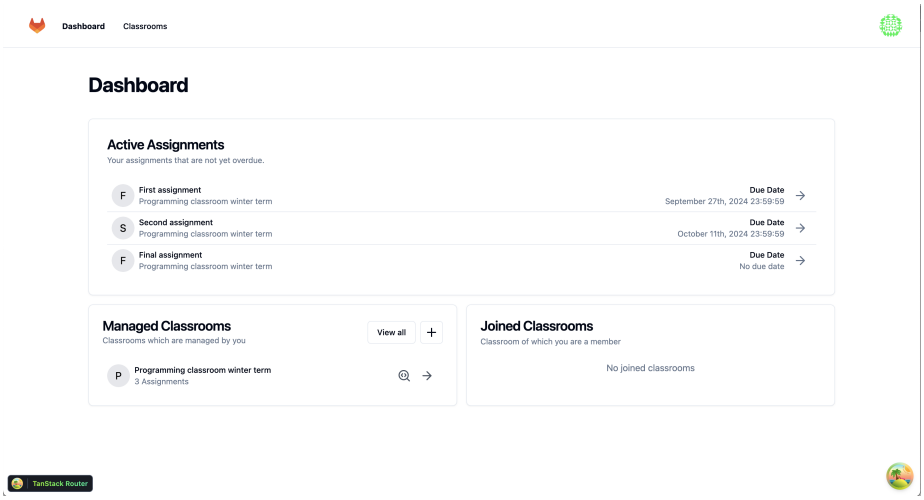

Abbildung 4: Darstellung des Dashboards aus Sicht eines Dozierenden

Dashboard Ein zentrales Element der Anwendung ist das Dashboard, welches in Abbildung 4 zu sehen ist. Auf diesem Dashboard werden alle aktuellen Aufgaben (`Assignments`) mitsamt relevanten Details, wie beispielsweise Fälligkeitsdaten, angezeigt. Zudem bietet das Dashboard eine Übersicht über die selbstverwalteten Klassenräume sowie über jene Klassenräume, denen der Nutzende beigetreten ist.

Erstellen eines Klassenraums Vom Dashboard aus besteht die Möglichkeit, neue Klassenräume zu erstellen. Bei der Erstellung müssen grundlegende Informationen wie der Name und eine Beschreibung des Klassenraums angegeben werden.

Zusätzlich können Dozierende festlegen, ob die Studierenden in Teams zusammenarbeiten sollen, und falls ja, die maximale Teamgröße bestimmen. Hierbei steht zur Auswahl, ob die Studierenden die Teams selbst erstellen können oder ob diese vom Dozierenden vorgegeben werden sollen.

Eine weitere Einstellmöglichkeit betrifft die Sichtbarkeit der Projekt-Repositories anderer Studierender. Dozierende können festlegen, ob es möglich sein soll, dass Studierende die Projekt-Repositories von anderen Studierenden betrachten können oder ob der Zugang auf die eigenen Projekte beschränkt bleiben soll.

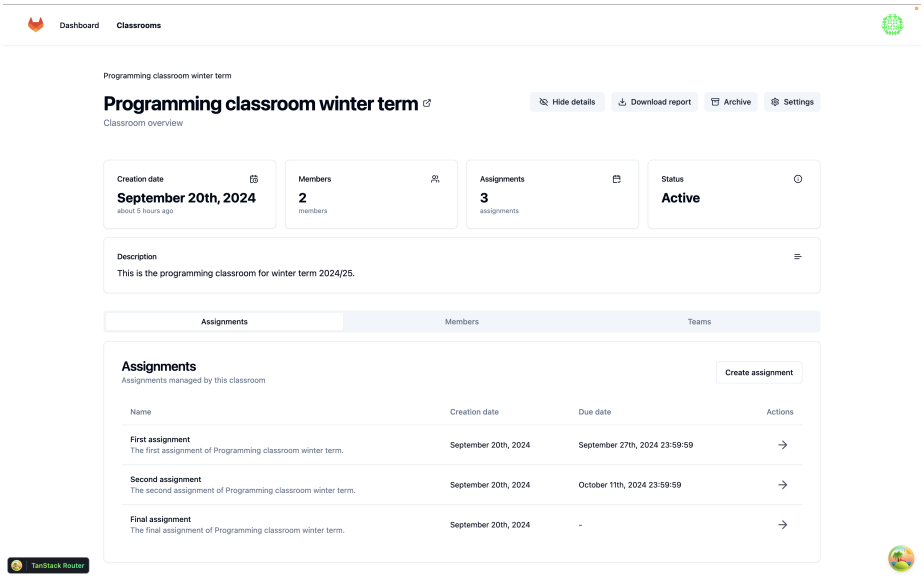

Abbildung 5: Darstellung eines Klassenraums aus Sicht eines Dozierenden

Klassenraumübersicht Die Übersicht eines Klassenraums, wie einer in Abbildung 5 zu sehen ist, stellt alle Informationen im Zusammenhang eines Klassenraums dar. Insbesondere zur Verbesserung der Übersicht auf kleinen (mobilen) Endgeräten gibt es die Möglichkeit, auf dieser Seite einige Informationen auszublenden. Sofern die Seite von dem Inhaber oder einem Moderator des Klassenraums aufgerufen wird, stehen zusätzliche Aktionen für die Verwaltung des Klassenraums zur Verfügung, wie in Unterunterabschnitt 3.2.2 näher beschrieben.

1. **Projektreport herunterladen** Zum einen ist es hier möglich, einen Projektreport, der, wie in Abschnitt 3.4.5 beschrieben, die Ergebnisse der manuelle und automatischen Bewertung enthält, als CSV-Datei herunterzuladen.

2. **Klassenraum archivieren** Außerdem ist es möglich, den Klassenraum zu archivieren. Hiermit wird der Klassenraum intern als archiviert markiert. Wie in Kapitel Abschnitt 1.4.1 beschrieben, verlieren alle Studierende Schreibrechte auf das Repository. Eine weitere Bearbeitung ist nicht mehr möglich.

3. **Einstellungen des Klassenraums öffnen** Darüber hinaus besteht die Möglichkeit, in die Einstellungen des Klassenraums zu gelangen, welche im nächsten Abschnitt näher erläutert werden.

4. **Detailansicht eines Assignments öffnen** Im Reiter *Assignments* können Dozierende ein bestehendes `Assignment` auswählen, um in die dazugehörige Detailansicht zu gelangen, welche in einem späteren Abschnitt erläutert wird. Zusätzlich besteht die Möglichkeit, von diesem Reiter aus ein neues `Assignment` zu erstellen.

5. **Verwaltung von Mitgliedern eines Klassenraums** Im Reiter *Members* über die Funktion »Manage members« hat ein Dozierender hat die Möglichkeit, die Teamzugehörigkeit aller Mitglieder nachträglich anzupassen. Außerdem können die Berechtigungsstufen (Student, Moderator oder Owner) der Mitglieder, wie in Unterunterabschnitt 3.2.2 beschrieben, individuell festgelegt werden. Die verfügbaren Rollen sind Student, Moderator und Owner.

6. **Einladen von Mitgliedern eines Klassenraums** Im Reiter *Members* können Studierende über die Funktion »Invite members« per E-Mail zu einem Klassenraum eingeladen werden. Nach dem Erhalt der Einladung haben die Studierenden die Möglichkeit, diese entweder anzunehmen oder abzulehnen.

7. **Teams einsehen, bearbeiten und erstellen** Im Reiter *Teams* werden alle erstellten Teams sowie ihre Anzahl der Mitglieder aufgelistet. Außerdem können weitere Teams erstellt und bestehende Teams umbenannt werden.

