

Umzug der Übungsabgaben in der Programmierung von Github zu GitLab

Sebastian Brenke

Bachelorarbeit

Vorgelegt am Lehrstuhl für Rechnernetze.

Abgabe der Arbeit: 04. Juni 2026

Erstgutachten: Dr. Markus Brenneis

Zweitgutachten: Dr. Daniel Schmidt

Zusammenfassung

Die Lehrveranstaltung „Programmierung“ ist ein Pflichtmodul für alle Informatik-Studierenden im ersten Semester an der Heinrich-Heine-Universität Düsseldorf und wird jedes Wintersemester von rund 500 Studierenden belegt. Bisher wurde die Verteilung der praktischen Programmierübungen über GitHub Classroom abgewickelt, einen Dienst, der Studierenden per Einladungslink automatisch persönliche Kopien von Template-Repositories bereitstellt.

Nach einer Änderung der Preisstruktur von GitHub entschied der Lehrstuhl, auf die universitätseigene GitLab-Instanz (`git.hhu.de`) umzuziehen. GitLab bietet jedoch keine vergleichbare Classroom-Funktionalität, sodass die Verwaltung der Studierenden-Repositories ohne ein ergänzendes Tool einen enormen Mehraufwand für den betreuenden Lehrstuhl bedeutet hätte.

Im Rahmen dieser Bachelorarbeit wurde die *GitLab Classroom App* entwickelt, eine Spring Boot-Webanwendung, die die relevanten Funktionen von GitHub Classroom für eine GitLab-Instanz anbietet. Lehrende hinterlegen die URLs von Template-Repositories in einer YAML-Konfigurationsdatei und verteilen aufgabenspezifische Einladungslinks. Wenn Studierende einen solchen Link öffnen, authentifiziert die Anwendung sie über das GitLab, erstellt automatisch eine persönliche Repository-Kopie aus dem Template, vergibt die Zugriffsrechte und leitet direkt zum neuen Repository weiter. Zusätzlich zu den Classroom Funktionen ist die Anwendung in der Lage eine DSGVO-konforme Einwilligung zur Datenauswertung einzuholen, die es dem Lehrstuhl erlaubt, den Zusammenhang zwischen der Beteiligung an praktischen Übungen und dem Abschneiden in der Abschlussklausur zu untersuchen.

Das Ergebnis dieser Arbeit ist eine einsatzbereite Anwendung mit bewusst fokussiertem Funktionsumfang, die dadurch mit geringem Aufwand wartbar bleibt. Die Anwendung wird mittels der Container-Technologie Docker bereitgestellt und der produktive Ersteinsatz an der HHU ist für das Wintersemester 2026/27 geplant.

Inhaltsverzeichnis

Abbildungsverzeichnis	v
Abkürzungsverzeichnis	vi
1 Einleitung & Motivation	1
2 Related Work	3
2.1 GitHub Classroom	3
2.2 GitLab-Bordmittel als mögliche Lösung	4
2.3 Bestehende Open-Source-Alternativen	4
2.4 Automatisierung per Skript	5
2.5 Abgrenzung und Begründung der Eigenentwicklung	5
3 Anforderungen	6
3.1 Funktionale Anforderungen	6
3.2 Datenschutzrechtliche Anforderungen	7
3.3 Rahmenbedingungen	8
4 Methoden & Entscheidungen	9
4.1 Iterative Entwicklung	9
4.2 Technische Entscheidungen	10
4.2.1 Technologiewahl	10
4.2.2 GitLab-Zugriff: Service Account und Group Access Token	10
4.2.3 Entwurf des UI-Flows	11
4.2.4 Duplikatbehandlung (Repository-Existenzprüfung)	11
5 Ergebnisse	13
5.1 Umgesetzte Funktionen	13

5.2	Tests	17
5.2.1	Teststrategie	17
5.2.2	Automatisierte Tests	17
5.3	Nicht umgesetzte Funktionen	17
6	Fazit	19
6.1	Reflexion und Lessons Learned	19
6.2	Ausblick	20
	Literatur	21

Abbildungsverzeichnis

5.1	Vollständiger Ablauf des Aufgaben-Workflows	14
-----	---	----

Abkürzungsverzeichnis

API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Deployment
CSV	Comma-Separated Values
DSGVO	Datenschutz-Grundverordnung
HHU	Heinrich-Heine-Universität Düsseldorf
HTTP	Hypertext Transfer Protocol
JPA	Jakarta Persistence API
OAuth2	Open Authorization 2.0
PAT	Personal Access Token
REST	Representational State Transfer
UI	User Interface
URL	Uniform Resource Locator
YAML	YAML Ain't Markup Language

Kapitel 1

Einleitung & Motivation

Die Lehrveranstaltung „Programmierung“ ist ein Pflichtmodul für alle Informatik-Studierenden im ersten Semester an der HHU und wird jedes Wintersemester von rund 500 Studierenden belegt. Kernbestandteil sind praktische Programmierübungen, bei denen Studierende Aufgaben in eigenen Git-Repositories bearbeiten. Git ist ein verteiltes Versionsverwaltungssystem, das Änderungen am Quellcode nachvollziehbar festhält und das gemeinsame Arbeiten an Software ermöglicht.

Für die Verteilung dieser Repositories wurde bisher GitHub Classroom genutzt. Lehrende hinterlegen je Aufgabe ein Template-Repository, Studierende erhalten einen Einladungslink und gelangen nach einem Klick und einem GitHub-Login automatisch zu ihrem persönlichen Repository. Diese einfache Handhabung erleichtert Studienanfängern ohne Vorerfahrung in Versionsverwaltungssystemen den Start in die Veranstaltung. Für die Lehrenden war GitHub Classroom zudem ohne zusätzliche Kosten nutzbar und ließ sich mit geringem Verwaltungsaufwand betreiben.

Am 16.12.2025 verkündete GitHub, dass ab dem 01.03.2026 auch self-hosted runner, welche die Infrastruktur für eigene CI/CD-Pipelines (Continuous Integration/Continuous Deployment) bereitstellen, kostenpflichtig werden [Git25]. Der Lehrstuhl entschied daraufhin auf die universitätseigene GitLab-Instanz der HHU (`git.hhu.de`) zu wechseln. Nach negativem Feedback hat GitHub zwar angekündigt diese Entscheidung neu zu evaluieren, während der Erstellung dieser Arbeit wurde jedoch bekannt, dass GitHub Classroom eingestellt wird, sodass eine Ablösung ohnehin notwendig geworden wäre [Git26b].

GitLab bietet jedoch keine vergleichbare Classroom-Funktionalität. Es gibt keinen eingebauten Mechanismus, über den Lehrende Template-Repositories hinterlegen und Studierenden per Einla-

dungslink automatisch persönliche Kopien bereitstellen können. Ohne ein unterstützendes Werkzeug müsste der Lehrstuhl für jede Aufgabe und jeden Studierenden manuell ein Repository anlegen, die Vorlage importieren und Zugriffsrechte vergeben. Bei rund 500 Studierenden und einer Vielzahl an Aufgaben ist dies keine tragbare Option.

Um diese Lücke zu schließen, wurde im Rahmen dieser Arbeit die *GitLab Classroom App* entwickelt, eine Spring Boot-Webanwendung, die den GitHub-Classroom-Workflow auf GitLab überträgt. Die Anwendung ermöglicht es dem Lehrstuhl die Veranstaltung „Programmierung“ mit vergleichbarem Aufwand über das GitLab der HHU zu betreuen, ohne dass die Studierenden mit Git vertraut sein müssen.

Die übrige Arbeit ist wie folgt aufgebaut. Kapitel 2 ordnet die Lösung in bestehende Werkzeuge ein und begründet die Eigenentwicklung. Kapitel 3 hält die funktionalen, datenschutzrechtlichen und technischen Anforderungen fest. Kapitel 4 beschreibt die Vorgehensweise und die wesentlichen technischen Entscheidungen. Kapitel 5 stellt die umgesetzten Funktionen und die Teststrategie vor, bevor Kapitel 6 die Arbeit reflektiert und einen Ausblick gibt.

Kapitel 2

Related Work

Vor der Entwicklung einer eigenen Lösung wurden bestehende Werkzeuge und Plattformen betrachtet, die das Problem der automatischen Repository-Verteilung lösen könnten. Dieses Kapitel stellt GitHub Classroom als direkten Vorgänger vor, untersucht GitLab-eigene Bordmittel, eine externe Open-Source-Alternative sowie eine Skript-Lösung und begründet die Entscheidung zur Eigenentwicklung.

2.1 GitHub Classroom

GitHub Classroom [Git26a] ist ein Dienst von GitHub für die Hochschullehre. Er ermöglicht es Lehrenden, Aufgaben als GitHub-Repositories zu definieren, mit Startcode vorzubereiten und über individuelle Einladungslinks an Studierende zu verteilen.

Beim Aufruf des Links legt GitHub Classroom für den Studierenden automatisch ein eigenes Repository auf Basis des Templates an, ohne dass dieser selbst ein Projekt erstellen muss. Darüber hinaus bietet GitHub Classroom Funktionen wie automatisiertes Testen über GitHub Actions (Autograding), Verwaltung von Studierendenlisten, Gruppenaufgaben sowie eine Visualisierung des Abgabestatus.

