Lehrstuhl für Rechnernetze
Prof. Dr. Martin Mauve

Unter der Leitung von Dr. Markus Brenneis

Effiziente Testerstellung und -bewertung mit Java Spring Boot: Eine webbasierte Lösung

---

Bachelorarbeit

Vorgelegt von
Jan Thilo Savary
25. August 2024

Erstgutachter: Dr. Markus Brenneis
Zweitgutachter: Dr. Jens Bendisposto

Abstract

Wissensabfragen sind in vielen Studiengängen ein fester Bestandteil der Lehre. Die Hausaufgaben, die in der Regel wöchentlich abgegeben werden, sind ein Mittel, um die Studierenden auf die Modulprüfungen vorzubereiten. Allerdings kann es bei hohen Teilnehmerzahlen schwierig sein, diese zeitnah zu korrigieren. Eine Variante von solchen Hausaufgaben sind Quizze, die je nach Fragestellung automatisch korrigiert werden können. In dieser Bachelorarbeit wird eine webbasierte Lösung entwickelt, die es ermöglicht, Quizze zu erstellen und durchzuführen.

Danksagung

Zuerst möchte ich mich bei meinem Betreuer, Dr. Markus Brenneis, für die aktive Betreuung und Unterstützung während der gesamten Arbeit bedanken. Die wöchentlichen Meetings haben mir eine klare Struktur gegeben und mir geholfen, mich auf das Wesentliche zu konzentrieren.

Ein weiterer Dank gilt meiner Partnerin Judith Ditsche, die mich während der gesamten Zeit unterstützt hat.

Auch allen Freunden, Bekannten und Kommilitonen, die mich während der Arbeit unterstützt haben, möchte ich danken.

Zuletzt möchte ich mich bei meinem Vater Holger Kaltwasser bedanken, der mir das Studium ermöglicht hat.

Inhaltsverzeichnis

1 Einleitung

In zahlreichen Studiengängen, darunter auch in der Informatik, ist es gängige Praxis, dass sich die Studierenden zunächst durch entsprechende Vorleistungen für die Teilnahme an Modulprüfungen qualifizieren müssen.
Dabei werden häufig wöchentliche Hausaufgaben als Form der Leistungskontrolle eingesetzt. Allerdings kann es bei hohen Teilnehmerzahlen herausfordernd sein, diese zeitnah zu korrigieren. Eine mögliche Lösung stellen automatisierte Tests dar.

Als Lernplattform wird an der Heinrich-Heine-Universität Düsseldorf überwiegend ILIAS verwendet (aktuell die Version v7.30.1 vom 28.05.2024) [ILIAS]. Die E-Learning-Plattform ILIAS bietet die Möglichkeit, Tests mit einer Vielzahl von Fragetypen zu erstellen. Dazu zählen unter anderem Multiple-Choice-Fragen, Lückentexte, numerische Fragen, Text-Teilmengen, Zu- und Anordnungsaufgaben mit Drag-and-Drop-Unterstützung sowie Freitextfragen.
Allerdings erweist sich die manuelle Korrektur von Aufgaben über ILIAS unter Umständen als relativ komplex.
Es ist üblich, dass Studierende mehrere Durchläufe zur Bearbeitung eines Quiz haben. Bei der Korrektur muss darauf geachtet werden, dass der letzte Durchlauf jeder Aufgabe bewertet wird.

In ILIAS stehen zwei Korrekturansichten zur Verfügung: die Korrektur pro Teilnehmer und die Korrektur pro Frage.
Die Korrekturansicht pro Teilnehmer ermöglicht es, für einen Studierenden für jede Frage die letzte Antwort zu korrigieren. Allerdings ist die gleichzeitige Bearbeitung nicht möglich, da nur die gesamte Korrektur gespeichert werden kann und nicht die einzelnen Korrekturen pro Frage. Bei einer großen Teilnehmerzahl ist es üblich, dass Korrektoren aufgabenweise ihre Korrekturen zugewiesen bekommen, wodurch die Korrektur pro Teilnehmer nicht möglich ist.
Die Korrekturansicht pro Frage zeigt alle Antworten einer Frage gruppiert nach Durchlaufnummer. Da sämtliche Durchläufe angezeigt werden, ist eine manuelle Ermittlung des letzten Durchlaufs erforderlich, was ohne die Verwendung von externem Javascript nicht möglich ist.