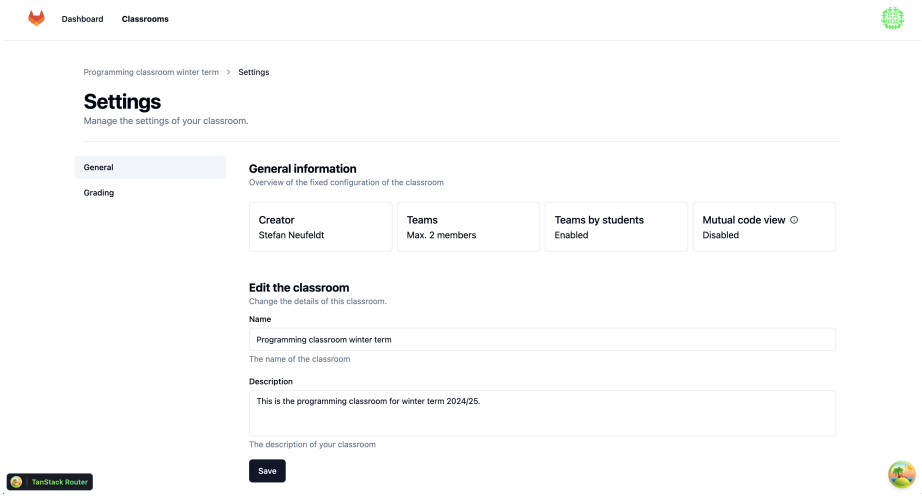

Abbildung 6: Allgemeine Einstellungen eines Klassenraums

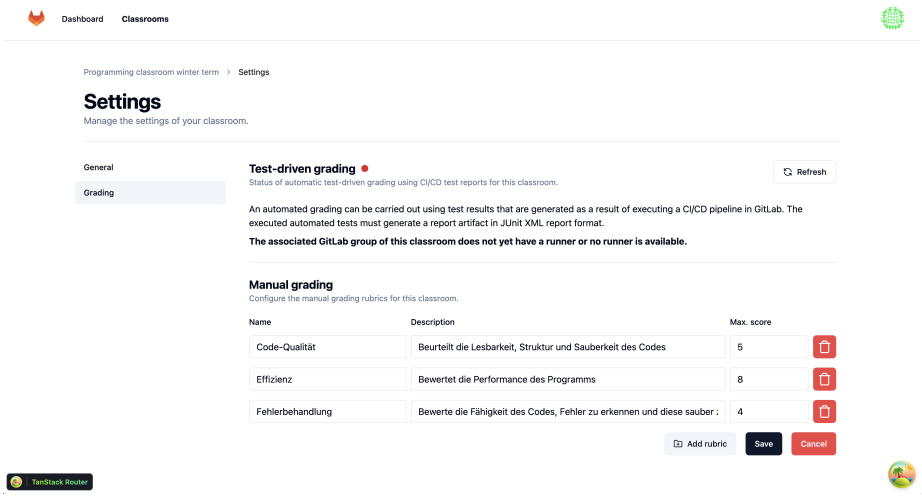

Abbildung 7: Einstellungen zur Benotung eines Klassenraums

Klassenraumeinstellungen In den Einstellungen eines Klassenraums (Abbildung 6) kann der Name sowie die Beschreibung eines Klassenraums auch nachträglich geändert werden. Auf einer weiteren Registerkarte (Abbildung 7) können Einstellung zur Benotung, welches in Abschnitt 3.4.5 beschrieben ist, festgelegt werden. Hier kann die automatisierte Bewertung aktiviert werden, sofern ein entsprechend GitLab-Runner eingerichtet ist. Für die manuelle Benotung können hier Rubriken definiert werden, die Name, Beschreibung und maximal-erreichbare Punktzahl enthalten.

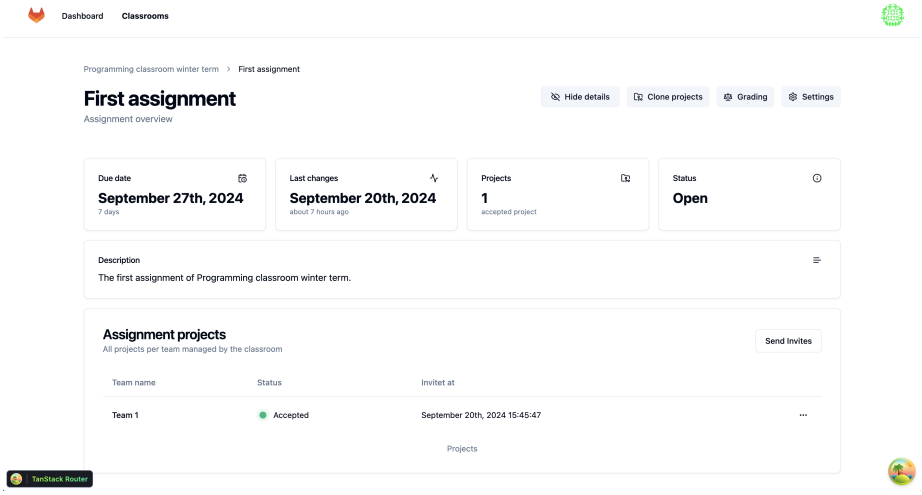

Abbildung 8: Übersicht eines Assignments

Aufgabenübersicht Die Aufgabenübersicht (»Assignment overview«) (siehe Abbildung 8) ist, wie bereits erwähnt, nur für Dozierende oder Moderatoren zugänglich. Auf dieser Seite werden alle Information zu einem Assignment dargestellt. Der Aufbau dieser Seite ist an die Klassenraumübersicht angelehnt. Auch hier können für eine kompaktere Darstellung einige Informationen verborgen werden, was die Übersicht auf kleineren Endgeräten erleichtert. Folgenden Aktionen können von hier aus ausgeführt werden:

1. **Klonen des Projekts** Damit ein Dozierender die Möglichkeit hat, auf eine einfache Weise alle Projekte der Studierenden innerhalb eines Assignments lokal auf den eigenen Rechner zu klonen, gibt es die Funktion »Clone projects«, welche ein Shell-Skript zur Verfügung stellt und in die Zwischenablage kopiert. Dieses Skript kann anschließend vom Dozierenden in einem Terminal ausgeführt werden, wodurch alle Repositories der Studierenden lokal auf dem Rechner geklont werden. Diese Automatisierung unterstützt den Dozierenden, indem der Zugriff auf die Projektdateien erleichtert wird und so die Bewertung vorgenommen werden kann.

2. **Benotung vornehmen** Im Bereich *Grading* kann die Benotung der einzelnen Abgaben für die Aufgabe (`Assignment`) vorgenommen werden. In einem weiteren Abschnitt wird näher auf diese Funktion eingegangen.

3. **Einstellungen der Aufgabe anpassen** Unter *Settings* sind die Einstellungen des `Assignments` bzw. der Aufgabe erreichbar. Diese Funktion wird in dem nächsten Abschnitt genauer erläutert.

4. **Einladungen für die Aufgabe versenden** Über den Knopf *Send Invites* haben Dozierende die Möglichkeit, Einladungen an alle Studierende des Klassenraums zu versenden, um diese zu dem ausgewählten `Assignment` hinzuzufügen.

5. **Verlinkung zu GitLab-Projekten der Teams** Weitere Möglichkeiten sind in der Liste der Teams zu finden. Hier kann zum einen ein Link zur jeweiligen Gruppe in GitLab aufgerufen werden.

6. **Abrufen von teamspezifischen Berichten** Außerdem besteht die Möglichkeit, einen Report der Bewertungen des jeweils ausgewählten Teams herunterzuladen. Dieser Report enthält detaillierte Informationen zur manuellen und automatisierten Bewertung, einschließlich der erreichten Punktzahlen und des Feedbacks.