GitHub Classroom war im Fach Programmierung der direkte Vorgänger der hier entwickelten Lösung. Das entscheidende Problem für den beschriebenen Anwendungsfall ist die enge Bindung an die GitHub-Plattform. Da GitHub Classroom fest in GitHub integriert ist, ist eine Nutzung mit

einer anderen Git-Instanz, etwa dem institutionellen GitLab der HHU, nicht möglich.

2.2 GitLab-Bordmittel als mögliche Lösung

Neben externen Werkzeugen wurde geprüft, welche Bordmittel GitLab selbst zur Lösung des Problems bietet. Drei Mechanismen wurden untersucht:

Project Templates. GitLab ermöglicht es, Projekte innerhalb einer Gruppe als Templates [Git26c] zu kennzeichnen. Andere Mitglieder können neue Projekte auf Basis dieser Templates anlegen. Allerdings erfordert dieser Mechanismus, dass Studierende selbst aktiv ein Projekt anlegen und dabei das richtige Template auswählen, ein Schritt, der für Erstsemester ohne Git-Erfahrung eine unnötige Hürde darstellt und fehleranfällig ist.

Fork-Mechanismus. GitLab bietet eine Fork-Funktion [Git26d], über die Studierende ein bestehendes Repository in ihren eigenen Account oder eine Gruppe kopieren können. Der entscheidende Nachteil ist auch hier der manuelle Schritt auf der GitLab-Oberfläche, den jeder Studierende selbst ausführen müsste.

Gruppen-Funktionen. GitLab-Gruppen [Git26f] ermöglichen die Verwaltung von Mitgliedschaften und Berechtigungen auf Gruppenebene. Sie bieten jedoch keinen Mechanismus, auf Abruf automatisch ein Repository zu erstellen und Zugriffsrechte zu vergeben oder Kopien für alle Gruppenmitglieder zu erzeugen.

Keiner dieser Mechanismen erlaubt das vollautomatische Erstellen einer individuellen Repository-Kopie für jeden Studierenden ohne manuelle Schritte auf der GitLab-Oberfläche, worauf es für den beschriebenen Anwendungsfall jedoch entscheidend ankommt.

2.3 Bestehende Open-Source-Alternativen

Bei der Suche nach einer geeigneten Lösung für GitLab wurde die *GitLab Classrooms*-Anwendung der Université de Lille [Uni23] betrachtet. Die Anwendung ist Open Source und bildet, ähnlich wie GitHub Classroom, den Workflow der Aufgabenverteilung auf einer GitLab-Instanz ab.

Ein Blick in den Quellcode zeigt jedoch, dass die Anwendung deutlich mehr Funktionen bietet, als für den Einsatz an der HHU benötigt werden. So existieren beispielsweise Klassen wie `QuizAssignment`, die auf Prüfungs- und Quiz-Szenarien ausgerichtet sind und mit dem eigentlichen Ziel, dem einfachen Klonen von Aufgaben-Repositories, nichts zu tun haben.

Zudem stellt der fehlende Support ein grundsätzliches Risiko bei einer fremden Open-Source-Lösung dar. In der Vergangenheit gab es am Lehrstuhl einen Fall, in dem eine intern entwickelte Anwendung größeren Umfangs betrieben wurde, in deren Abhängigkeiten eine Sicherheitslücke bekannt wurde. Die Lösung war so komplex, dass das Schließen der Lücke nicht mehr leistbar war und die Anwendung abgestellt werden musste. Eine externe Open-Source-Lösung wie die der Université de Lille trägt dasselbe strukturelle Risiko. Treten dort in einer Abhängigkeit Sicherheitslücken auf, kann der Lehrstuhl die Anwendung nur mit unverhältnismäßigem Aufwand oder gar nicht warten.

2.4 Automatisierung per Skript

Eine weitere Möglichkeit wäre, die Repository-Erstellung mit einem eigenen Skript über die GitLab-API [Git26h] zu automatisieren. Ein Skript, das vorab für jeden Studierenden ein Repository anlegt, würde jedoch schnell an die Ratenbegrenzung der API stoßen und zahlreiche Repositories erzeugen, die nie genutzt werden.

2.5 Abgrenzung und Begründung der Eigenentwicklung

Keine der betrachteten Alternativen löst das beschriebene Problem ohne wesentliche Einschränkungen. Eine schlanke Eigenentwicklung bietet demgegenüber klare Vorteile. Der Funktionsumfang lässt sich auf das Nötige beschränken, da für die Veranstaltung ausschließlich die automatische Verteilung von Template-Repositories benötigt wird. Es kann auf einen Technologie-Stack gesetzt werden, der an der HHU gelehrt wird. Da viele am Lehrstuhl diesen Stack beherrschen, lässt sich die Anwendung deutlich einfacher warten und erweitern, als es bei einer fremden Codebasis mit unbekannten Technologien der Fall wäre. Abschließend ermöglicht eine dedizierte Anwendung es, die Erstellung der Repositories nur nach Bedarf der Studierenden anzustoßen. Dies verhindert sowohl Konflikte mit dem API-Limit als auch die Erstellung einer Vielzahl ungenutzter Repositories.

Kapitel 3

Anforderungen

Dieses Kapitel beschreibt die Anforderungen, die im Verlauf des Projekts erarbeitet wurden. Sie umfassen funktionale Anforderungen, eine datenschutzrechtliche Anforderung sowie technische und organisatorische Rahmenbedingungen.

3.1 Funktionale Anforderungen

Die Anwendung muss folgende funktionale Anforderungen erfüllen:

- **FA-01 – Automatische Repository-Erstellung:** Beim Aufruf eines Einladungslinks wird automatisch eine persönliche Kopie des konfigurierten Template-Repositories für den eingeloggten Studierenden erstellt.
- **FA-02 – Authentifizierung über GitLab:** Studierende melden sich mit ihrem bestehenden HHU-GitLab-Konto an. Es werden keine separaten Nutzerkonten in der Anwendung angelegt.
- **FA-03 – Aufgabenspezifische Links:** Für jede Aufgabe existiert ein eigener Einladungslink, der den zugehörigen Workflow einleitet.
- **FA-04 – Weiterleitung bei erneutem Aufruf:** Ruft ein Studierender denselben Einladungslink erneut auf, wird er direkt zu seinem bereits existierenden Repository weitergeleitet. Es wird kein Duplikat erstellt.

- **FA-05 – Konfigurierbare Aufgabenverwaltung:** Lehrende können die verfügbaren Aufgaben durch Eintragen von Template-Repository-URLs in einer Konfigurationsdatei verwalten. Für jede eingetragene Aufgabe ist automatisch ein Einladungslink aktiv.

Bewusst außerhalb des Scopes liegen: automatisches Bewerten oder Testen von Abgaben, Abgabefristverwaltung sowie ein Leaderboard oder jede weitere Form der Punkteerfassung. Der Fokus liegt ausschließlich auf dem Schritt der Repository-Verteilung.

3.2 Datenschutzrechtliche Anforderungen

Der Lehrstuhl möchte untersuchen, ob und wie die Beteiligung an den praktischen Übungen das Abschneiden in der Abschlussklausur beeinflusst. Um entsprechende Daten erheben zu dürfen, ist nach der Datenschutz-Grundverordnung [Eur16] eine freiwillige, informierte Einwilligung der Studierenden erforderlich. Daraus ergeben sich folgende Anforderungen an die Anwendung:

- **DA-01 – Freie Entscheidung:** Studierende können die Einwilligung erteilen oder ablehnen. In beiden Fällen wird der reguläre Workflow (Repository-Erstellung und Weiterleitung) fortgesetzt.
- **DA-02 – Persistente Speicherung:** Die Entscheidung (Nutzername, Zustimmung Ja/Nein) wird dauerhaft gespeichert.
- **DA-03 – Einmalige Abfrage:** Eine Abfrage erfolgt nur, wenn für den jeweiligen Studierenden noch keine Entscheidung gespeichert ist.

Der genaue Wortlaut der Einwilligungserklärung ist bewusst nicht Teil dieser Arbeit. Er wird nach Abgabe vom Lehrstuhl gemeinsam mit der Datenschutzbeauftragten der HHU erarbeitet, weshalb die aktuelle Implementierung an dieser Stelle einen Platzhaltertext enthält.

3.3 Rahmenbedingungen

Technische Rahmenbedingungen

- **TR-01 – Ziel-Plattform:** Die Anwendung wird gegen die GitLab-Instanz der HHU (`git.hhu.de`) betrieben.
- **TR-02 – Deployment:** Die Anwendung muss vollständig als Docker-Container auf einem Linux-Server betreibbar sein.
- **TR-03 – Externe Abhängigkeiten:** Die Anwendung darf keine Verbindungen zu Diensten außerhalb der Universität aufbauen.
- **TR-04 – Wartbarkeit:** Der Funktionsumfang und der Abhängigkeitsbaum der Anwendung sollen minimal gehalten werden, damit Sicherheitsupdates zeitnah eingespielt werden können und die Anwendung auch ohne tiefe Systemkenntnis wartbar bleibt.

Organisatorische Rahmenbedingungen

- **OR-01 – Zielgruppe:** Erstsemester-Studierende ohne Git-Vorerfahrung, für die der Workflow ohne technisches Vorwissen nutzbar sein muss.
- **OR-02 – Sprache:** Die Benutzeroberfläche ist auf Deutsch.

Kapitel 4

Methoden & Entscheidungen

Dieses Kapitel beschreibt die Vorgehensweise bei der Entwicklung sowie die wesentlichen technischen Entscheidungen, die die Anwendung prägen.