In der vorliegenden Bachelorarbeit wird eine webbasierte Lösung für die Erstellung und Durchführung von Quizzen entwickelt, welche die oben genannten Probleme löst und sich dabei an den Anforderungen der Heinrich-Heine-Universität Düsseldorf orientiert.

2 Zielsetzung und Randbedingungen

In dieser Bachelorarbeit wird eine webbasierte Software entwickelt, die es ermöglicht, Quizze zu erstellen, durchzuführen und auszuwerten. Die Software soll in Java entwickelt werden und als Framework Spring Boot verwenden.

Folgende Features sollen implementiert werden:

- Administratoren können Quizze durch den Upload von Dateien erstellen. Vor Beginn des Bearbeitungszeitraums ist es möglich, die Quizze zu löschen. Nachträgliche Bearbeitungen von Quizzen sind möglich, wobei existierende Durchläufe automatisch korrigiert werden.
- Studierende können Quizze während des Bearbeitungszeitraums mehrfach bearbeiten. Die maximale Anzahl an Durchläufen wird bei der Erstellung des Quiz festgelegt.
- Nach Abschluss des Bearbeitungszeitraums beginnt die Korrekturphase, in der Korrigierende die Quizze der Studierenden bewerten. Fragen, die automatisch bewertet werden können, werden automatisch bewertet.
- Nach Abschluss der Korrekturphase können die Studierenden ihre Ergebnisse einsehen. Administratoren haben die Möglichkeit, die Ergebnisse der Studierenden einzusehen und zu exportieren.
- Als Fragetypen werden Freitextfragen, Single-Choice-Fragen und Multiple-Choice-Fragen sowie Lückentextfragen unterstützt.

Die Software muss in einer Docker-Umgebung laufen und Daten in einer Datenbank persistieren.

3 Schwierigkeiten und Herausforderungen

Bei der Entwicklung des Projekts sind einige Schwierigkeiten und Herausforderungen aufgetreten. Im Folgenden werden die wichtigsten davon beschrieben.

3.1 JDBC CRUD Repositories: Performanzprobleme bei OneToMany-Beziehungen

Die Anwendung verwendet JDBC CRUD Repositories, um auf die Datenbank zuzugreifen. Dabei ist aufgefallen, dass die Performance stark abnimmt, wenn viele Aggregatobjekte mit OneToMany-Beziehungen abgefragt werden. Dies liegt daran, dass erst alle n Aggregate in einer Query abgefragt werden und dann für jedes einzelne Aggregat eine neue Query ausgeführt wird, um die zugehörigen OneToMany-Objekte zu laden. Somit werden insgesamt $n + 1$ Queries ausgeführt, was zu einer schlechten Performance führt.

Dieser Fall macht sich bei der Abfrage von Quiz Durchläufen bemerkbar, da für ein Quiz sehr viele Quiz Durchläufe existieren können. Für jeden Quiz Durchlauf werden die zugehörigen Antworten abgefragt, was zu einer hohen Anzahl von Queries führt. Bei der Abfrage von Quizzen fällt dies hingegen nicht so stark ins Gewicht, da die Anzahl der Quizze in der Regel deutlich geringer ist.

Performanter wäre es, wenn alle Aggregatobjekte und anschließend alle OneToMany-Objekte in einer zweiten Query abgefragt werden könnten.
Um dies für den oben beschriebenen Fall zu erreichen, wird ein zweites Quiz Durchlauf CRUD Repository implementiert, welches keine OneToMany-Beziehungen enthält und somit nur die Quiz Durchläufe abfragt. Für die Antworten wird ein weiteres CRUD Repository erstellt. Um die Antworten zu den Quiz Durchläufen zu laden, werden die Quiz IDs bestimmt, anschließend die notwendigen Antworten in einer Query abgefragt und dann den Quiz Durchläufen zugeordnet.

Dieses Vorgehen ist jedoch nicht optimal, da es zu einer stärkeren Kopplung der Repositories führt und die Wartbarkeit des Codes erschwert.

Alternativ könnten keine CRUD Repositories verwendet werden, sondern die Queries direkt in den Services implementiert werden. Aber auch dieses Vorgehen koppelt den Code an die Datenbank und erschwert die Wartbarkeit.

Eine weitere Möglichkeit wäre die Verwendung von anderen Technologien. Der Autor ist in

diesem Bereich nicht ausreichend qualifiziert, um eine fundierte Einschätzung vorzunehmen, ob diese Alternativen geeigneter wären.