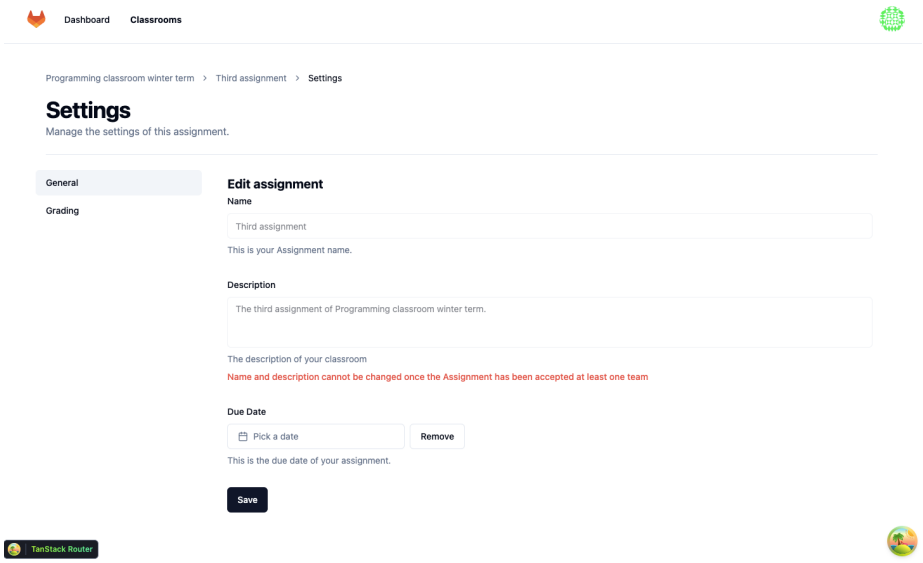

Abbildung 9: Allgemeine Einstellungen einer Aufgabe

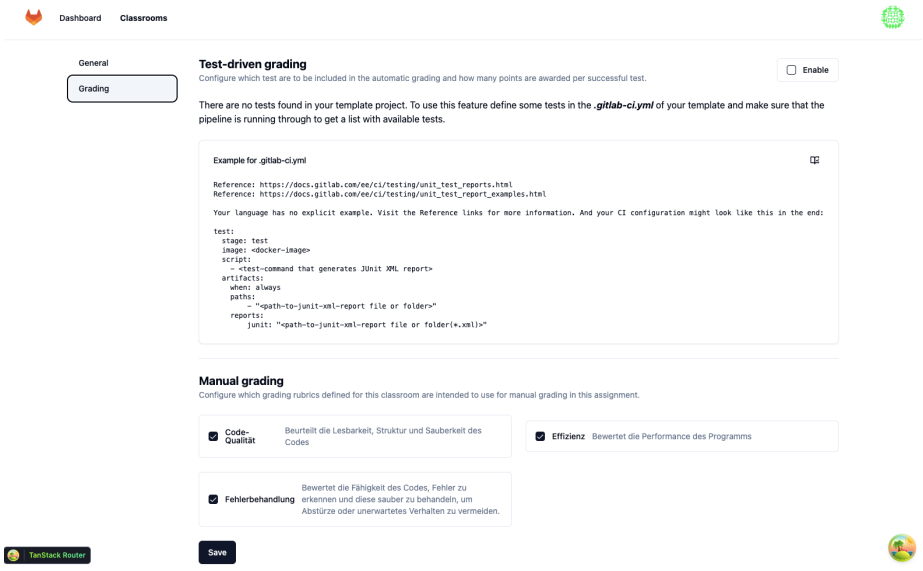

Abbildung 10: Einstellungen zur Benotung einer Aufgabe

Aufgabeneinstellungen In den Aufgabeneinstellungen (»Assignment settings«) (siehe Abbildung 9)lassen sich der Name und die Beschreibung anpassen, sofern die Aufgabe noch nicht von mindestens einer Gruppe akzeptiert worden ist. Außerdem kann hier das Abgabedatum nachträglich angepasst werden, wobei bereits geschlossene Aufgaben dabei erneut freigegeben werden. In den Bewertungseinstellungen (siehe Abbildung 10) können Dozierende festlegen, welche automatisierten Tests in die Bewertung einfließen sollen. Für jeden dieser Tests kann eine Punktzahl zugewiesen werden. Darüber hinaus können die zuvor definierten Rubriken für manuelle Bewertungen wahlweise einbezogen oder ausgeschlossen werden.

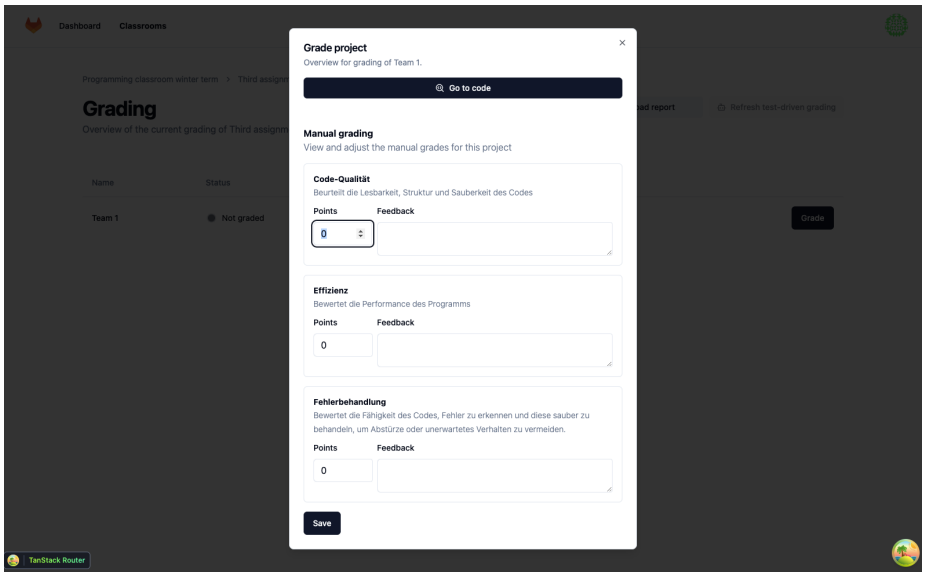

Abbildung 11: Benotung einer Aufgabe vornehmen

Bewertung vornehmen Mit der Funktion »Refresh test-driven grading« können die Ergebnisse der automatisierten Bewertungen abgerufen und dargestellt werden. Die manuelle Benotung wird vom Dozierenden für jede Gruppe separat vorgenommen. Dabei kann der Dozierende für jede zuvor definierte Rubrik individuell eine Punkteanzahl vergeben sowie ein Feedback zur jeweiligen Rubrik hinterlassen werden.

3.5.6 `Swagger UI`

Parallel zur `React`-Anwendung wird die `Swagger UI` bereitgestellt, welche eine interaktive und automatisch generierte Dokumentation der API-Endpunkte des Backends bietet. Über die Benutzeroberfläche, die im Browser unter dem Pfad `/docs.html` zugänglich ist, können alle verfügbaren Endpunkte der API eingesehen und deren Funktionalitäten detailliert geprüft werden. Darüber hinaus ermöglicht die `Swagger UI` das direkte Testen der Endpunkte, indem Anfragen mit verschiedenen Parametern und Daten ausgeführt werden können, welches die Validierung der API-Kommunikation vereinfacht und bei der Fehlersuche unterstützt.

4 Deployment

Die Bereitstellung der Anwendung erfolgt mit Docker als Container-Technologien. Detailliertere Informationen zur Bereitstellung befinden sich im Repository der Anwendung.

CI/CD und Staging Umgebung Eine CI/CD-Pipeline wird verwendet, um die Anwendung, bevor diese ausgeliefert wird in einer Staging-Umgebung bereitgestellt. Tests und Linting werden automatisch bei jedem Commit ausgeführt, um sicherzustellen, dass die Anwendung korrekt funktioniert. Nach erfolgreichem Testen wird ein Docker-Image erzeugt und in der Docker-Image-Registry veröffentlicht. Das Image kann von Entwicklern und Administratoren für die lokale oder serverseitige Ausführung der Anwendung heruntergeladen und verwendet werden.

Ausführung mit `Docker Compose` Für das Hosting der Anwendung wird `Docker Compose` verwendet. Dies erleichtert die Verwaltung von Abhängigkeiten und stellt sicher, dass alle notwendigen Komponenten in isolierten Containern ausgeführt werden können.

4.1 Schritte für die Bereitstellung

1. **Klonen des Repositories**
   oder direkte Nutzung der `docker-compose.yaml` und `.env.example`
2. **Umgebungsvariablen konfigurieren**
   `cp .env.example .env`
3. **Erstellen einer Anwendung in GitLab**
   (a) Erstelle eine neue Anwendung in GitLab mit der folgenden Redirect-URI:
   `<PUBLIC_URL>/api/v1/auth/gitlab/callback`
   (b) Setzen der Anwendung auf vertraulich `(Confidential: true)` und gib den Scope »api« an. Bei beiden Einstellungen handelt es sich um die Standardeinstellungen.
   (c) Kopiere der `Application ID` und des `Secret` in die `.env`-Datei und füge die URLs `PUBLIC_URL` und `GITLAB_URL` hinzu.
4. **SMTP-Konfiguration** Füge SMTP-Zugangsdaten hinzu, um Einladungs-E-Mails versenden zu können.
5. **Datenbank in `.env`-Datei konfigurieren**

6. **Anwendung starten** Die Anwendung und eine PostgreSQL-Datenbank mit `Docker Compose` zu starten:
   `docker compose up -env-file .env -d`

Alternativ kann eine Bereitstellung auch ohne PostgreSQL-Datenbank erfolgen, indem das nur das Anwendungs-Image ausgeführt wird (siehe `docker-compose.yml`). Zusätzlich kann eine bereits gebaute Binary verwendet werden.

4.2 Kurzfassung für die Entwicklungsumgebung

Diese Schritte können befolgt werden, um die Umgebung für die Entwicklung zu starten. Eine detailliertere Beschreibung befindet sich im Projekt-Repository unter `git-classrooms/docs`.

Voraussetzungen

- Node.js >= 20
- Yarn 1.22.22
- Go >= 1.22

Repository klonen

```
git clone git@github.com/GitClassrooms/git-classrooms
cd git-classrooms
```

Umgebungsdatei konfigurieren `cp .env.example .env`

GitLab-Anwendung erstellen

- Redirect URIs:
  - `http://localhost:3000/api/v1/auth/gitlab/callback`
  - `http://localhost:5173/api/v1/auth/gitlab/callback`
- Confidential: true, Scope: api
- `Application ID` als `AUTH_CLIENT_ID` und `AUTH_CLIENT_SECRET` in die `.env`-Datei kopieren
- `GITLAB_URL` und `PUBLIC_URL=http://localhost:5173` hinzufügen

SSL-Zertifikat für den Mail-Server generieren `openssl req -x509 -newkey rsa:4096 -nodes -keyout .docker/mail/privkey.pem -out .docker/mail/cert.pem -sha256 -days 3650`

Anwendung starten

```
./script/dev.sh
```

Auf die Anwendung zugreifen

Um die Anwendung im Entwicklungsmodus lokal zu erreichen und zu testen, stehen verschiedene Endpunkte zur Verfügung:

- `http://localhost:5173` – Vite Development Server mit Hotreloading
- `http://localhost:3000` – Go Server (stellt gebuildetes Frontend zur Verfügung)
- `http://localhost:8025` – Weboberfläche des SMTP-Mail-Testservers

Hinweis: Ausführlichere Informationen befinden sich im Repository der Anwendung.