4.1 Iterative Entwicklung

Die Entwicklung erfolgte iterativ, mit wöchentlichen Treffen zur Abstimmung mit dem Betreuer. Dabei wurden der aktuelle Stand besprochen, neue Anforderungen identifiziert und Entscheidungen getroffen. Dass sich mehrere Anforderungen erst im Laufe der Entwicklung konkretisiert haben, etwa die DSGVO-Einwilligungsabfrage, bestätigte diesen Ansatz.

Die Implementierung wurde schrittweise aufgebaut: zunächst ein lauffähiger Kern (Authentifizierung über GitLab, erfolgreiche API-Calls), dann Erweiterungen (konkreter Workflow, Fehlerbehandlung) und schließlich die datenschutzrechtliche Komponente (DSGVO-Einwilligungsabfrage). Dieses Vorgehen ermöglichte es, früh funktionale Zwischenstände vorzustellen und gezieltes Feedback einzuholen.

4.2 Technische Entscheidungen

Im Verlauf des Projekts waren mehrere technische Entscheidungen zu treffen. Die folgenden Abschnitte behandeln die Technologiewahl, den Zugriff auf GitLab, den Entwurf des Benutzer-Flows sowie die Behandlung wiederholter Aufrufe.

4.2.1 Technologiewahl

Als Basis-Framework wurde Spring Boot gewählt. Die Entscheidung fiel bewusst auf diese vertraute Technologie. Spring Boot wird, ebenso wie das Frontend-Framework Thymeleaf, im Informatik-Studium an der HHU gelehrt. Da die Anwendung langfristig am Lehrstuhl betrieben und weiterentwickelt werden soll, war diese Vertrautheit wichtiger als technische Modernität. Ein an der HHU gelehrter Technologie-Stack kann von vielen Personen gewartet und erweitert werden, ohne dass diese sich zunächst in unbekannte Werkzeuge einarbeiten müssen. Die Authentifizierung via GitLab übernimmt Spring Security, der Datenbankzugriff erfolgt über Spring Data JPA, und das serverseitige Rendering der Benutzeroberfläche wird mit Thymeleaf realisiert.

4.2.2 GitLab-Zugriff: Service Account und Group Access Token

Alle Operationen auf der GitLab-Instanz werden über die GitLab REST API v4 [Git26h] ausgeführt. Die Anwendung agiert dabei als *Service Account*, also als technischer Benutzer, der im Namen des Systems und nicht im Namen der eingeloggtten Studierenden handelt.

Für diesen Zweck wird ein *Group Access Token* [Git26e] verwendet, das an die GitLab-Gruppe der Veranstaltung gebunden und unabhängig von einzelnen Nutzeraccounts ist. Die Alternative, ein PAT (Personal Access Token) [Git26g] eines konkreten Nutzers zu verwenden, wurde verworfen. Ein PAT wäre an den persönlichen Account des jeweiligen Nutzers gebunden und würde bei einem Kontowechsel oder einer Kontosperrung den Betrieb der gesamten Anwendung gefährden. Zudem lässt sich ein PAT nicht ohne Weiteres auf eine einzelne Gruppe einschränken, sondern gewährt tendenziell Zugriff auf alle Ressourcen seines Inhabers.

4.2.3 Entwurf des UI-Flows

Der Benutzer-Flow, also die Abfolge der Schritte beim Aufruf eines Einladungslinks, hat im Laufe des Projekts drei Versionen durchlaufen. Die erste Version schaltete zwischen Linkaufruf und Repository-Erstellung eine Bestätigungsseite, auf der die Studierenden die bevorstehende Aktion aktiv bestätigen mussten. Dieser an GitHub Classroom orientierte Schritt erwies sich als unnötig. In der zweiten Version entfiel jede Zwischenseite. Nach dem OAuth2-Login (Open Authorization) [OAu26] wurde das Repository sofort erstellt und der Studierende direkt weitergeleitet. Aus Nutzersicht war dieser Flow optimal, ließ sich aber nicht mit der später hinzugekommenen DSGVO-Einwilligung vereinbaren, die ohne eine sichtbare Benutzeroberfläche nicht umsetzbar ist.

Die dritte und aktuelle Version verbindet beide Anforderungen. Beim der ersten Interaktion mit der Anwendung wird nach dem Login eine Einwilligungsseite angezeigt, auf der Studierende der Datenerhebung zustimmen oder sie ablehnen. Anschließend läuft der reguläre Workflow weiter. Bei allen weiteren Aufrufen entfällt dieser Schritt, da die Entscheidung pro Nutzer in der Datenbanktabelle `consent_given` persistiert ist.

4.2.4 Duplikatbehandlung (Repository-Existenzprüfung)

Eine konkrete technische Herausforderung stellte FA-04 dar. Wenn ein Studierender denselben Einladungslink ein zweites Mal aufruft, soll er zu seinem bereits existierenden Repository weitergeleitet werden, ohne dass ein Duplikat oder ein Fehler entsteht.

Zunächst wurde versucht, die Existenz über den URL-encodierten Projekt-Pfad abzufragen (`GET /projects/gruppe%2FProjektname`). Dieser Ansatz schlug fehl. Die naheliegendste Erklärung ist, dass URL-encodierte Schrägstriche in Pfadsegmenten von der eingesetzten Netzwerkinfrastruktur (Firewall bzw. Reverse Proxy) abgefangen werden und die Anfrage nicht korrekt an die GitLab API weitergeleitet wird. Abschließend verifizieren ließ sich diese Vermutung nicht, da die GitLab-Instanz nicht vom Lehrstuhl betrieben wird. Sie erschien jedoch plausibel genug, um stattdessen einen anderen Weg zu wählen.

Daraufhin wurden drei alternative Strategien erwogen:

1. **Alle Repos der Gruppe abfragen** und lokal auf den erwarteten Namen prüfen. Nachteil: Bei vielen Projekten entstehen durch Pagination viele API-Aufrufe.
2. **Alle Repos des eingeloggten Nutzers abfragen.** Nachteil: Erfordert zusätzliche API-Rechte im Namen des Nutzers statt des Service Accounts.
3. **Repository-Erstellung direkt versuchen und den Fehlerfall behandeln.**

Gewählt wurde Option 3. Wurde bereits ein entsprechendes Repository erstellt, antwortet die GitLab API mit einer eindeutigen Fehlermeldung auf einen erneuten Aufruf mit den gleichen Parametern auf `POST /projects`. Diese Antwort lässt sich zuverlässig als Signal für einen Wiederholungsaufruf erkennen.

Alle erzeugten Repositories liegen in einer fest konfigurierten GitLab-Gruppe und folgen einem einheitlichen Namensschema aus Aufgabenkennung und Nutzernamen (Kapitel 5). Wird der Duplikatfehler erkannt, lässt sich die URL des bestehenden Repositories daher allein aus den bereits bekannten Komponenten lokal zur Form

`<base-url>/<group-name>/Aufgabe-<aufgabenName>-<username>` zusammensetzen, zu dieser der Aufrufende dann weitergeleitet werden kann.

Dieser Ansatz besitzt einen bekannten Randfall. Die URL-Rekonstruktion setzt voraus, dass das Repository nach wie vor nach dem Schema `Aufgabe-{aufgabenName}-{username}` benannt ist. Hat ein Studierender es manuell umbenannt, zeigt die Weiterleitung auf eine ungültige URL. Die Rekonstruktion birgt damit zwar das Risiko, nach einer solchen Umbenennung inkorrekt weiterzuleiten, in der Praxis ist dies bei Erstsemester-Studierenden jedoch äußerst unwahrscheinlich.

Kapitel 5

Ergebnisse

Dieses Kapitel stellt die implementierten Funktionen der GitLab Classroom App vor, beschreibt die gewählte Teststrategie und benennt Punkte, die nicht umgesetzt wurden oder noch offen sind.

Eine ausführliche technische Dokumentation findet sich im Quellcode. Dort liegen eine README als Einrichtungs- und Nutzungsanleitung sowie eine technische Dokumentation nach arc42 bei, die Architektur und Aufbau der Anwendung für eine spätere Wartung und Weiterentwicklung beschreiben.

5.1 Umgesetzte Funktionen

Kernworkflow: Repository-Erstellung

Die zentrale Funktion der Anwendung ist der Endpunkt `GET /aufgaben/{aufgabenName}`, über den der gesamte Aufgaben-Workflow eingeleitet wird. Abbildung 5.1 zeigt den vollständigen Ablauf.