3.2 Verschiedene Fragentypen: Kopplung im Code

Ein weiteres Problem, das sich während der Entwicklung des Projekts gezeigt hat, ist die Implementierung der verschiedenen Fragentypen. Jeder Fragentyp benötigt unterschiedliche Informationen und somit unterschiedliche Klassen. Diese Klassen erben zwar von einer gemeinsamen Oberklasse, jedoch müssen die spezifischen Informationen in den jeweiligen Klassen implementiert werden.

Dies führt an verschiedenen Stellen in den äußeren Schichten des Projekts zu einer starken Kopplung: Beispielsweise müssen die Klassen unterschiedlich im Browser dargestellt werden. Beim Darstellen einer Frage muss also ermittelt werden, um welchen Fragentyp es sich handelt, um dann abhängig davon die richtige Darstellung zu wählen.

Um das Problem zu lösen, könnten wir die Verantwortlichkeit für die Darstellung in die Klassen selbst verlagern, sodass jede Klasse für die Darstellung ihrer eigenen Informationen verantwortlich ist. Dies würde zwar die Kopplung zwischen den Schichten reduzieren, aber auch das Single-Responsibility-Prinzip verletzen und nicht der gewählten Architektur entsprechen.

4 Fazit

Im Rahmen dieser Arbeit wurde eine webbasierte Anwendung entwickelt, die es ermöglicht, Quizze zu erstellen, durchzuführen und zu bewerten. Dabei wurden die Anforderungen, die in der Zielsetzung definiert wurden, erfüllt. Die Anwendung ist funktionsfähig und kann verwendet werden. Sie ist erweiterbar und wartbar, mit den Einschränkungen, die in den Schwierigkeiten beschrieben wurden. Die Anwendung ist getestet und dokumentiert.

Die Performance der Anwendung ist nicht optimal, aber akzeptabel. Dabei ist insbesondere die Abfrage von Quiz Durchläufen mit vielen Antworten problematisch. Die Anwendung unterstützt die geforderten Fragentypen, jedoch könnten noch weitere hinzugefügt werden.

Mögliche Verbesserungen wären die automatische Zuordnung der Antworten zu Korrektoren, sowie die Möglichkeit, Fragen als Pool zu definieren, aus dem nur eine Frage ausgewählt wird.

Literatur

[ILIAS] *Internet Archive: ILIAS der HHU*. https://web.archive.org/web/20240617113745/https://ilias.hhu.de/ilias.php?baseClass=ilrepositorygui&reloadpublic=1&cmd=frameset&ref_id=1 (Abruf am 24.06.2024; Snapshot vom 17.06.2024).

Eidesstattliche Erklärung

Hiermit erkläre ich, Jan Thilo Savary, an Eides statt, dass ich die vorliegende Bachelorarbeit mit dem Titel „Effiziente Testerstellung und -bewertung mit Java Spring Boot: Eine webbasierte Lösung“ selbstständig verfasst und ohne Nutzung anderer als der angegebenen Hilfsmittel und Quellen angefertigt habe. Ich versichere, dass ich die wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Außerdem versichere ich, dass diese Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt wurde. Mir ist bekannt, dass im Falle einer Täuschung die Abschlussarbeit mit „nicht bestanden“ bewertet wird.

Ort, Datum

Unterschrift

Erklärung zur Nutzung generativer KI

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

- ChatGPT zur sprachlichen Überarbeitung von Texten.
- Copilot während der Arbeit am Code als integriertes Werkzeug in der IDE.
- DeepL Write zur sprachlichen Überarbeitung von Texten.

Studi Quiz

Inhaltsverzeichnis

Chapter 1. Einführung und Ziele

1.1. Aufgabenstellung

Studi Quiz ist eine an der HHU im Rahmen einer Bachelorarbeit entwickelte Webanwendung, die es Lehrenden ermöglicht Quizze für ihre Studierenden (oder anderen Zielgruppen) zu erstellen und durchzuführen. Die Antworten werden je nach Aufgabentyp entweder automatisch oder manuell ausgewertet. Für die manuelle Auswertung können Korrektoren die Antworten der Studierenden einsehen und bewerten.