4.3 Beispielhafte Konfigurationsdatei

Dies ist ein Beispiel einer vollständigen Konfiguration für die Anwendung, welche als `.env`-Datei bereitgestellt wird.

```
# Application configuration
PUBLIC_URL=
PORT=3000
TRUSTED_PROXIES= #optional

# Database configuration
POSTGRES_HOST=postgres
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres
POSTGRES_DB=postgres

# Auth configuration
AUTH_CLIENT_ID=      % GitLab Application ID
AUTH_CLIENT_SECRET= % GitLab Application Secret
AUTH_SCOPES=api      % GitLab Application Scope

# Gitlab configuration
GITLAB_URL=https://hs-flensburg.dev
GITLAB_SYNC_INTERVAL=5m % Worker to sync with GitLab

# Email configuration
SMTP_HOST=mail
SMTP_PORT=1025
SMTP_PASSWORD=password
SMTP_USER=classroom@hs-flensburg.dev
```

5 Projektbericht

Im folgenden Abschnitt wird der Projektverlauf des Projektes grob erläutert. Dabei werden sowohl das gewählte Entwicklungsmodell als auch die Erreichung zentraler Meilensteine beschrieben, um einen Einblick in die zeitlichen und organisatorischen Rahmenbedingungen des Projekts zu geben.

5.1 Wahl des Entwicklungsmodells

Für das Projektmanagement wurde ein SCRUM-ähnlicher Ansatz verfolgt, bei dem es wöchentliche Rücksprachen gab und die Entwicklung in mehrere Sprints (Entwicklungszyklen) unterteilt wurde, innerhalb derer sich bestimmten Aspekten des Projekts gewidmet wird. Das Endprodukt wurde dafür in viele kleine Arbeitspakete aufgeteilt, zu denen es im GitLab Projekt jeweils einzelne »Issues« gibt, welche einzelnen Teammitglieder zugewiesen waren. Grundsätzlich gab es bei der Bearbeitung der Issues keine klare Trennung zwischen Backend- und Frontend-Entwicklern. Allerdings zeigen sich in der Praxis schon Präferenzen der einzelnen Entwickler, welche bei der Arbeitsverteilung berücksichtigt wurden.

5.2 Verwendung des Git Flow Modells

Im Rahmen des Projekts wurden Änderungen nach dem Git Flow Modell eingebracht, bei dem klar strukturierte Branches für die Entwicklung genutzt werden. Dies ermöglicht eine saubere Trennung zwischen neuen Features, Bugfixes und Releases. Zusätzlich sorgte dieses Modell dafür, dass alle Änderungen vor ihrer Integration durch automatische Tests auf mögliche Wechselwirkungen mit bestehendem Code überprüft wurden.

5.3 Meilensteine

Für einen vollständigen Entwicklungsüberblick ist es ratsam, die Commit-Historie des Projekt-Repositories zu konsultieren. Es folgt ein grundlegender Überblick über die einzelnen Meilensteine bzw. Sprints:

- **Kickoff**
  25. Oktober 2023 bis 8. November 2023
  Beim Kickoff-Meeting wurde gemeinsam mit dem Dozenten das Ziel des Projekts ausführlich erörtert. Dabei wurden die zu verwendenden Technologien festgelegt und organisatorische Aspekte wie Termine und die Rollenverteilung innerhalb der Gruppe besprochen. Zudem wurde vereinbart, dass sich jedes Teammitglied vor Projektbeginn mit den ausgewählten Technologien vertraut macht, um eine solide Grundlage für den Projektstart zu schaffen. Während der initialen Phase wurde auch erster UI/UX-Prototyp mit Figma erschaffen.

- **Sprint 1-3: Die Grundlage**
  8. November 2023 bis 4. Februar 2024
  Im Sprint 1 wurde überprüft, wie die GitLab API testweise angebunden werden kann, und das Datenmodell sowie die Datenbank-Anbindung (via `GORM`) implementiert. Das Deployment wurde mithilfe von einer CI/CD-Pipeline auf Docker-Basis aufgebaut, und die Authentifizierung über GitLab als OAuth2-Provider sowie der OAuth2-Flow im Backend integriert. Insgesamt wurde eine erste Basis für die Anwendung geschaffen. Als Backend-Web-Framework wird `GoFiber` eingesetzt.

  Weiterhin wurde die GitLab-Anbindung durch eine Repository-Architektur eingeführt, welche das Ziel hat, den API-Client bei Bedarf austauschen zu können. Als API-Client für GitLab wurde das Go-Paket `xanzy/go-gitlab` verwendet und testweise angebunden. Auf Basis dessen wurden grundlegende Funktionen wie das Erstellen eines Klassenraums und einer *Aufgabe* eingeführt. Des Weiteren wurde das Session-Management initial implementiert.

  Innerhalb von Sprint 3 wurde die initiale Version der eigenen Backend-API erstellt, welche zahlreiche Funktionen rund um das Klassenraum-Management implementiert. So können Studierende über E-Mail-Adressen zu einem Klassenraum eingeladen werden. Diese erhalten einen Einladungslink, über welchen sie in Kombination mit ihrem GitLab-Account zum Klassenraum beitreten können, zu einem eingeladen werden und Einladungslinks an Mitglieder versendet werden. Zusätzlich wird eine GitLab-Gruppe (welche den Klassenraum in GitLab repräsentiert) erstellt und der Klassenraum im Frontend sowie Backend verwaltbar. Außerdem wurde die Möglichkeit, über den Einladungslink einem Klassenraum beizutreten, im Frontend implementiert, sowie die Benutzerverwaltung nach OAuth-Authentifizierung abschließend integriert. Des Weiteren wurde die E-Mail-Einrichtung, die Frontend-Projektstruktur und die ersten Anpassungen in Tailwind abgeschlossen sowie erste Frontend-Komponenten mit der React-Komponentenbibliothek `shadcn/ui` erstellt.

  Damit war eine Grundlage für die weiteren Implementierungen vorhanden, da die technologische Grundbasis gelegt war und so die Weiterentwicklung folgen konnte.

- **Sprint 4: Team-basierte Klassenräume und UI-Überarbeitung**
  5. April 2024 bis 18. April 2024
  Im Sprint 4 wurde die erste Version der Backend-API vollständig überarbeitet und unter anderem die Bearbeitung von Aufgaben in Teams implementiert. Dies stellte nochmal eine große Veränderung des gesamten Backends dar. Klassenräume werden in GitLab durch eine Gruppe repräsentiert, welche in Namen des Dozierenden angelegt wird. Teams bzw. Studierende verfügen jeweils über eine Untergruppe, welche die eigentlichen Projekte pro gestellter Aufgabe beinhalten. Das Frontend wurde um die Anzeige von Avataren sowie die Benutzer-Logout-Funktionen erwei-

tert. Zudem wurden die Navigation und die Avatar-URL in den Benutzerabfragen integriert. Auch wurde für die CI/CD-Pipeline ein Test-Setup eingerichtet und API-Endpunkte für das Bearbeiten von Klassenräumen hinzugefügt.

- **Sprint 5: Feedback und Berechtigung**
  19. April 2024 bis 3. May 2024
  Im Sprint 5 wurden Webhooks für GitLab-Gruppen untersucht, um auf Ereignisse innerhalb von GitLab, wie das Verlassen einer Gruppe, reagieren zu können. Dabei stellte sich heraus, dass GitLab grundsätzlich Webhooks unterstützt, die auch über die API erstellt werden können. Allerdings bietet die Community-Edition von GitLab keine Webhooks auf Gruppenebene an. Aus diesem Grund wurde entschieden, die Reaktion auf solche Änderungen später in einem eigens entwickelten Synchronisierungssystem zu implementieren, um eine einheitliche Funktionalität in allen GitLab-Editionen zu gewährleisten. Im weiteren Verlauf wurde erste Features für die Bewertung implementiert, dies umfasste unter anderem Feedback-Branches mit Merge Requests für Dozierende. Es wurden außerdem bereits mögliche Konzepte für automatische Bewertungen beziehungsweise Tests geprüft. Im Frontend wurden die Komponenten für die Aufgabenliste, Mitgliederliste und Teamliste aktualisiert. Ein weiterer wichtiger Meilenstein war die Einführung des Go-Paketes `swaggo/swag` und die damit verbundene Ergänzung von Swagger-Kommentaren zu allen Endpunkten. Dies sorgt unter anderem für eine generative Dokumentation der Backend-API und ermöglicht auch die Code-Generierung von API-Clients für diese auf Basis der entstandenen OpenAPI-Spezifikation. Schließlich wurde auch begonnen, die Berechtigungen sowohl für die Nutzung der Backend-API als auch die Berechtigungen der Studierende in den GitLab-Gruppen beziehungsweise Klassenräumen zu überarbeiten.