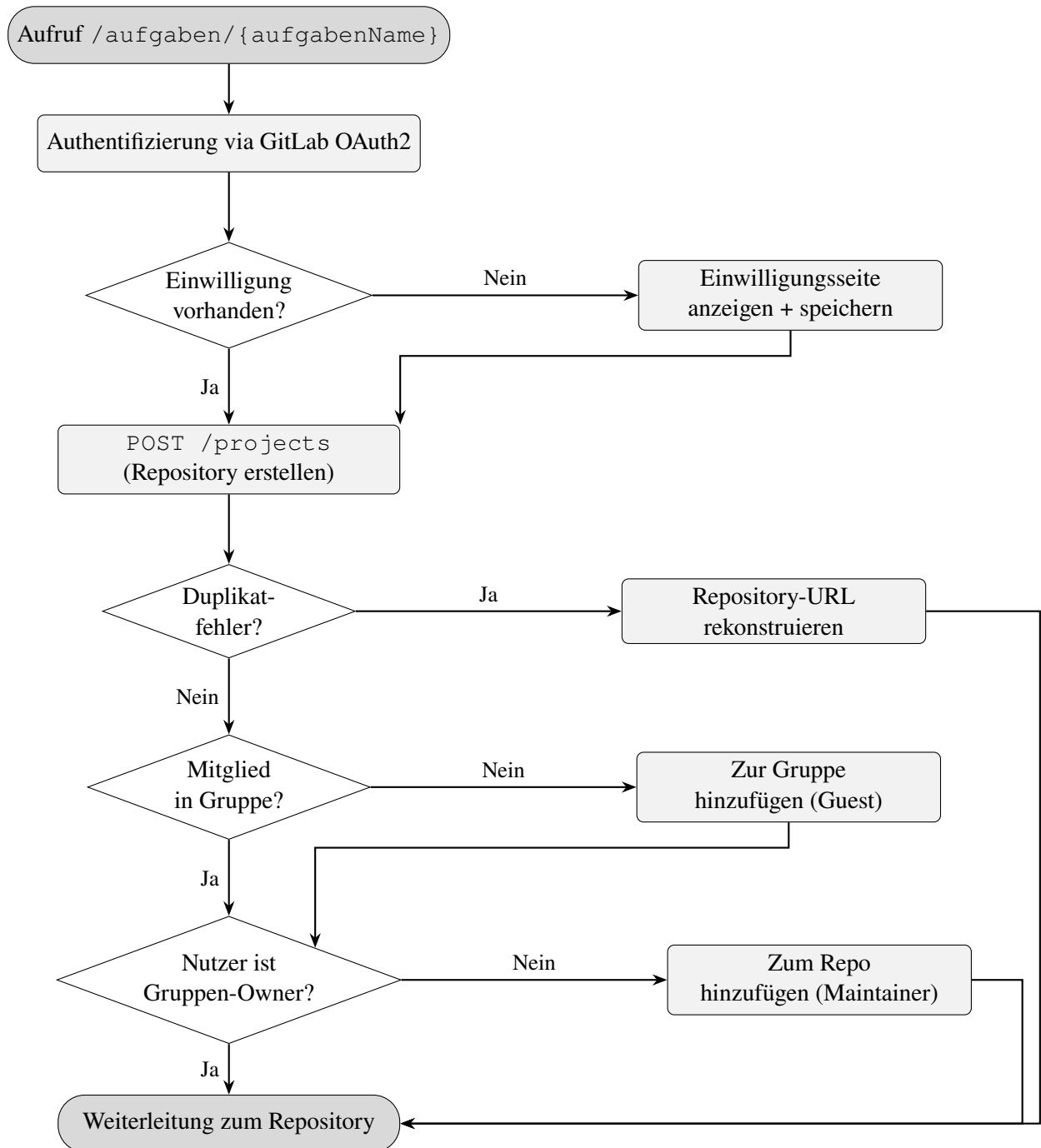


Abbildung 5.1: Vollständiger Ablauf des Aufgaben-Workflows

1. Beim Aufruf des Einladungslinks übernimmt Spring Security die Authentifizierung via GitLab OAuth2. Nicht eingeloggte Studierende werden zunächst zur GitLab-Loginseite weitergeleitet und danach automatisch in den Workflow zurückgeführt.
2. Die Anwendung prüft, ob für den Nutzernamen bereits ein Einwilligungseintrag in der Datenbank vorhanden ist. Ist kein Eintrag vorhanden, wird die Einwilligungsseite angezeigt (GET /datenschutz/{aufgabenName}). Über
POST /datenschutz/zustimmen/{aufgabenName} oder
POST /datenschutz/ablehnen/{aufgabenName} wird die Entscheidung gespeichert und der Workflow fortgesetzt.
3. Die Anwendung ruft POST /projects auf der GitLab API auf und übergibt den Namen des neuen Repositories sowie die Import-URL des Template-Repositories.
4. Gibt die GitLab API zurück, dass Name und Namespace bereits vergeben sind, wird die URL des bestehenden Repositories aus bekannten Parametern rekonstruiert und der Studierende direkt weitergeleitet (siehe Abschnitt 4.2.3).
5. Ist die Erstellung erfolgreich, wird geprüft, ob der Studierende bereits Mitglied der konfigurierten GitLab-Gruppe ist (über den groups-Claim im OpenID Connect-Token). Falls nicht, wird er über POST /groups/:id/members hinzugefügt (Access Level: Guest).
6. Ist der eingeloggte Nutzer der Gruppen-Owner, entfällt der folgende Schritt, da GitLab das Zuweisen einer niedrigeren Rolle an den Owner nicht erlaubt. Andernfalls wird der Studierende über POST /projects/:id/members als Maintainer zum neuen Repository hinzugefügt.
7. Der Studierende wird per HTTP-Redirect zur GitLab-URL des neuen Repositories weitergeleitet.

Repository-Benennung

Repositories werden nach dem Schema Aufgabe-{aufgabenName}-{username} benannt, wobei aufgabenName der Aufgabenkennung aus der Konfiguration und username dem GitLab-Nutzernamen des Studierenden entspricht. Dieses Schema gewährleistet eindeutige Namen innerhalb

der Gruppe und erlaubt es dem Lehrpersonal, Repositories ihren Eigentümern zuzuordnen. Dass die Namen eindeutig und einer Person zuzuordnen bleiben, beruht darauf, dass GitLab-Nutzernamen keine Bindestriche enthalten. Allgemeingültig ist diese Annahme nicht, an der HHU trifft sie jedoch zu.

DSGVO-Einwillungsverwaltung

Die Einwilligungsabfrage ist in den Workflow integriert und tritt nur beim ersten Aufruf auf. Die Entscheidung (Nutzername, Zustimmung Ja/Nein) wird in der PostgreSQL-Datenbank in der Tabelle `consent_given` persistiert.

Für den Gruppen-Owner steht die URL `/admin/datenschutz` zur Verfügung, die eine Liste aller gespeicherten Einwilligungen anzeigt. Über `/admin/datenschutz/export` kann die Liste als CSV-Datei heruntergeladen werden. Der Zugang zu diesen Endpunkten ist auf den konfigurierten Owner-Nutzernamen beschränkt und durch Spring Security abgesichert.

Konfigurierbarkeit

Die gesamte Konfiguration der Anwendung (GitLab-Instanz, Gruppe, Aufgaben-URLs) erfolgt über die Datei `application.yml`. Neue Aufgaben können durch das Hinzufügen eines Schlüssel-Wert-Paares unter `veranstaltung.aufgaben` eingetragen werden, ohne dass Code-Änderungen notwendig sind. Nach einer Konfigurationsänderung ist ein Neustart der Anwendung erforderlich.

Deployment

Die Anwendung wird über Docker Compose betrieben. Das enthaltene `compose.yml` definiert zwei Services: den Spring Boot-Applikationscontainer und einen PostgreSQL-Container. Secrets (Datenbankpasswort, OAuth-Credentials, GitLab-Token) werden über eine `.env`-Datei als Umgebungsvariablen injiziert und nicht in die Versionsverwaltung eingecheckt.

5.2 Tests

5.2.1 Teststrategie

Als primäre Integrationsteststrategie wurde das direkte Testen gegen die echte GitLab-Instanz der HHU (`git.hhu.de`) gewählt. Neue Funktionen und Änderungen wurden dabei unmittelbar gegen die Produktionsinfrastruktur verifiziert, nicht gegen einen Mock oder eine lokale Testinstanz. Das hat sich als wertvoll erwiesen, da bestimmte Fehler, etwa das vermutete Abfangen von URL-encodierten Schrägstrichen durch den Reverse Proxy der Universität, nur auf der realen Infrastruktur sichtbar wurden.

5.2.2 Automatisierte Tests

Ergänzend dazu wurden für `ClassRoomService`, die zentrale Geschäftslogikklasse, Unit-Tests mit JUnit 5 und Mockito implementiert. Die Tests decken die wesentlichen Szenarien des Kernworkflows ab, darunter die Einwilligungsprüfung, die korrekte Parameterübergabe bei der Repository-Erstellung, die Behandlung des Duplikat-Falls sowie die Behandlung unbekannter API-Fehler. Automatisierte Integrationstests, die ohne manuelle Eingriffe gegen GitLab laufen, wurden im Rahmen dieser Arbeit nicht implementiert.

5.3 Nicht umgesetzte Funktionen

Folgende Punkte wurden im Projektverlauf diskutiert, aber bewusst nicht im Rahmen dieser Bachelorarbeit umgesetzt:

- **CI/CD-Skripte für Aufgaben:** Ursprünglich war die Überführung der GitHub-CI/CD-Skripte auf GitLab als weiterer Bestandteil dieser Arbeit geplant. Im Projektverlauf wurde dieser Punkt zugunsten der DSGVO-Einwilligung aus dem Scope genommen und nicht umgesetzt. Er bleibt als mögliche Erweiterung erhalten (Kapitel 6).
- **Leaderboard:** In frühen Planungsdokumenten wurde ein Leaderboard als mögliche Erweite-

rung erwähnt. Dieses Feature wurde bewusst aus dem Scope ausgeschlossen und bleibt als Future Work erhalten.