Wesentliche Features:

- Erstellung von Quizzen mit verschiedenen Aufgabentypen über TOML Dateien
- Durchführung von Quizzen
- Mehrere Durchläufe eines Nutzers pro Quiz möglich
- Automatische Auswertung von:
- Single Choice Aufgaben
- Manuelle Auswertung von:
- Freitext Aufgaben
- Export der Ergebnisse oder der Antworten als CSV

1.2. Qualitätsziele

Qualitätsziel	Motivation und Erläuterung
Nutzerfreundlichkeit	Die Anwendung soll einfach und intuitiv zu bedienen sein und sich auf das Wesentliche konzentrieren.
Performance	Die Anwendung soll auch bei vielen Nutzern schnell und zuverlässig reagieren. Als Ziel werden 15 Quizze und 1000 Nutzer angenommen.
Erweiterbarkeit	Änderungen und Erweiterungen sollen einfach und schnell möglich sein. So sollen z.B. neue Aufgabentypen einfach hinzugefügt werden können.

1.3. Stakeholder

Wer?	Interesse, Bezug
Lehrende	• Erstellung von Quizzen • Durchführung von Quizzen • Auswertung von Quizzen

Wer?	Interesse, Bezug
Studierende	• Durchführung von Quizzen • Einsehen der Ergebnisse • Einsehen der eigenen Antworten
Entwickler	• Weiterentwicklung der Anwendung • Weitere Fragentypen hinzufügen
Betreiber	• Einfache Einrichtung der Anwendung

Chapter 2. Randbedingungen

2.1. Technische Randbedingungen

Randbedingung	Erläuterungen, Hintergrund
Webanwendung	Die Anwendung soll als Webanwendung realisiert werden, um eine einfache Nutzung zu ermöglichen.
Implementierung in Java	Die Anwendung soll in Java implementiert werden. Java wird an der HHU in der Lehre verwendet wodurch nachfolgende Entwickler bereits Erfahrung mit der Sprache haben.
Verwendung von Spring Boot	Spring Boot bietet eine einfache Möglichkeit eine Webanwendung zu erstellen und zu betreiben. Wie Java wird Spring Boot an der HHU in der Lehre verwendet und ist den Entwicklern bekannt.
Onion Architektur	Die Anwendung soll nach dem Onion Architekturprinzip aufgebaut werden. Diese Architektur ermöglicht einfache Erweiterungen und Änderungen. Zusätzlich wird sie in der Lehre an der HHU verwendet.
Datenbank	Die Anwendung soll eine Datenbank zur Persistierung der Daten verwenden.
Docker	Die Anwendung soll in einem Docker Container laufen.
Logging	Die Anwendung soll Zugriffe loggen.

2.2. Organisatorische Randbedingungen

Randbedingung	Erläuterungen, Hintergrund
Team	Jan Thilo Savary, unter der Betreuung von Dr. Markus Brenneis, Janine Golov und Dr. Jens Bendisposto.
Zeitplan	Die Entwicklung der Anwendung erfolgt im Rahmen einer Bachelorarbeit. Der Zeitplan ist daher durch die Vorgaben der Hochschule und des Prüfungsamtes vorgegeben. Insgesamt stehen 3 Monate zur Verfügung.
Entwicklungswerkzeuge	Die Entwicklung erfolgt in IntelliJ IDEA. Gradle wird als Build-Tool verwendet. Die Software muss allein mit Gradle, also ohne IDE baubar sein.
Konfigurations- und Versionsverwaltung	Die Anwendung wird in einem Git-Repository verwaltet.
Testwerkzeuge und -prozesse	Die Anwendung wird mit JUnit getestet. Die Tests sollen automatisiert durchgeführt werden.

2.3. Konventionen

Konvention	Erläuterungen, Hintergrund
Architekturdokumentation	Terminologie und Gliederung nach dem deutschen arc42-Template in der Version 6.0
Code-Konventionen	Die Anwendung soll den Google Java Style Guide einhalten. Geprüft wird dies durch Checkstyle.
Sprache	Die Software wird in englischer Sprache entwickelt, wobei Fachbegriffe in Deutsch verwendet werden. Testbeschreibungen sind in Deutsch, Kommentare in Englisch. Die Dokumentation ist in Deutsch.