- **Sprint 6: Frontend und Studierende**
  3. Mai 2024 bis 16. Mai 2024
  Im Sprint 6 wurden die Berechtigungen für Klassenraum-Mitglieder weiter implementiert. Die Struktur sieht nun vor, dass der Erschaffende des Klassenraums jederzeit `Owner`-Rechte für die gesamte Klassenraumstruktur hat. Studierende werden als Gäste im Klassenraum geführt und erhalten in ihrer Untergruppe Reporter-Rechte. Während eine Aufgabe noch bearbeitet werden darf (daher bis zur Fälligkeit) haben Studierende `Maintainer`-Rechte in ihrem jeweiligen Projekt-Repository. Moderatoren haben allgemein `Reporter`-Rechte im gesamten Klassenraum. Im Frontend wurden die Seiten »Klassenraum beitreten« sowie »Aufgabe annehmen« aktualisiert. Zudem wurde das Avatar-Dropdown-Menü überarbeitet und ein Pop-up für die Zuweisung von Klassenraum-Mitgliedern erstellt. Das Backend wurde ebenfalls für die Funktionen »Klassenraum beitreten« und »Aufgabe annehmen« überarbeitet, um den aktuellen Status über Einladungen dazu einzusehen.

- **Sprint 7: Mehr Frontend und Studierende**
  17. May 2024 bis 30. May 2024
  Im Sprint 7 wurde das neue Layout im Frontend angepasst und die Möglichkeit eingeführt, Aufgaben zu bearbeiten, solange sie noch nicht akzeptiert sind. Zudem wurde die erlaubten Zeichen für die Namensgebung von Aufgaben im Frontend eingeschränkt, um GitLab-konform zu sein und einige weitere Fehler wurden behoben.

- **Sprint 8: Grading und UI**
  31. May 2024 bis 14. Juni 2024
  Im Sprint 8 wurde das Frontend weiter an die Überarbeitungen der Backend-API (aus Sprint 4) angepasst. Auch wurde das »Klassenraum erstellen«-Formular überarbeitet. Für die automatische Bewertung von Tests wurden, für das Backend, im GitLab-API-Repository Funktionen für die Auswertung von Testergebnisse von CI-/CD-Pipelines implementiert. Dies stellt den ersten Schritt dar, um automatische, testgetriebene Bewertungen durchzuführen. Die Frontend-Routen wurden an die API-Version 2 angepasst, und die Archivierungsfunktion für Klassenräume hinzugefügt.

- **Sprint 9: Grading und UI II**
  14. Juni 2024 bis 28. Juni 2024
  Im Sprint 9 wurde die Login-Seite überarbeitet und Möglichkeiten für automatisches Grading sowie automatisierte Test und Endpoints weiter implementiert. Ein Endpunkt zum Zurückziehen gesendeter Einladungen wurde ebenfalls hinzugefügt. Auch wurde eine minimale Corporate Identity erstellt. Das Frontend wurde aktualisiert, um responsiver zu werden. Dies beinhaltete beispielsweise die mobile Ansicht des Menüs bzw. des Benutzer-Menüs. Wie in Sprint 5 erwähnt, wurde begonnen, ein auf Go-Routinen-basiertes Worker-System zu entwickeln, welches Hintergrundaufgaben abwickelt. Das umfasst im konkreten Fall Synchronisierungsaufga-

ben, um die Konsistenz mit GitLab erhalten, da Mitglieder, Bezeichnungen und Weiteres synchronisiert werden müssen.

- **Sprint 10: Grading und UI III**
  27. Juni 2024 bis 19. Juli 2024
  Im Sprint 10 wurde im Frontend die Klassenraum-Liste aktualisiert, um Änderungen im Backend zu entsprechen, welche unter anderem neue Endpunkte betreffen, um Git-Repository-Links für Projekte zu einer Aufgabe abzurufen. Diese werden im Frontend verwendet, um schnell zu einem Studierenden-Repository zu gelangen. Zudem wurden fehlenden Seiten identifiziert und entsprechende Issues erstellt. Auch die Implementierung der Synchronisation mit dem Worker-System wurde erstmals abgeschlossen.

- **Sprint 11: Frontend und Frontend**
  9. Juli 2024 bis 23. August 2024
  Im Sprint 11 wurde der Figma-Prototyp weiterentwickelt und die Backend-Strukturen für die Bewertung zu Ende implementiert. Im Frontend wurden Team-Slots im Teammanagement hinzugefügt. Ein Worker wurde ergänzt, welcher die Berechtigung für alle fälligen Aufgaben so ändert, dass Studierende nicht mehr weitere Änderungen am Quellcode in ihren Projekt-Repositories durchführen können. Für das implementierte Bewertungssystem wurde ein Konzept für die UI der zur Konfiguration und Bewertung im Frontend geschaffen. Zudem wurden das Exportieren der Bewertungsergebnisse in einer CSV-Datei implementiert. Im Frontend wurden die visuellen Elemente beim Beitreten eines Klassenraums und das Akzeptieren von Aufgaben aktualisiert. Auch wurden verschiedene Seiten wie »Klassenräume bearbeiten« und die »Klassenraum-Übersicht« hinzugefügt. Weiterhin wurde die Synchronisierung von Klassenraum- und Teamdaten überprüft und weitere Fehler behoben. Auch das Mail-Senden wurde optimiert. Schließlich wurde noch ein neuer Endpunkt für aktuell aktive Aufgaben ergänzt.

- **Sprint 12: Frontend und Dokumentation**
  24. August 2024 bis 13. September 2024
  Im Sprint 12 wurden restliche fehlende Endpunkte im Backend für Mitglieder des Klassenraums ergänzt und die Ansicht der Aufgaben-Details im Frontend aktualisiert. Es wurden neue Seiten für Klassenraum-Mitglieder sowie Einstellungen für Klassenräume und Aufgaben hinzugefügt. Zudem wurden Endpunkte für Bearbeitung von Aufgaben implementiert. Die Worker zur Synchronisierung eröffnen nun auch geschlossene Aufgaben wieder, wenn das Fälligkeitsdatum in die Zukunft verlegt wurde. Die Integrationstests im Backend wurden vollständig überarbeitet. Im Frontend entstanden auch Komponenten für die aktive Aufgaben-Liste und die Klassenraum-Ansicht für Studierende. Abschließend wurde die Softwaredokumentation strukturiert.

- **Sprint 13: Abgabe**
  6. September 2024 bis 27. September 2024
  Im Sprint 13, dem finalen Sprint zur Abgabe, wurde die Anwendung intensiv auf Fehler getestet, um eine stabile Version zu gewährleisten. Es wurde entschieden, welche Funktionen für die Version 1 noch implementiert werden sollen. Zudem wurde weiter an der Dokumentation gearbeitet und über das zukünftige Vorgehen mit dem Projekt beraten, insbesondere ob eine Veröffentlichung als Open-Source in Betracht gezogen wird.

6 Ergebnis

Die Umsetzung des »GitClassrooms«-Projekts brachte einige technische und teambezogene Herausforderungen mit sich. Ein zentrales Ergebnis war die erfolgreiche Integration der Klassenraumstruktur in GitLab, ohne in den GitLab-Quellcode einzugreifen. Dies wurde durch die Entwicklung einer eigenen Anwendung und der Verwendung von Gruppen und Zugriffsrechten innerhalb von GitLab erreicht. Die Klassenräume wurden als Gruppen und die Aufgaben als Projekte strukturiert, wobei die Studierenden oder Teams jeweils separate Repositories für ihre Aufgaben erhielten und die Zugriffsrechte von der Anwendung verwaltet werden.

Die automatisierte Bewertung der Programmieraufgaben wurde ebenfalls erfolgreich in die Anwendung integriert. GitLab-CI/CD-Pipelines führten Tests durch, deren Ergebnisse zur automatisierten Bewertung herangezogen wurden. Diese Methode ermöglichte eine standardisierte und gut dokumentierte Auswertung von Testläufen direkt in GitLab, deren Ergebnisse automatisch in die Bewertung mit einbezogen werden können.

Zusätzlich wurde ein Worker-basiertes System implementiert, das die GitLab-Instanz in regelmäßigen Abständen auf Änderungen überprüft. Dies war notwendig, um die Anwendung mit allen Editionen von GitLab kompatibel zu machen, da die Open-Source-Version nicht über gruppenbasierte Webhooks verfügte.