Kapitel 6

Fazit

Dieses abschließende Kapitel reflektiert den Entwicklungsprozess, leitet Lessons Learned ab und zeigt Möglichkeiten zur Weiterentwicklung auf.

6.1 Reflexion und Lessons Learned

Iterative Entwicklung mit wöchentlicher Abstimmung hat sich als tragfähiges Vorgehensmodell erwiesen. Anforderungen ließen sich schrittweise konkretisieren und Entscheidungen kurzfristig revidieren, ohne die kostspieligen Nachbesserungen eines Wasserfallansatzes. Deutlich wurde das an der DSGVO-Einwilligungsabfrage, die erst im Projektverlauf aufkam und eine Überarbeitung des UI-Flows und eine Datenbankanbindung erforderte. Dank des iterativen Vorgehens ließ sie sich dennoch sauber einarbeiten und früh im echten Betrieb verifizieren.

Wertvoll war zudem das durchgängige Testen gegen die echte GitLab-Instanz der HHU. Ein Problem wie die vermutlich abgefangenen URL-encodierten Schrägstriche wäre lokal nicht aufgefallen und erst im produktiven Einsatz sichtbar geworden. Für dieses Projekt war das direkte Testen gegen die Produktionsumgebung daher wertvoll. In größeren oder mehrköpfigen Projekten wäre es jedoch kein tragfähiges Modell.

Bewährt hat sich schließlich der fokussierte Funktionsumfang. Die Anwendung löst ein klar umrissenes Problem ohne unnötige Komplexität und bleibt durch YAML-Konfiguration und Docker-Compose-Betrieb auch ohne tiefe Kenntnis wartbar. Angesichts früherer Erfahrungen am Lehrstuhl

mit zu komplexen Eigenlösungen wurde dieses Ziel bewusst priorisiert.

6.2 Ausblick

Zum Zeitpunkt der Abgabe ist die GitLab Classroom App noch nicht produktiv im Einsatz. Der Erstbetrieb ist für das Wintersemester 2026/27 vorgesehen. Dafür muss der Einwilligungstext durch den Lehrstuhl gemeinsam mit der Datenschutzbeauftragten der HHU finalisiert werden.

Über den Funktionsumfang dieser Arbeit hinaus ergeben sich mehrere Ansätze zur Weiterentwicklung. Einige davon adressieren bewusst ausgeklammerte Einschränkungen der aktuellen Lösung:

- **CI/CD-Skripte für Aufgaben:** Die Überführung der GitHub-CI/CD-Skripte auf GitLab wurde zugunsten der DSGVO-Einwilligung aus dem Scope genommen. Sie wäre ein naheliegender nächster Schritt, um den ursprünglich geplanten Funktionsumfang zu vervollständigen.
- **Automatisierte Integrationstests:** Das manuelle Testen gegen die echte GitLab-Instanz war für den aktuellen Umfang praktikabel, skaliert aber schlecht. Mit wachsendem Funktionsumfang und mehreren Beteiligten gewinnen automatisierte Tests an Bedeutung.
- **Gruppenabgaben:** Die Anwendung erstellt ausschließlich individuelle Repositories. Gruppenabgaben lagen außerhalb des in Kapitel 3 definierten Scopes. Eine Unterstützung von Team-Repositories würde kooperative Übungsformate über dieselbe Infrastruktur ermöglichen.
- **Mehrere Veranstaltungen:** Aktuell ist die Anwendung für eine einzelne GitLab-Gruppe und eine Veranstaltung konfiguriert. Eine Erweiterung um Mehrmandantenfähigkeit würde den Einsatz für mehrere Kurse gleichzeitig ermöglichen.
- **Leaderboard:** Als optionale Erweiterung könnte ein Leaderboard Studierenden Rückmeldung über ihren Fortschritt geben. Voraussetzung wäre die Anbindung an die CI/CD-Ergebnisse der Repositories.

Literatur

- [Eur16] Europäische Union. *Allgemeine Datenschutz-Grundverordnung (DSGVO)*. 2016. URL: <https://eur-lex.europa.eu/DE/legal-content/summary/general-data-protection-regulation-gdpr.html> (besucht am 04.06.2026).
- [Git25] GitHub, Inc. *Coming soon: Simpler pricing and a better experience for GitHub Actions*. 2025. URL: <https://github.blog/changelog/2025-12-16-coming-soon-simpler-pricing-and-a-better-experience-for-github-actions/> (besucht am 04.06.2026).
- [Git26a] GitHub, Inc. *GitHub Classroom*. 2026. URL: <https://classroom.github.com> (besucht am 04.06.2026).
- [Git26b] GitHub, Inc. *GitHub Classroom sign-ups are no longer available*. 2026. URL: <https://github.blog/changelog/2026-05-26-github-classroom-sign-ups-are-no-longer-available/> (besucht am 04.06.2026).
- [Git26c] GitLab B.V. *Custom group-level project templates*. 2026. URL: https://docs.gitlab.com/user/group/custom_project_templates/ (besucht am 04.06.2026).
- [Git26d] GitLab B.V. *Forking workflow*. 2026. URL: https://docs.gitlab.com/user/project/repository/forking_workflow/ (besucht am 04.06.2026).
- [Git26e] GitLab B.V. *Group access tokens*. 2026. URL: https://docs.gitlab.com/user/group/settings/group_access_tokens/ (besucht am 04.06.2026).
- [Git26f] GitLab B.V. *Groups*. 2026. URL: <https://docs.gitlab.com/user/group/> (besucht am 04.06.2026).
- [Git26g] GitLab B.V. *Personal access tokens*. 2026. URL: https://docs.gitlab.com/user/profile/personal_access_tokens/ (besucht am 04.06.2026).
- [Git26h] GitLab B.V. *REST API*. 2026. URL: <https://docs.gitlab.com/api/rest/> (besucht am 04.06.2026).

- [OAu26] OAuth Working Group. *OAuth 2.0*. 2026. URL: <https://oauth.net/2/> (besucht am 04. 06. 2026).
- [Uni23] Université de Lille. *GitLab Classrooms*. Open-Source-Webanwendung zur Repository-Verteilung auf GitLab-Instanzen. 2023. URL: <https://gitlab.univ-lille.fr/gitlab-classrooms/gitlab-classrooms> (besucht am 04. 06. 2026).

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 04. Juni 2026

Sebastian Brenke

Erklärung zur Nutzung generativer KI

Im Rahmen der vorliegenden Bachelorarbeit wurde generative KI zu folgenden Zwecken genutzt:

Für alle genannten Punkte wurde Claude von Anthropic genutzt:

- Ausarbeitung und Erstellung der schriftlichen Bachelorarbeit. Die Inhalte wurden vom Verfasser inhaltlich vorgegeben, geprüft und angepasst.
- Erstellung der ReadMe und der arc42 Dokumentation. Die Inhalte wurden vom Verfasser inhaltlich vorgegeben, geprüft und angepasst.
- Generierung der Javadocs. Die Inhalte wurden vom Verfasser inhaltlich geprüft und angepasst.

- Erstellung der HTML-Seiten. Die Inhalte wurden vom Verfasser inhaltlich vorgegeben, geprüft und angepasst.

1. Einführung und Ziele

1.1 Aufgabenstellung

Die GitLab Classroom App ist eine Spring Boot-Webanwendung, die den Prozess der Repository-Verteilung für Programmierkurse automatisiert. Ohne das Tool sieht der Ablauf so aus:

- **Für Lehrende:** Repositories manuell für jeden Studierenden anlegen und Rechte vergeben
- **Für Studierende:** Git-Repository selbst einrichten, Zugriffsrechte klären, Abgabe organisieren
- **Für beide:** Viel administrativer Aufwand, der eigentlich nichts mit der eigentlichen Aufgabe zu tun hat

Die App übernimmt das alles automatisch:

1. **Template-Repositories:** Lehrende hinterlegen einmalig ein Template-Repository pro Aufgabe
2. **Automatische Repository-Erstellung:** Studierende klicken einen Einladungslink
3. **Login über GitLab:** Kein neuer Account nötig, der bestehende GitLab-Account wird verwendet
4. **Rechtevergabe:** Passiert automatisch im Hintergrund

Was die App nicht macht

- Code-Review oder automatisches Bewerten
- CI/CD-Pipelines verwalten
- Ein vollständiges Lernmanagementsystem ersetzen

Der Fokus liegt ausschließlich auf der **automatischen Repository-Verteilung**.

1.3 Stakeholder

Rolle	Erwartungen	Interesse an der Architektur
Lehrende (Dozenten, Tutoren)	Einfache Bedienung, minimaler Aufwand	Gering – nur als Endnutzer
Studierende (Kursteilnehmer)	Schneller Zugang, keine technischen Hürden	Sehr gering – nur Nutzung

1.5 Weitere Dokumente

Dokument	Zweck	Für wen
README.md	Schnelleinstieg und Setup	Alle
docs/arc42/	Architekturdokumentation	Entwickler
Inline-Kommentare im Code	Details zu einzelnen Implementierungen	Entwickler

Wo anfangen?