Chapter 3. Kontextabgrenzung

3.1. Fachlicher Kontext

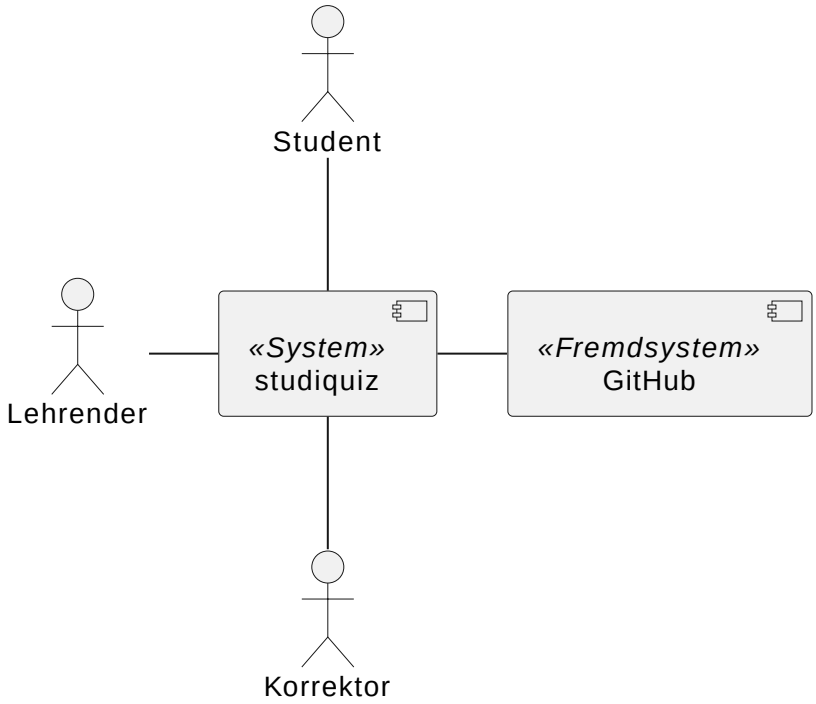

Abbildung 1. Fachlicher Kontext

GitHub (Fremdsystem):
Die Authentifizierung der Nutzer erfolgt über GitHub OAuth. Nutzer müssen sich über GitHub anmelden. Beim Login werden die GitHub ID und der GitHub Name des Nutzers gespeichert. Die Zuordnung erfolgt über die GitHub ID. Rollen werden in der Applikation verwaltet. **Studierende (Benutzer)**:

Studierende starten Quiz Durchläufe und beantworten die Fragen. Sie können die Ergebnisse einsehen.

Korrektoren (Benutzer):
Korrektoren können die Antworten der Studierenden einsehen und bewerten.

Lehrende (Benutzer):
Lehrende erstellen Quizze. Sie können die Antworten als auch die Ergebnisse exportieren. Lehrende können auch die Antworten der Studierenden bewerten. Im Gegensatz zu Korrektoren können Lehrende dies auch nach Abschluss der Korrekturphase.

3.2. Technischer Kontext

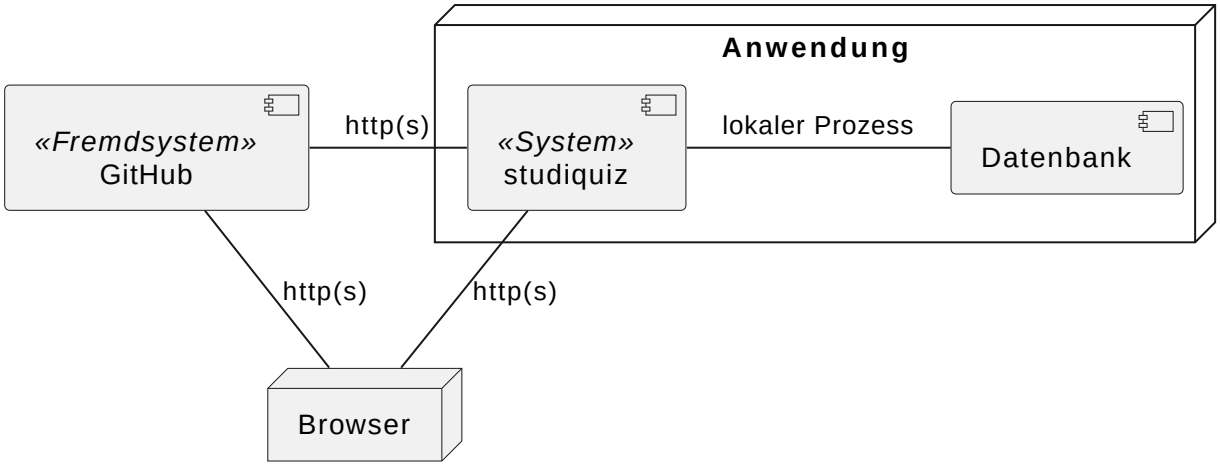

Abbildung 2. Technischer Kontext

Browser (Fremdsystem): Die Anwendung wird über einen Browser aufgerufen.