Die Anwendung insgesamt wurde so konzipiert, dass GitLab als Versionskontrollsystem theoretisch mit einigen Modifikationen auch durch andere Systeme ersetzt werden könnte, wie beispielsweise Gitea.

Letztlich konnten alle funktionalen Anforderungen des Projekts (siehe Unterabschnitt 1.4 auf Seite 9) erfolgreich umgesetzt werden. Die Anwendung bietet die grundlegende Verwaltung von Klassenräumen, Aufgaben und Teams, sowie die Unterstützung von automatisierten und manuellen Bewertungssystemen. Alle geplanten Features sind im vorgegebenen Zeitrahmen implementiert worden und stehen den Nutzern zur Verfügung.

ID	Bezeichnung	Erledigt (✓)
A01	Auth über OAuth mit GitLab	✓
A02	Übersicht über verwaltete und beigetretene Klassenräume	✓
A03	Übersicht über anstehende Aufgaben	✓
A04	Erstellung von Klassenräumen	✓
A05	Editierung von Klassenräumen	✓
A06	Archivierung von Klassenräumen	✓
A07	Einladen von Personen in einen Klassenraum	✓
A08	Beitreten zu einem Klassenraum	✓
A09	Übersicht über den Klassenraum	✓
A10	Übersicht einer Aufgabe	✓
A11	Erstellung von Aufgaben	✓
A12	Editierung von Aufgaben	✓
A13	Einladen von Studierenden zu einer Aufgabe	✓
A14	Akzeptieren von Aufgaben	✓
A15	Einsehen und Bearbeiten von Projekten	✓
A16	Erstellung von Teams	✓
A17	Editierung von Teams	✓
A18	Beitreten zu einem Team	✓
A19	Zuweisen von Teams	✓
A20	Zuweisen von Rollen	✓

Tabelle 1: Erledigung von funktionalen Anforderungen

ID	Bezeichnung	Erledigt (✓)
K01	Übersicht über Bewertung aller Projekte einer Aufgabe	✓
K02	Erstellen von manuellen Bewertungskriterien	✓
K03	Editieren von manuellen Bewertungskriterien	✓
K04	Manuelle Kriterien für Aufgaben aktivieren	✓
K05	Manuelle Bewertung einer Aufgabe	✓
K06	Automatische Bewertung einer Aufgabe mit Tests	✓
K07	Feedback-Branch zum Kommentieren vom Quellcode	✓

Tabelle 2: Erledigung von optionalen funktionalen Anforderungen

6.1 Herausforderungen

6.1.1 Technische Herausforderungen

Konzeptionierung einer Klassenraumstruktur in GitLab Die Herausforderung bestand darin, die Rechte und Strukturen von Klassenräumen in GitLab umzusetzen, ohne dabei in den GitLab-Code einzugreifen (siehe Seite 23). Mehrere Ideen und Ansätze wurden ausprobiert, um Gruppen und Berechtigungen sinnvoll zu strukturieren. Nach verschiedenen Versuchen konnte schließlich eine geeignete Lösung gefunden werden, die die Verwaltung von Klassenräumen in beliebigen GitLab-Instanzen ermöglicht und für die Einrichtung keinerlei Administrationsrechte auf der Instanz benötigt.

Unerwartetes Verhalten der GitLab-API Ein unvorhersehbares Verhalten der GitLab-API stellte eine weitere Hürde dar, da bestimmte API-Endpunkte nicht wie erwartet funktionierten.

Eine davon war das »Forking« von Projekten und Repositories. Es stellte sich heraus, dass dieser Vorgang in GitLab eine asynchrone Aktion auslöst, die trotz bestätigender Antwort der API noch nicht vollständig abgeschlossen ist. So wird der Standard-Branch eines Projekts zunächst als »main« angezeigt, bevor er nach Abschluss des Fork-Vorgangs endgültig aktualisiert wird. Dies führte dazu, dass einige Funktionen, wie beispielsweise das Setzen von Branch-Schutzregeln, nicht sofort angewendet werden konnten.

Da nicht exakt bestimmbar war, wann GitLab den Fork-Vorgang vollständig abgeschlossen hatte, wurde in der Anwendung ein asynchrones Monitoring des Status implementiert. Der Status des Forks wird über einen Zeitraum von fünf Minuten hinweg in regelmäßigen Abständen aktualisiert, um sicherzustellen, dass alle erforderlichen Aktionen, einschließlich der Branch-Protection, korrekt ausgeführt wurden.

Verwendung von GORM zur objektrelationalen Abbildung Die Verwendung von `GORM` zur objektrelationalen Abbildung führte zu mehreren technischen Herausforderungen. Ein zentrales Problem war die unvollständige Generierung von Assoziationen, insbesondere bei automatisch generierten typsicheren Query-Funktionen. Dies führte dazu, dass wichtige Beziehungen zwischen Datenbankentitäten nicht ohne manuelle Ergänzungen abgerufen werden konnten. Dieser Fehler ist bereits bekannt, aber nur in chinesischer Sprache dokumentiert und mit einer manuellen Lösung versehen.

Zusätzlich unterstützte `GORM` nur explizites Preloading, was bedeutete, dass Assoziationen manuell geladen werden mussten, anstatt automatisch im Zuge der Abfragen berücksichtigt zu werden. Dies erhöhte den Aufwand bei der Verwaltung komplexer Datenbankoperationen.

Die Testbarkeit des Codes stellte eine weitere Schwierigkeit dar. Unit-Tests konnten nur eingeschränkt genutzt werden, da keine geeigneten Schnittstellen für »Mocking« vorhanden sind. Um die benötigte Funktionalität zuverlässig zu testen, waren Integrationstests unumgänglich.

Weiterhin stellte die Auto-Migration von `GORM` eine Herausforderung dar, da der gewünschte Zustand der Datenbank nicht immer vollständig erreicht wird. So werden beispielsweise keine bestehenden Felder gelöscht, und einige Änderungen an Constraints werden nicht korrekt aktualisiert. Dies führte in der Anwendung zu Abstürzen, wenn etwa bei der Neuerstellung eines Objekts ein `NOT NULL`-Feld in der Datenbank vorhanden ist, aber im Schema nicht mehr existiert. Dadurch konnten bestimmte Operationen nicht mehr ausgeführt werden. Da keine inkrementellen Migrationen möglich sind, bei denen ein Feld zunächst auf `NULL` gesetzt und anschließend gelöscht wird, wurde immer versucht, vom aktuellen Datenbankzustand – egal welcher Version – direkt auf die neue Version zu migrieren. Um dieses Problem zu lösen, wurde ein eigenes Tool entwickelt, wie in Kapitel 3.4.3 (siehe Seite 30) beschrieben. Es ermöglicht die Erstellung von Migrationen, und die Automigration von `GORM` wurde auf das Tool `Goose` umgestellt.

Vollständige Überarbeitung der Backend-API Im Verlauf des Projekts wurde die Backend-API zweimal grundlegend überarbeitet, um neuen Zielen und geänderten Anforderungen gerecht zu werden. Die erste Version bot nur grundlegende Funktionalitäten. Die zweite Version fokussierte sich auf die Arbeit in Teams, war jedoch zu komplex, da die Routen für erstellte und beigetretene Klassenräume getrennt waren. In der finalen Überarbeitung wurden alle Endpunkte vereinheitlicht, die Routen für Mitglieder und Ersteller zusammengeführt und verschiedene Middlewares für eine kontextbezogene Berechtigungsprüfung eingeführt.

Universelle Lösung für automatische Bewertung Es wurde ein Konzept angestrebt, das mit einer Vielzahl von Programmiersprachen und Projektarten kompatibel ist. Verschiedene Ansätze wurden geprüft, darunter die Ausführung von Tests direkt auf der »GitClassrooms«-Plattform sowie die manuelle Auswertung von Testergebnissen. Das »Unit test teport«-Feature von GitLab, das anfangs in der GitLab-API-Dokumentation nicht sofort ersichtlich war, stellte sich letztlich als die geeignetste Methode heraus, da es eine standardisierte und gut dokumentierte Möglichkeit bietet, Testergebnisse aus CI/CD-Pipelines zu generieren und auszuwerten. Die Ergebnisse der Testläufe werden übersichtlich im GitLab-Interface dargestellt, was Lehrenden eine klare Übersicht über die Leistung der Studierenden bietet. Fehlgeschlagene Tests, erfolgreiche Tests und Details zu den einzelnen Testfällen sind direkt einsehbar, was eine einfache und detaillierte Fehleranalyse ermöglicht. Gleichzeitig kann »GitClassrooms« diese Daten über die GitLab-API abrufen und zum Zwecke der automatischen Bewertung verwenden. Auch benötigt die Anwendung damit keinerlei zusätzliche

Funktionen oder Ressourcen, um Tests selbstständig auszuführen bzw. zu evaluieren.