- Neu im Projekt? → [README.md](#)
- Deployment oder Betrieb? → [README.md](#) , Abschnitt „Produktivbetrieb“
- Code verstehen oder anpassen? → Diese arc42-Dokumentation

2. Randbedingungen

Technischer Stack

Technologie	Warum
Java 25 + Spring Boot 3	Spring Boot wird an der HHU gelehrt
Thymeleaf	Thymleaf wird an der HHu gelehrt
PostgreSQL	Daten müssen persistent sein; welche SQL-Datenbank es genau wird, war eher pragmatisch
Docker Compose	So wurde es vom Betreuer gewünscht – alles läuft auf einem Uni-Server
GitLab	Zentraler Anker: OAuth2-Login und Repository-Hosting laufen beide dort

Rahmenbedingungen

- **Nutzerzahlen:** ca. 50–200 Studierende pro Semester
- **Team:** 1–2 Entwickler, oft wechselnd – deshalb ist Wartbarkeit wichtiger als technische Finesse
- **Budget:** Keins – alles Open Source

DSGVO

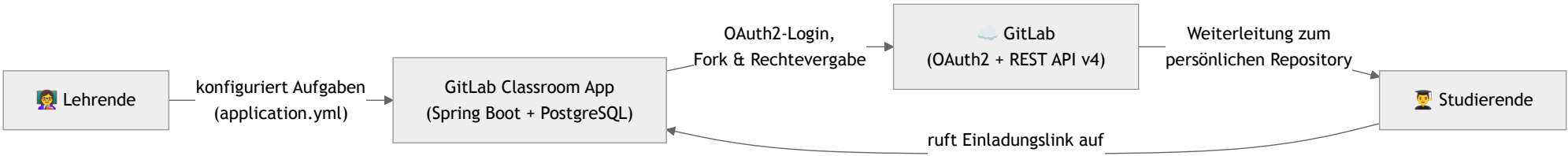
Die App speichert personenbezogene Daten (GitLab-User-ID, Name, E-Mail). Dafür ist eine explizite Einwilligung nötig (Art. 6 Abs. 1 lit. a DSGVO), die beim ersten Login abgefragt wird.

Secrets

Alle sensiblen Werte (DB-Passwort, GitLab-Token, OAuth2-Credentials) kommen ausschließlich aus der `.env`-Datei, die nie in Git eingcheckedt wird.

3. Kontextabgrenzung

GitLab Classroom - Systemkontext



Wer kommuniziert mit der App?

Akteur / System	Was passiert
Lehrende	Tragen Template-Repository-URLs in <code>application.yml</code> ein und geben die Einladungslinks an Studierende weiter
Studierende	Klicken den Einladungslink, loggen sich mit ihrem GitLab-Account ein und landen am Ende im eigenen Repository
GitLab	Übernimmt den Login (OAuth2) und bekommt API-Aufrufe für Fork und Rechtevergabe
PostgreSQL	Speichert die Einwilligungen – läuft intern, ist von außen nicht erreichbar

Was die App nicht macht

- Repositories hosten (das ist Sache von GitLab)
- Code bewerten oder reviewen
- GitLab-Nutzer verwalten – die bestehenden Accounts werden einfach genutzt

4. Lösungsstrategie

Leitprinzipien

1. Bekannte Technologien statt modernster Lösung

Spring Boot und Thymeleaf werden an der HHU in Pflichtveranstaltungen gelehrt. Da die App langfristig am Lehrstuhl betrieben werden soll, war es wichtiger, dass zukünftige Entwickler aus dem Uni-Umfeld direkt loslegen können, als die technisch modernste Lösung zu wählen.

2. Studierende sollen die App kaum merken

Das Tool ist ein Hilfsmittel, damit Studierende GitLab einfacher nutzen können – nicht eine weitere Plattform, bei der sie sich registrieren müssen. Deshalb wird der Login vollständig an GitLab delegiert: kein eigenes Passwort, kein zweiter Account. Der gesamte Ablauf endet mit einer direkten Weiterleitung zum Repository.

3. Komplexität nur wo nötig

Das Projekt ist nicht sonderlich komplex, daher war ein Monolith die naheliegende Wahl. Eine aufwändigere Architektur hätte hier für unverhältnismäßigen Mehraufwand gesorgt, ohne wirklichen Nutzen zu bringen.

4. Vorgaben des Betreuers

Einige Entscheidungen wurden direkt mit meinem Betreuer abgestimmt oder von ihm vorgegeben:

- **Deployment:** Alles muss am Ende als Docker-Container auf einem Uni-Server laufen.
- **Keine externen Verbindungen:** Die App darf keine Verbindungen zu Diensten außerhalb der Uni aufbauen – das gilt auch für Schriftarten oder Bibliotheken in HTML-Seiten.
- **Owner-Unterstützung:** Auch der GitLab-Gruppen-Owner soll die App fehlerfrei nutzen können. GitLab erlaubt es nicht, einem Owner eine niedrigere Rolle in einem Projekt zuzuweisen, weshalb `addUserToProject` für den Owner übersprungen wird.

Technologie-Stack

Frontend: Thymeleaf (Server-Side Rendering, Teil des Spring-Stacks)

Backend: Spring Boot 3 / Java 25

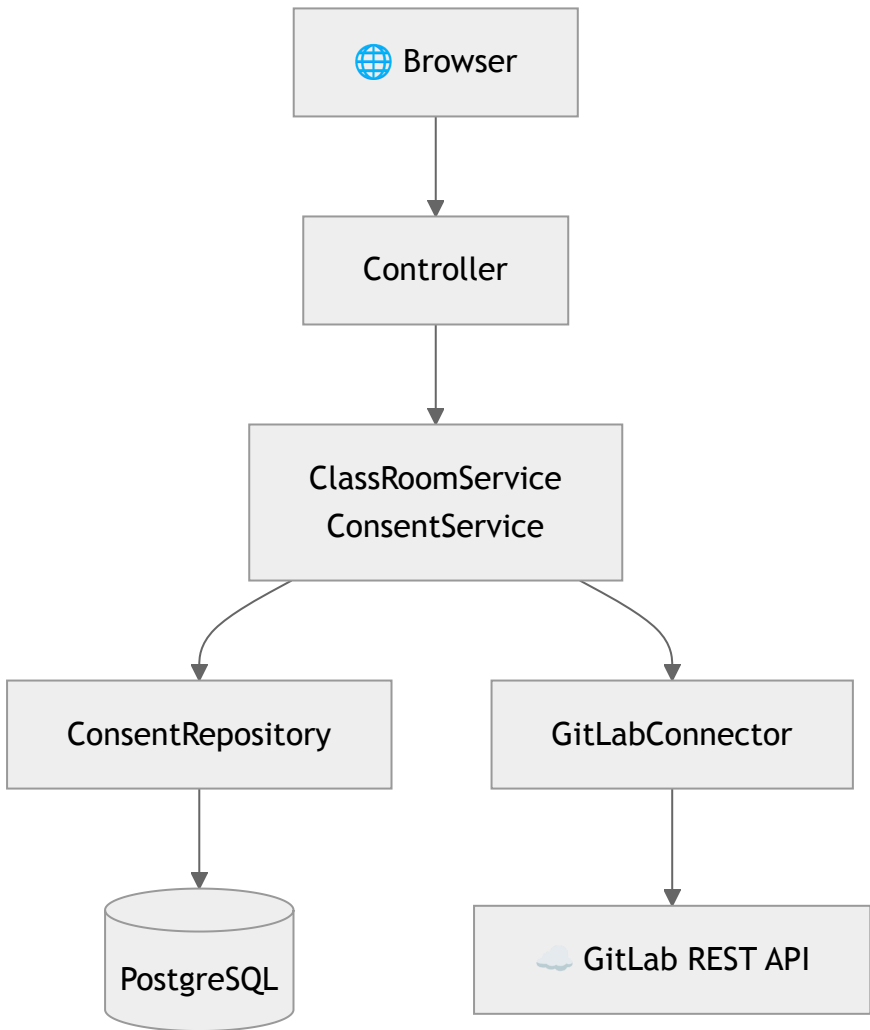
Datenbank: PostgreSQL (via Spring Data JPA / Hibernate)

Deployment: Docker Compose (App-Container + PostgreSQL-Container)

Auth: GitLab OAuth2 / OpenID Connect

Architekturmuster

Klassische 3-Schicht-Architektur:

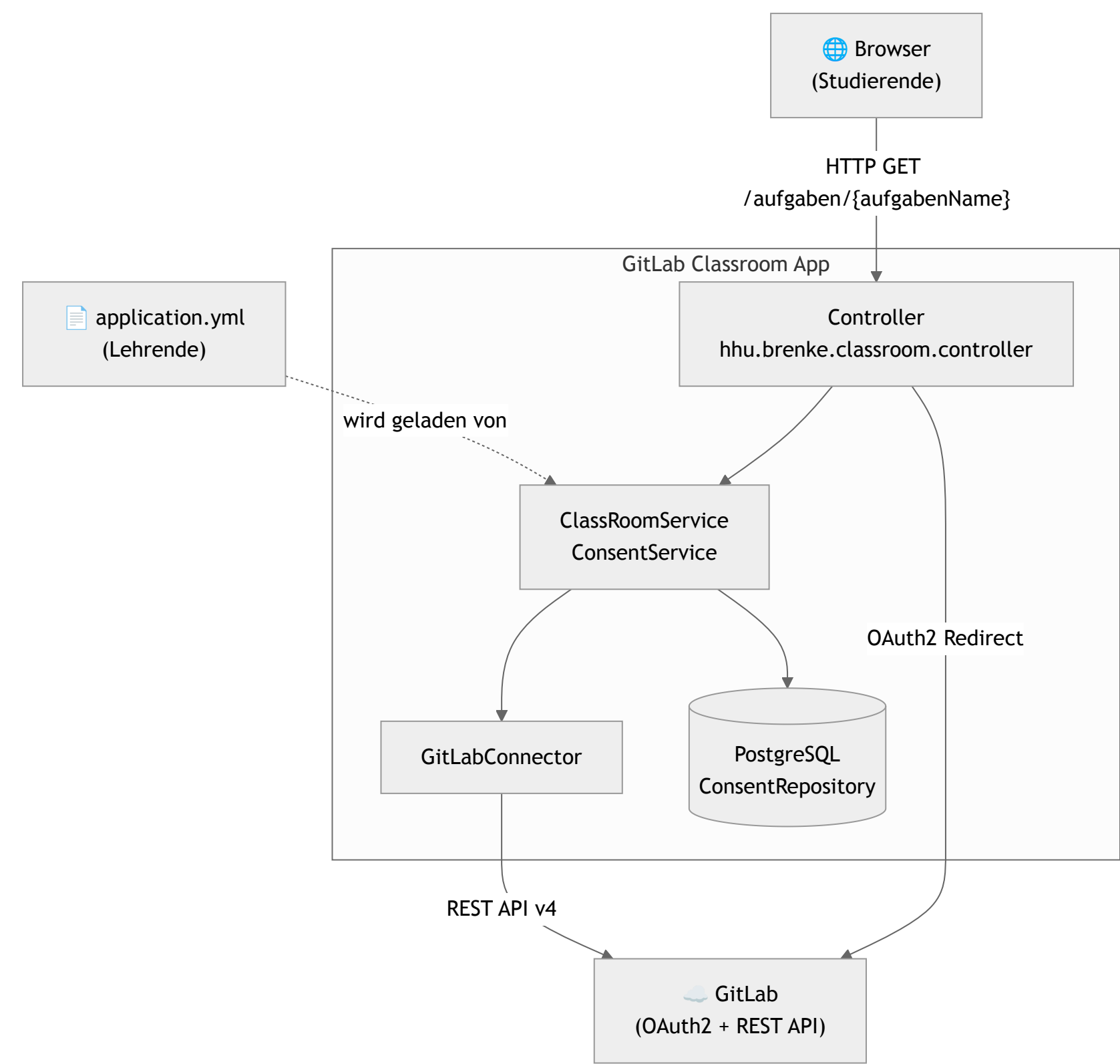


- **Controller** – nimmt HTTP-Requests entgegen, extrahiert OAuth2-User, leitet weiter
- **Service** – Geschäftslogik: Consent-Prüfung, GitLab-API-Calls, Transaktionssteuerung

- **Repository** – Spring Data JPA, CRUD auf Datenbank
- **GitLabConnector** – kapselt alle GitLab-REST-API-Calls (Fork, Rechtevergabe)

5. Bausteinsicht

GitLab Classroom - Bausteinsicht Level 1



Bausteine

Baustein	Was er macht
Controller	Nimmt Requests entgegen, liest den eingeloggtten User aus und gibt die Arbeit an die Services weiter
ClassRoomService	Hauptlogik: prüft die Einwilligung, legt Repos an und ruft die nötigen GitLab-APIs auf
ConsentService	Kümmert sich um die DSGVO-Einwilligung – prüfen und speichern
GitLabConnector	Wrapper für die GitLab REST API
ConsentRepository	Spring Data JPA Interface für den Datenbankzugriff
PostgreSQL	Speichert die Einwilligungen

Schnittstellen

Von	Nach	Art
Browser	Controller	HTTP GET (Spring MVC)
Controller	ClassRoomService	Java-Methodenaufruf
ClassRoomService	GitLabConnector	Java-Methodenaufruf

Von	Nach	Art
ClassRoomService	ConsentRepository	Spring Data JPA
GitLabConnector	GitLab	HTTPS REST (JSON)
Controller	GitLab	OAuth2 Redirect

Konfiguration

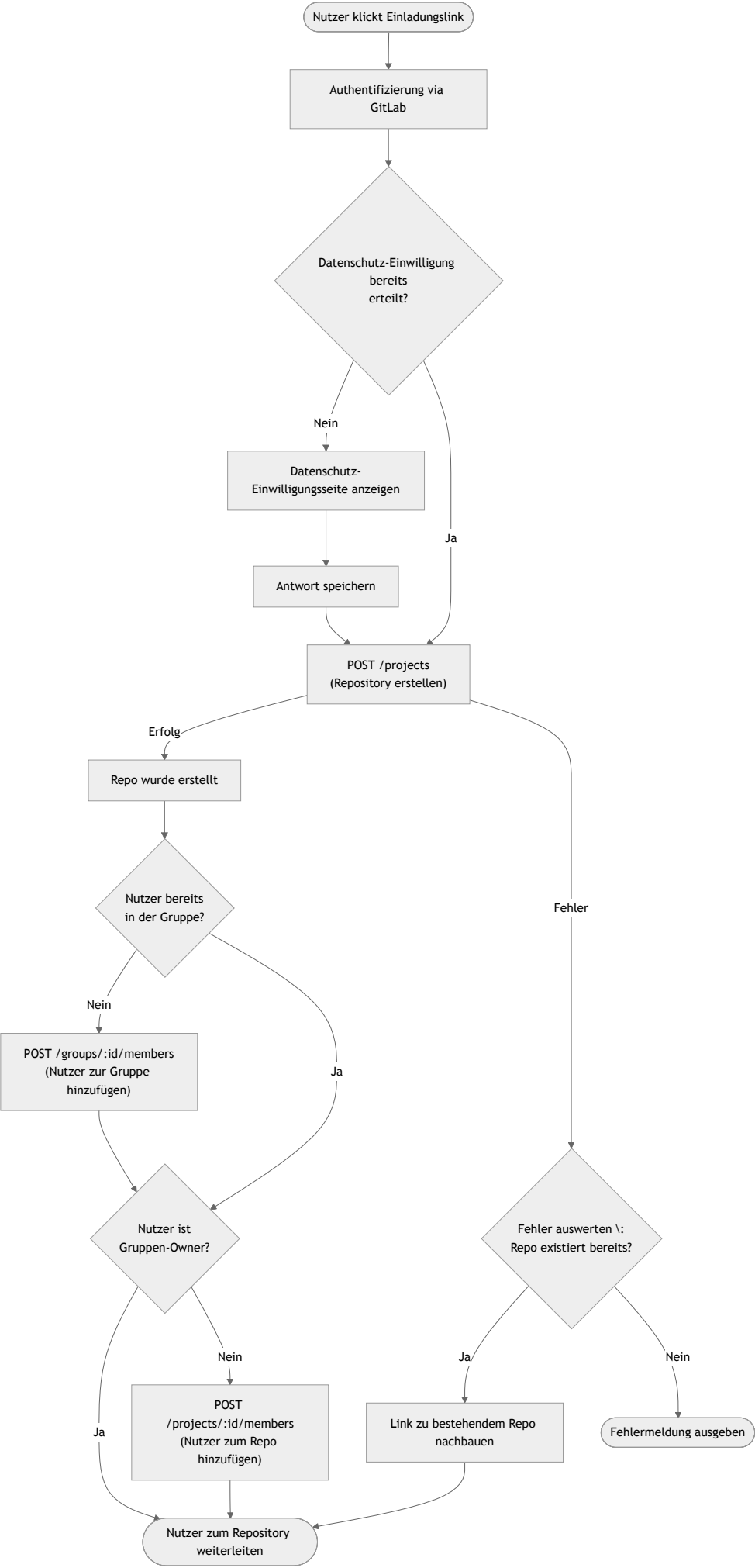
Klasse	Prefix in application.yml	Wichtige Felder
GitLabConfig	gitlab	baseUrl, apiUrl, token, groupId, groupName, owner, studentLevelInGroup, studentLevelInProject
VeranstaltungConfig	veranstaltung	aufgaben (Map: Schlüssel → Template-URL)
SecurityConfig	–	Spring Security FilterChain (OAuth2, CSRF, Session)

Neue Aufgabe hinzufügen – nur application.yml ändern, kein Neudeployment nötig:

```
veranstaltung:
  aufgaben:
    "1": https://git.hhu.de/gruppe/aufgabe-1.git
    "neu": https://git.hhu.de/gruppe/neue-aufgabe.git # ← hier eintragen
```

6. Laufzeitsicht

6.1 Hauptablauf: Aufruf eines Einladungslinks



6.2 Ablaufbeschreibung

Schritt	Beteiligte Komponenten	Was passiert
Authentifizierung via GitLab	SecurityConfig , GitLab OAuth2	Nicht eingeloggte Nutzer werden direkt zu GitLab weitergeleitet. Nach dem Login speichert Spring Security den Nutzernamen und die Gruppenzugehörigkeit in der Session.
Datenschutz- Einwilligung bereits erteilt?	ConsentService , ConsentRepository	Es wird geprüft, ob der Nutzer schon in der consent -Tabelle steht. Falls nicht, kommt er auf die Einwilligungsseite.
API: Repository erstellen	ClassRoomService , GitLabConnector	Die App ruft POST /projects auf GitLab auf und gibt dabei die Import-URL des Template-Repositories mit.
Fehler auswerten: Repo existiert bereits?	ApiResultWrapper	Falls GitLab „has already been taken" zurückgibt, wird die URL des bestehenden Repos aus der Konfiguration und den Nutzerdaten zusammengebaut.
Nutzer bereits in der Gruppe?	ClassRoomService.isUserInGroup	Der groups -Claim im OIDC-Token wird gegen GitLabConfig.groupName geprüft.
API: Nutzer zur Gruppe hinzufügen	GitLabConnector	POST /groups/:id/members mit Access Level Guest.
Nutzer ist Gruppen- Owner?	ClassRoomService.addUserToProject	GitLab lässt es nicht zu, einem Owner eine niedrigere Rolle zuzuweisen. Deshalb wird dieser Schritt für den Owner einfach übersprungen.
API: Nutzer zum Repo hinzufügen	GitLabConnector	POST /projects/:id/members mit Access Level Maintainer.
Nutzer zum Repository weiterleiten	Controller	HTTP 302 zur GitLab-Repo-URL.