Datenbank (Fremdsystem): Die Anwendung speichert die Daten in einer Datenbank.

Chapter 4. Lösungsstrategie

Studi Quiz wird als Webanwendung realisiert. Die Anwendung wird entsprechend den technischen Randbedingungen in Java mit Spring Boot entwickelt. Die Anwendung wird nach dem Onion Architekturprinzip aufgebaut.

Der Core besteht aus den Domänen Objekten und Service. Wichtige Domänen Objekte sind dabei "Quiz", "Quiz Durchlauf", "Frage" sowie "Antwort". Weitere wichtige Komponenten sind

- die Controller, welche die Anfragen entgegennehmen und an die Services weiterleiten,
- die Anwendungs Services, welche die Geschäftslogik enthalten
- die Persistenzschicht, welche die Daten in einer Datenbank speichert
  - als Datenbank wird eine PostgreSQL Datenbank verwendet
  - zur Kommunikation mit der Datenbank wird Spring Data JDBC verwendet

Zusätzliche Adapter sind

- der TOML Parser, welcher verwendet wird, um neue Quizze zu importieren
- der SVG Generator, welcher verwendet wird, um die Ergebnisse der Quizze als SVG zu exportieren

Die Verwendung von Spring Boot erlaubt es, einfach eine Webanwendung zu erstellen. Zusätzlich verbindet Spring Boot die einzelnen Komponenten der Anwendung per Dependency Injection. Es soll nur Konstruktor Injection verwendet werden. Dadurch wird die Anwendung testbarer und die Abhängigkeiten sind klarer.

Spring Boot Programme können als Jar-Datei gebaut werden. Diese Jar-Datei kann dann in einem Docker Container laufen.

Die Web Ansichten werden mit Thymeleaf erstellt. Thymeleaf ist ein Template Engine für Webanwendungen. Jede Ansicht wird als HTML-Datei erstellt. Thymeleaf erlaubt es, Variablen in den HTML-Dateien zu verwenden.

Chapter 5. Bausteinsicht

5.1. Whitebox Gesamtsystem

Studi Quiz besteht aus den unten dargestellten Packeten. Die durchgezogenen Pfeile stellen die Abhängigkeiten zwischen den Packeten dar. (x → y für x abhängig von y). Die gestrichelten Pfeile stellen Implementierungen von Interfaces dar. (x → y für x implementiert Interface aus y).

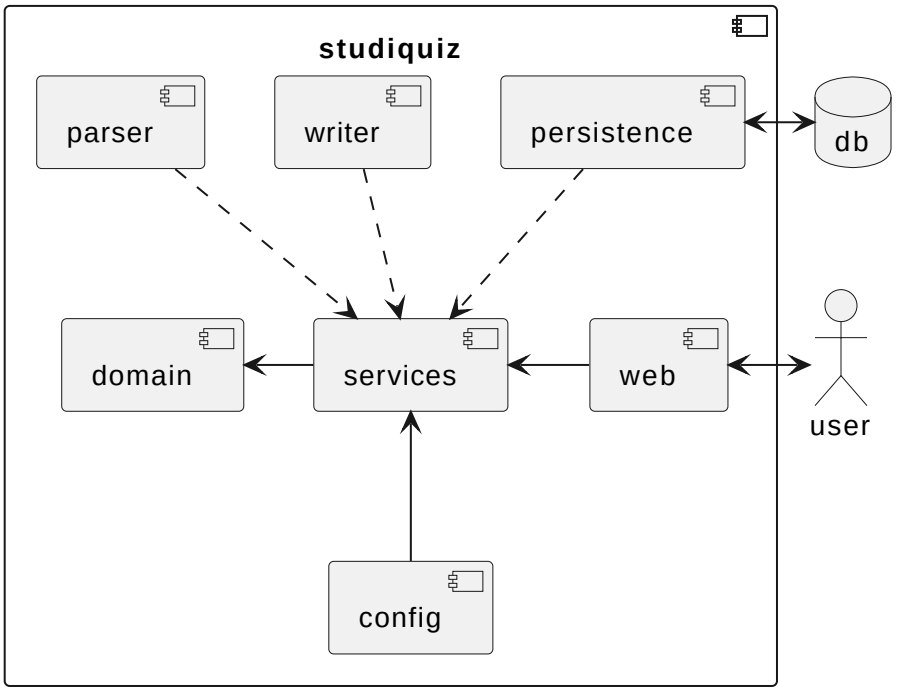

Abbildung 3. Studi Quiz, Bausteinsicht, Ebene 1

Von diesen Packeten ist nur das web Packet für den Nutzer sichtbar. Die anderen Packete sind intern.