Synchronisation mit GitLab Die Synchronisation zwischen der Anwendung und GitLab stellte eine zusätzliche Herausforderung dar, insbesondere aufgrund der Einschränkungen, die durch die Verwendung der Open-Source-Version von GitLab entstehen. In der Premium-Edition von GitLab stehen Gruppen-Webhooks zur Verfügung, mit denen Änderungen innerhalb von GitLab übertragen und damit überwacht werden könnten. Da jedoch in der Open-Source-Version bzw. Community-Edition diese nicht zur Verfügung stehen, musste eine alternative Lösung entwickelt werden, um die Anwendung weiterhin mit allen GitLab-Editionen verwenden zu können.

Um dieses Problem zu lösen, wurde ein worker-basierter Ansatz gewählt. Diese wurden implementiert, um GitLab in regelmäßigen Abständen auf Änderungen zu überprüfen. Diese Worker führen periodische Anfragen an die GitLab-API aus, um den aktuellen Zustand der Klassenräume, Projekte, Teams und Benutzer abzufragen (siehe Seite 34).

6.1.2 Teambezogene Herausforderungen

Verschiedenes Vorwissen der Entwickler Eine der größten Herausforderungen zu Beginn des Projekts war das unterschiedliche Vorwissen der einzelnen Entwickler. Einige Teammitglieder konnten bereits Erfahrung in der Entwicklung von Webanwendungen sammeln, während andere noch nie an einem vergleichbaren Projekt beteiligt waren. Darüber hinaus variierte auch der technische Hintergrund der Teilnehmer stark, was die Wahl eines gemeinsamen Tech-Stacks erschwerte. Nachdem eine Einigung auf Go für das Backend und `React` für das Frontend erzielt wurde, standen mehrere Entwickler vor der Herausforderung, sich in diese für sie neuen Technologien und Konzepte einzuarbeiten.

Um diesen unterschiedlichen Wissensständen Rechnung zu tragen, wurde in den ersten Wochen des Projekts ein intensives Selbststudium zu den gewählten Technologien angesetzt. Dieser Zeitraum war zwar nicht ausreichend, um bei allen Teammitgliedern einen sicheren Umgang mit diesen Technologien zu etablieren, jedoch konnte dadurch die nötige Basis für den Projektstart gelegt werden. Es war zudem hilfreich, dass viele der Entwickler zeitgleich die Vorlesung `Cloud-Engineering` belegten, die ebenfalls Go und einige Cloud-Konzepte behandelte, was den Lernprozess unterstützte.
Außerdem wurde gelegentlich bei komplexeren Aufgaben im Pair Programming gearbeitet, um nicht so erfahrene Entwickler bei ihrer Arbeit zu unterstützen

Synchronisation zwischen Frontend und Backend Während der Entwicklung war die Synchronisation der Funktionalitäten und Daten zwischen dem Frontend und dem Backend der Anwendung zeitweise problematisch. Während

sich das Backend schnell entwickelte und neue Features hinzufügte, war der Entwicklungsstand des Frontends oft zurückliegend. Dies führte zu Schwierigkeiten bei der Durchführung bei der Überprüfung der UX und stellte zusätzlich die Herausforderung, sicherzustellen, dass das Frontend nur die Daten erhält und verarbeitet, die es tatsächlich benötigt. Dies führte später zu wiederholten Notwendigkeit, Anpassungen sowohl an Frontend als auch Backend durchzuführen, um beide Systeme miteinander zu synchronisieren.

Unterschiedlich starkes Engagement Die regelmäßigen wöchentlichen Meetings stellten sicher, dass alle Teammitglieder kontinuierlich zum Projektfortschritt beitragen konnten und ihre zugewiesenen Aufgaben bearbeiteten. Dabei wurde jedoch deutlich, dass das Engagement und der Arbeitsaufwand einzelner Entwickler deutlich über dem Durchschnitt lagen. Diese Entwickler übernahmen nicht nur die ihnen zugewiesenen Aufgaben, sondern zeigten auch proaktive Initiative bei der Gestaltung der Projektarchitektur und dem Voranschreiten der Anwendung. Dieser Einsatz soll an dieser Stelle einmal gewürdigt werden.

7 Fazit und Ausblick

Im Verlauf des Projekts »GitClassrooms« wurde erfolgreich die erste Version einer Anwendung entwickelt, die die Verwaltung von Programmieraufgaben in Bildungseinrichtungen in Kombination mit dem Versionskontrollsystem GitLab erleichtert. Es bietet eine Plattform, die Lehrenden die Verwaltung von Kursen, Aufgaben und Bewertungen in einem GitLab-Umfeld ermöglicht. Die Implementierung der wesentlichen funktionalen Anforderungen, wie die Verwaltung von digitalen Klassenzimmern, die automatisierte und manuelle Bewertung der eingereichten Programmieraufgaben sowie die rollenbasierte Teamverwaltung, haben das Potenzial, eine Unterstützung für den Lehrbetrieb im Bereich der Programmierung zu sein. Die im Verlauf des Projekts identifizierten technischen und organisatorischen Herausforderungen wurden erfolgreich adressiert und das Projekt bietet eine valide Grundlage für zukünftige Weiterentwicklungen.

Darüber hinaus liegt der Fokus in den nächsten Entwicklungsschritten auf der Erweiterung der Funktionalitäten, da es noch mehrere Bereiche innerhalb der Anwendung gibt, die noch verbessert oder ausgebaut werden müssen. Daher ist zukünftig vorgesehen, das Projekt als Open-Source-Lösung unter der Mozilla Public License 2.0 (MPL 2.0) oder einer ähnlichen Lizenz zu veröffentlichen und die Arbeit daran fortzusetzen. Dies bietet die Möglichkeit, »GitClassrooms« einer breiteren Nutzerschaft zugänglich zu machen und eventuell auch mit der Unterstützung von Dritten eine kontinuierliche Weiterentwicklung zu beginnen.

7.1 Ausblick

Im Folgenden werden mögliche zukünftige Verbesserungen, Entwicklungen und Erweiterungen der Anwendung diskutiert, die darauf abzielen, »GitClassrooms« funktional zu erweitern und besser in bestehende Systeme zu integrieren.

Einführung von Routen zum Löschen von Ressourcen Derzeit können Ressourcen, wie beispielsweise Aufgaben in einem Klassenraum, nur bearbeitet, aber nicht gelöscht werden. In Zukunft soll es auch möglich sein, diese Ressourcen zu entfernen. Da diese Art von Vorgängen mit dem Risiko des Datenverlustes verbunden sind, muss die Art der Umsetzung sorgfältig geprüft werden.

Anbindung an weitere Git-Lösungen wie Gitea und GitHub Eine potenzielle Weiterentwicklung betrifft die Anpassung an andere oder weitere Versionskontrollsysteme. Dies könnte durch eine Abstraktion der GitLab-Schnittstelle erfolgen, sodass eine Integration weiterer Plattformen wie Gitea oder GitHub ermöglicht wird. Die Verwaltung mehrerer solcher Anbindungen in einem Backend-System würde es ermöglichen, für jeden Klassenraum individuell zu bestimmen, welches Versionskontrollsystem verwendet wird. Dies würde die Nutzbarkeit in unterschiedlichen institutionellen Umgebungen fördern.

Verbesserung der Synchronisation mit Webhooks Da GitLab in der kostenlosen Version keine Gruppen-Webhooks unterstützt, wurde die Synchronisation bisher manuell durchgeführt. Es wäre jedoch wünschenswert, Webhooks zu nutzen, um auch in Echtzeit auf Änderungen reagieren zu können, anstatt sie erst im nächsten Synchronisationszyklus festzustellen.

Überarbeitetes Logging Das derzeit implementierte Logging beschränkt sich weitgehend auf das grundlegende Request-Logging. Ein zukünftiger Ausbau der Logging-Funktionalitäten durch die Nutzung des Go-Paketes slog könnte eine feinere Granularität in der Fehlerprotokollierung und eine differenzierte Unterscheidung von Log-Leveln ermöglichen. Dies könnte bei der Fehlersuche einen schnelleren und präziseren Einblick ermöglichen.

Überarbeitung der Authentifizierung Bisher erfolgt die Authentifizierung über Session-Cookies, die in der Datenbank gespeichert werden. Es sollten zukünftig `JWT`-Tokens im Autorisations-Header verwendet werden, um zukünftige Clients verschiedener Art einfacher integrieren zu können. Dadurch könnte auch der Einsatz von `CSRF`-Tokens entfallen.

Eigene CLI zur Automatisierung von Prozessen Die Einführung eines eigenen CLI-Tools wäre denkbar, um Prozesse wie das Klonen von Repositories oder das Herunterladen von Berichten zu automatisieren. Dies würde administrative Aufgaben vereinfachen.

Digitales Benutzerhandbuch Neben der automatisch generierten `OpenAPI`-Dokumentation sollte ein gehostetes Benutzerhandbuch bereitgestellt werden. Dieses Handbuch sollte Informationen zur Benutzung, Entwicklung und zum Betrieb der Software für Benutzer, Entwickler und Administratoren umfassen.