7. Verteilungssicht

Docker Compose Setup



Gestartet mit `docker compose up -d`. Die Secrets kommen aus einer `.env`-Datei im selben Verzeichnis.

Lokale Entwicklung

Empfohlene Variante: nur PostgreSQL in Docker laufen lassen, die App direkt in der IDE starten:

```
docker compose up postgres  # nur DB starten
```

In der IDE müssen die Werte aus `.env` als Umgebungsvariablen gesetzt werden. Außerdem muss `SPRING_PROFILES_ACTIVE=local` gesetzt sein, damit die App die `application-local.yml` mit der lokalen DB-URL verwendet.

10. Qualitätsanforderungen

Das eigentliche Ziel war es, GitHub Classroom nachzubauen – die konkreten Qualitätsanforderungen wurden nicht vorab formell definiert, sondern haben sich aus dem Kontext ergeben.

Ziel	Was das konkret bedeutet
Zuverlässigkeit	GitLab API-Aufrufe sind idempotent, ein Fehler führt zu einer verständlichen Meldung, nicht zu einem inkonsistenten Zustand
Benutzerfreundlichkeit	Studierende brauchen keine Git-Kenntnisse um die App nutzen zu können
Wartbarkeit	Neue Aufgabe hinzufügen bedeutet nur eine YAML-Änderung, kein neues Deployment
Sicherheit	Secrets nicht in Git
DSGVO	Einwilligung wird explizit eingeholt und gespeichert

Hinweis zur Datenbank: Die DB ist ein eigenständiges Zusatzfeature für DSGVO-Zwecke. Die Kernfunktionalität – Repository erstellen und Zugriffsrechte vergeben – läuft komplett über die GitLab API und ist unabhängig davon ob die DB erreichbar ist oder nicht.

11. Risiken und technische Schulden

Risiken

Risiko	Wahrscheinlichkeit	Auswirkung	Was tun?
GitLab-API-Änderungen	Mittel	Hoch	Vor Semesterstart manuell testen; genutzte API-Version im README festhalten
GitLab-Ausfall	Niedrig	Sehr hoch	Fehlerseite mit verständlichem Hinweis – mehr ist nicht möglich
Token-Kompromittierung	Niedrig	Sehr hoch	Token nur in <code>.env</code> ; regelmäßig rotieren (alle 6–12 Monate)
Datenverlust durch <code>down -v</code>	Niedrig	Hoch	Regelmäßige <code>pg_dump</code> -Backups; Warnung in README; nie <code>-v</code> verwenden

Technische Schulden

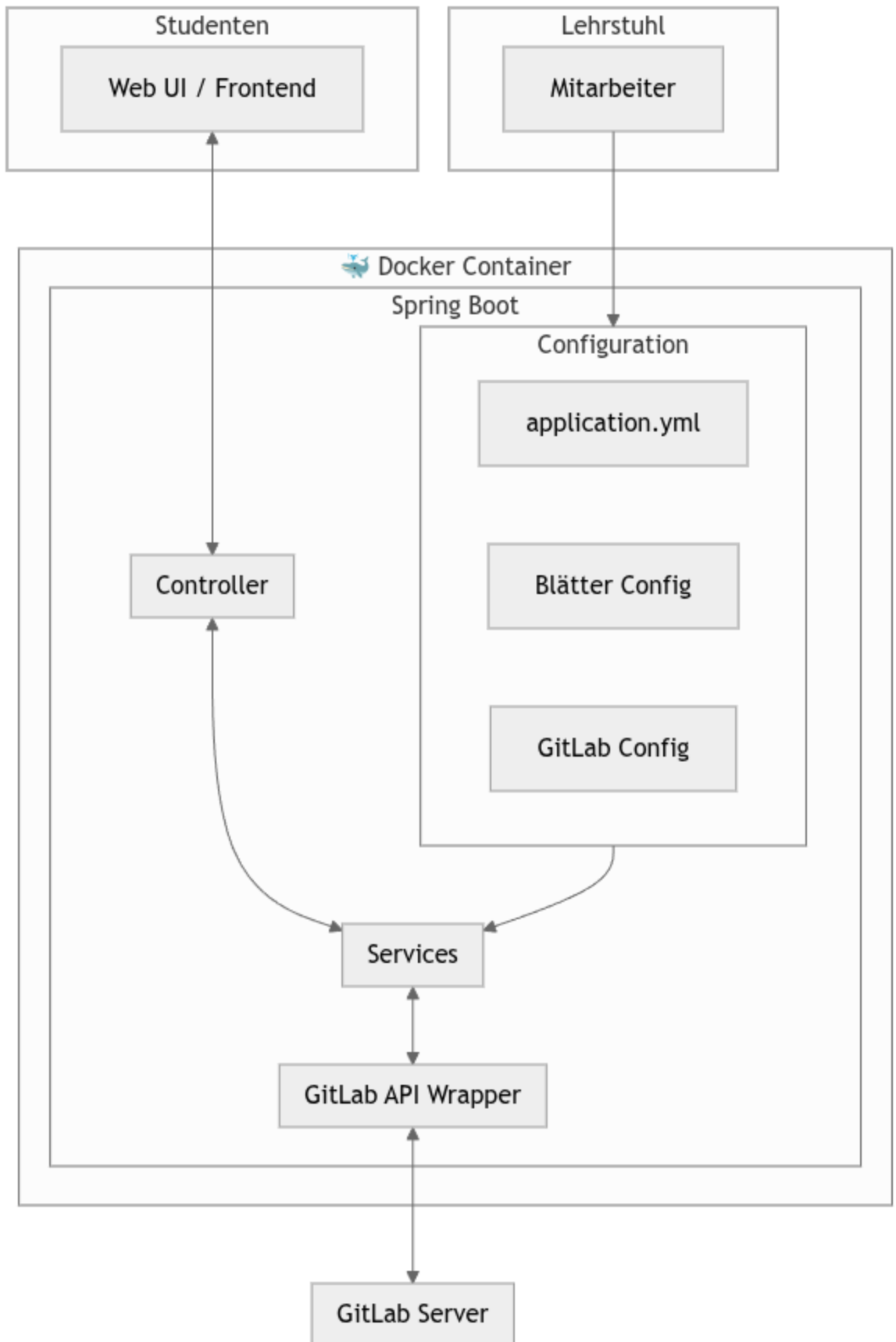
Schuld	Auswirkung	Nächster Schritt
Keine Integrationstests	Zusammenspiel von GitLab-API und App nur manuell testbar	End-to-End-Test für den kompletten Einladungslink-Flow
Manuelle Backups	Backups werden ggf. vergessen	Cron-Job für tägliche <code>pg_dump</code> -Sicherung
Kein Monitoring	Performance-Probleme werden spät erkannt	Spring Boot Actuator aktivieren; für den aktuellen Maßstab reichen Logs
Keine Gruppenaufgaben	Gruppenarbeit muss manuell organisiert werden	Erweiterung möglich, aber außerhalb des aktuellen Scope

Bekannte Limitierungen

- Kein automatisches Repo-Löschen: Repositories werden nur erstellt, nie automatisch entfernt. Das Aufräumen am Semesterende muss manuell in GitLab erfolgen.

12. Glossar

Begriff	Bedeutung
Aufgabe	Eine Programmieraufgabe, die Lehrende für Studierende bereitstellen. Jede Aufgabe hat ein zugehöriges Template-Repository in GitLab.
Aufgaben-Schlüssel	Eindeutiger Kurzbezeichner einer Aufgabe in der Konfiguration (z.B. "1" oder "bonus"). Taucht direkt in der URL auf: /aufgaben/1 .
Einladungslink	URL, die Lehrende an Studierende weitergeben (z.B. http://host:8080/aufgaben/1). Ein Klick startet den kompletten Ablauf: Login → Einwilligung → Repository-Erstellung.
Template-Repository	GitLab-Repository mit Startcode, das als Vorlage für eine Aufgabe dient. Jeder Studierende bekommt eine eigene Kopie davon.
Einwilligung	Zustimmung des Nutzers zur Speicherung seiner Daten (Name, E-Mail, GitLab-ID) gemäß DSGVO. Wird beim ersten Login abgefragt.
Korrektor	Person, die Abgaben bewertet. Wird einmalig manuell als Maintainer zur GitLab-Gruppe hinzugefügt und erbt diese Rolle automatisch in allen Studierenden-Repos.
Veranstaltung	Sammelbegriff für einen Kurs oder ein Modul; in der Konfiguration als veranstaltung.aufgaben abgebildet.



Group & Role structure in GitLab