Enthaltene Bausteine

Baustein	Beschreibung
web	Enthält die Web Kontroller um eingehende Web Anfragen zu verarbeiten.
domain	Enthält Domain Objekte und Services.
services	Anwendungsservices, die von der Web-Schicht aufgerufen werden.
persistence	Adapter, verantwortlich für die Persistierung von Domain Objekten.
writer	Adapter zum Exportieren von Daten.
parser	Adapter zum Importieren von Daten.
config	Konfiguration von Spring Boot.

Wichtige Schnittstellen

Schnittstelle	Beschreibung
web — Browser	Kommunikation mit dem Nutzer, welcher über den Browser auf die Anwendung zugreift.
persistence — Datenbank	Kommunikation mit der Datenbank.

5.1.1. web (Blackbox)

Das web Packet ist für die Verarbeitung von eingehenden Web Anfragen zuständig. Es enthält Web Kontroller für Studierende, Korrektoren und Admins (Lehrende).

Wichtige Schnittstellen

Schnittstelle	Beschreibung
HTTP /student/*	Die Schnittstelle für Studierende.
HTTP /korrektor/*	Die Schnittstelle für Korrektoren.
HTTP /admin/*	Die Schnittstelle für Admins.

Mehr Informationen über die Endpunkte können in der OpenAPI Spezifikation gefunden werden.

5.1.2. domain (Blackbox)

Enthält die Domain Objekte und Services.

Wichtige Schnittstellen

Schnittstellen	Typ	Beschreibung
Quiz	Aggregate Root	Ein Quiz enthält notwendige Informationen für die Durchführung eines Quiz, sowie die Fragen.
Frage	Entity	Eine Frage enthält die Fragestellung, die Antwortmöglichkeiten und die richtige Antwort.
Quiz Durchlauf	Aggregate Root	Ein Durchlauf enthält die Antworten eines Studierenden auf ein Quiz.
Antwort	Entity	Eine Antwort enthält die Antwort eines Studierenden auf eine Frage.
Bewertung	Entity	Eine Bewertung enthält die Bewertung einer Frage in einem Durchlauf.

5.1.3. services (Blackbox)

Verantwortlich für das Laden von Domain Aggregaten, die Modifikation von Domain Aggregaten, das Speichern von Domain Aggregaten sowie die Ausführung von Anwendungsfällen.

Wichtige Schnittstellen

Schnittstelle	Beschreibung
QuizService	Enthält Methoden um Quizze zu laden, speichern und zu modifizieren.
QuizRepository	Interface, enthält Methoden um Quizze zu laden, speichern und zu modifizieren. Muss von Adaptern implementiert werden.
QuizDurchlaufService	Enthält Methoden um Quiz Durchläufe zu laden, speichern und zu modifizieren.
QuizDurchlaufRepository	Interface, enthält Methoden um Quiz Durchläufe zu laden, speichern und zu modifizieren. Muss von Adaptern implementiert werden.
QuizIoService	Enthält Methoden um Quizze zu importieren und exportieren.
QuizFileParser	Interface, enthält Methoden um Quizze zu importieren. Muss von Adaptern implementiert werden.
QuizFileWriter	Interface, enthält Methoden um Quizze zu exportieren. Muss von Adaptern implementiert werden.
UserService	Enthält Methoden um Nutzer zu laden, speichern und zu modifizieren.
UserRepository	Interface, enthält Methoden um Nutzer zu laden, speichern und zu modifizieren. Muss von Adaptern implementiert werden.

5.1.4. persistence (Blackbox)

Adapter, verantwortlich für die Persistierung von Domain Objekten. Verbindet die Anwendung mit der Datenbank.

Wichtige Schnittstellen

Schnittstelle	Beschreibung
QuizRepositoryImpl	Implementiert QuizRepository.
QuizDurchlaufRepositoryImpl	Implementiert QuizDurchlaufRepository.
UserRepositoryImpl	Implementiert UserRepository.