Website zur Vorstellung von »GitClassrooms« Derzeit existiert keine eigene Landing-Page oder Website, die das Projekt »GitClassrooms« der Öffentlichkeit vorstellt. In Zukunft sollte eine solche Website entwickelt werden, um das Projekt besser zu präsentieren.

Erweiterung der Tests Der aktuelle Teststand deckt die erfolgreichen Nutzungsszenarien der verschiedenen Endpunkte ab. Zukünftig sollten auch Fehlerfälle umfassender in die Testabdeckung integriert werden, um Regressionen bei Weiterentwicklungen zu vermeiden.

Verbesserung der Code-Qualität Zum Ende der Projektlaufzeit lag der Fokus stark auf der Implementierung der geplanten Features, um das Projekt fristgerecht abzuschließen. In einem nächsten Schritt wird es wichtig sein, diese Features weiter zu verfeinern, um die Code-Qualität nachhaltig zu verbessern.

Dies kann durch die Anwendung von Best Practices in `React` und den eingesetzten Paketen erreicht werden, was sowohl die Lesbarkeit als auch die Performance des Codes steigern wird.

Anbindung an Lernmanagement-Systeme Um die Software in Zukunft noch nutzerfreundlicher für Hochschulen und Universitäten zu gestalten, wäre die Entwicklung von Anbindungen zu bekannten Lernmanagement-Systemen wie »Stud.IP« und »Moodle« denkbar, um beispielsweise die Einladung von Kursteilnehmern zu vereinfachen.

Helm Chart Um das Deployment von »GitClassrooms« in Kubernetes-Clustern zu erleichtern, wäre die Bereitstellung eines Helm Charts eine sinnvolle Erweiterung. Helm, ein Paketmanager für Kubernetes, erlaubt es, Anwendungen und ihre Abhängigkeiten in wiederverwendbare und konfigurierbare Pakete zu verpacken. Mit einem Helm Chart könnten Benutzer die »GitClassrooms«-Anwendung einfach in ihre Kubernetes-Umgebung integrieren, indem sie eine standardisierte und dokumentierte Bereitstellungskonfiguration nutzen. Dies würde es ermöglichen, verschiedene Deployment-Optionen wie die Anzahl der Repliken oder die Zuweisung von Ressourcen (z.B. CPU und Speicher) zu definieren, was insbesondere für große Infrastrukturen hilfreich ist.

Einbettung von Sentry Für die Optimierung der Fehlersuche und -behebung in der Anwendung wäre die Integration von `Sentry`, einem Monitoring-Tool für Fehler- und Performance-Management, eine sinnvolle Erweiterung. `Sentry` bietet Echtzeit-Überwachung von Applikationen und kann Fehler direkt an das Entwicklerteam melden, einschließlich detaillierter Informationen über die betroffenen Codezeilen und die Umstände, die zum Fehler geführt haben. Diese Informationen könnten den Prozess der Fehleranalyse erheblich beschleunigen und gleichzeitig die Zuverlässigkeit der Anwendung verbessern. Darüber hinaus bietet `Sentry` die Möglichkeit, Performance-Daten zu sammeln, um Engpässe in der Anwendung frühzeitig zu erkennen und zu beheben.

Appendix

Glossar

CI Continuous Integration (CI) ist ein Softwareentwicklungsprozess, bei dem Codeänderungen kontinuierlich in ein gemeinsames Repository integriert und automatisch getestet werden.

CD Continuous Delivery bedeutet, dass der Code, nachdem er die CI-Phase durchlaufen hat, automatisch in eine produktionsähnliche Umgebung bereitgestellt wird.

CI-/CD-Pipeline Eine CI/CD-Pipeline ist ein automatisierter Workflow in der Softwareentwicklung, der die Prozesse der Continuous Integration (CI) und Continuous Delivery (CD) oder Continuous Deployment (CD) kombiniert, um die Qualität und Geschwindigkeit der Softwarebereitstellung zu verbessern.

CLI Eine Command-Line Interface (CLI), auch Kommandozeilen-Schnittstelle genannt, ist eine textbasierte Benutzerschnittstelle, über die Benutzer direkt mit einem Computer oder einer Software durch das Eingeben von Befehlen interagieren.

CSRF Cross-Site Request Forgery. Es ist eine Art von Sicherheitsanfälligkeit im Web, bei der ein Angreifer versucht, einen autorisierten Benutzer dazu zu bringen, unbeabsichtigt Aktionen auf einer Webseite auszuführen, auf der er eingeloggt ist.

Dependency Injection Ein Entwurfsmuster in der Softwareentwicklung, das dazu dient, Abhängigkeiten zwischen Klassen und Modulen zu entkoppeln. Anstatt dass eine Klasse ihre benötigten Abhängigkeiten selbst erstellt, werden diese von außen bereitgestellt (»injiziert«).

JavaScript eine weitverbreitete Programmiersprache, die hauptsächlich für die Entwicklung von dynamischen Webseiten verwendet wird.

JWT JSON Web Token ist ein offener Standard (RFC 7519) für die sichere Übertragung von Informationen zwischen zwei Parteien als JSON-Objekt.

Let's Encrypt »Let's Encrypt« ist eine kostenlose, automatisierte und offene Zertifizierungsstelle (Certificate Authority, CA), die TLS-Zertifikate für Websites ausstellt, um sichere HTTPS-Verbindungen zu ermöglichen.

MPL Die Mozilla Public License (MPL) ist eine Open-Source-Lizenz, die von der Mozilla Foundation entwickelt wurde.

Middleware Im Web-Entwicklungsumfeld handelt es sich um Teile von Programmen, die in der Anfragenverarbeitungskette eines Servers zwischengeschaltet sind. Sie können Anfragen modifizieren, Authentifizierung durchführen oder andere Aufgaben erledigen, bevor die Anfrage die Hauptlogik der Anwendung erreicht.

OAuth2 OAuth 2.0 ist ein offenes Protokoll, das es Anwendungen ermöglicht, im Namen eines Benutzers sicher auf Ressourcen auf einem Server zuzugreifen, ohne die Anmeldeinformationen des Benutzers direkt zu übermitteln. Es verwendet Access Tokens zur Authentifizierung und Autorisierung.

PKCE PKCE steht für Proof Key for Code Exchange. Es ist eine Erweiterung des OAuth 2.0-Standards. Besonders geeignet für Clients die keine sicheren Möglichkeiten haben, Client-Geheimnisse zu speichern

ORM Object-Relational Mapping, ist eine Programmiertechnik, die es Entwicklern ermöglicht, Daten aus einer relationalen Datenbank als Objekte oder Strukturen in ihrer Programmiersprache zu behandeln.

REST Representational State Transfer

Reverse Proxy Ein Reverse Proxy ist ein Server, der Anfragen von Clients entgegennimmt und sie an einen oder mehrere Backend-Server weiterleitet.

SQL Structured Query Language, eine standardisierte Programmiersprache für die Verwaltung von relationalen Datenbanken.

Traefik Traefik ist ein dynamischer Reverse-Proxy und Lastverteiler, der speziell für Microservices und containerisierte Anwendungen entwickelt wurde. Er integriert sich nahtlos mit Orchestrierungssystemen wie Docker, Kubernetes, Consul oder Nomad und bietet eine automatische Erkennung von Diensten, wodurch manuelle Konfigurationsaufwände minimiert werden.

UI User Interface (UI) bezeichnet die grafische und interaktive Oberfläche eines Systems, mit der Nutzer direkt interagieren.

UX User Experience (UX) bezieht sich auf das Gesamterlebnis eines Nutzers bei der Interaktion mit einem Produkt, einer Dienstleistung oder einem System.

Literatur und Quellen

@motatoes, Mohamed. *GitLab Classrooms Feature Proposal*. https://gitlab.com/gitlab-org/gitlab/-/issues/16923. GitLab Issue #16923. Juni 2020. URL: https://gitlab.com/gitlab-org/gitlab/-/issues/16923 (besucht am 20. 09. 2024).

al., Mark Tareshawty et. *GitHub Classroom*. https://github.com/github-education-resources/classroom. 2024-09-17.

Benante, Ruben Carlo. *Classroom GitLab Pages Repository*. https://gitlab.com/classroom/classroom.gitlab.io. GitLab repository. 2018. URL: https://gitlab.com/classroom/classroom.gitlab.io (besucht am 20. 09. 2024).

Joly, Clément. *Telecomnancy Web: Github Education-like dashboard for GitLab*. https://github.com/cljoly/telecomnancy-web. GitHub repository, archived on August 23, 2021. 2021. URL: https://github.com/cljoly/telecomnancy-web (besucht am 20. 09. 2024).

Tabellenverzeichnis

Abbildungsverzeichnis