5.1.5. writer (Blackbox)

Adapter zum Exportieren von Daten.

Wichtige Schnittstellen

Schnittstelle	Beschreibung
QuizCsvFileWriter	Implementiert QuizFileWriter, exportiert Quizze in CSV Dateien.

5.1.6. parser (Blackbox)

Adapter zum Importieren von Daten.

Wichtige Schnittstellen

Schnittstelle	Beschreibung
QuizTomlFileParser	Implementiert QuizFileParser, importiert Quizze aus TOML Dateien.

5.1.7. config (Blackbox)

Konfiguration von Spring Boot.

Wichtige Schnittstellen

Schnittstelle	Beschreibung
SecurityConfig	Konfiguriert die Authentifizierung und Autorisierung von Nutzern für Spring Boot.
LoggableDispatcherServlet	Konfiguriert das Logging von eingehenden Web Anfragen.

5.2. Ebene 2

5.2.1. web (Whitebox)

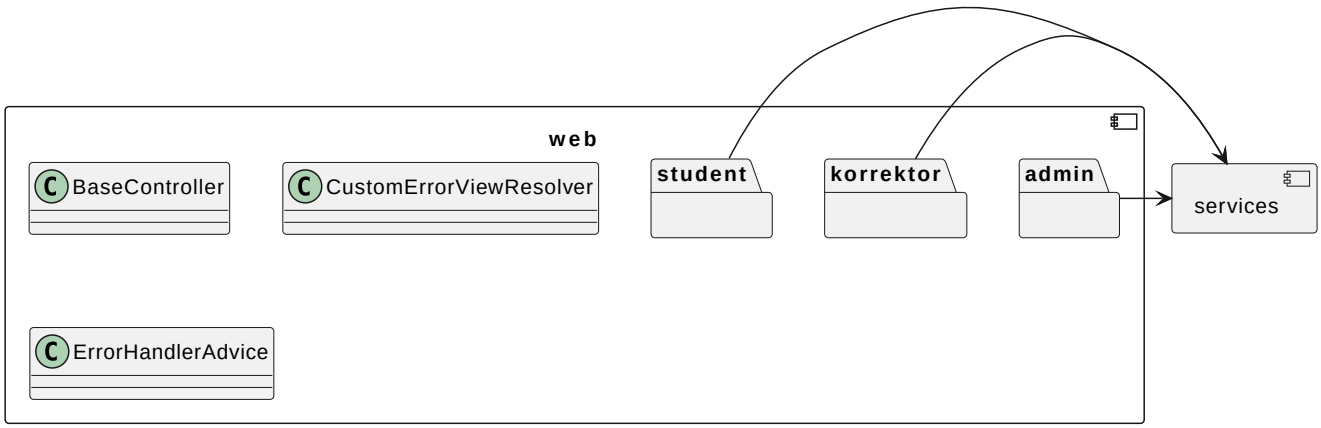

Abbildung 4. Studi Quiz, Bausteinsicht, Ebene 2: Web

Enthaltene Bausteine

Baustein	Beschreibung
BaseController	Enthält einfache Weiterleitungen je nach Nutzerrolle.

Baustein	Beschreibung
CustomErrorViewResolver	Enthält Methoden um Fehlermeldungen zu verarbeiten.
ErrorHandlerAdvice	Enthält Methoden um Exceptions zu verarbeiten.
admin	Enthält Web Kontroller für Admins.
korrektor	Enthält Web Kontroller für Korrektoren.
student	Enthält Web Kontroller für Studierende.

5.2.2. services (Whitebox)

5.2.3. domain (Whitebox)

5.3. Ebene 3

5.3.1. web.admin (Whitebox)

5.3.2. web.korrektor (Whitebox)

5.3.3. web.student (Whitebox)

5.3.4. domain.models (Whitebox)

domain.models.quiz (Blackbox)

domain.models.quizDuarchlauf (Blackbox)

Chapter 6. Laufzeitsicht

Im Folgenden wird die Laufzeitsicht des Systems beschrieben. Diese Sicht beschreibt die Laufzeitstruktur des Systems und zeigt, wie die Bausteine zur Laufzeit zusammenarbeiten.

Dazu wurden einige interessante Laufzeitszenarien identifiziert, die im Folgenden beschrieben werden.

6.1. Student lädt Quiz Detailansicht

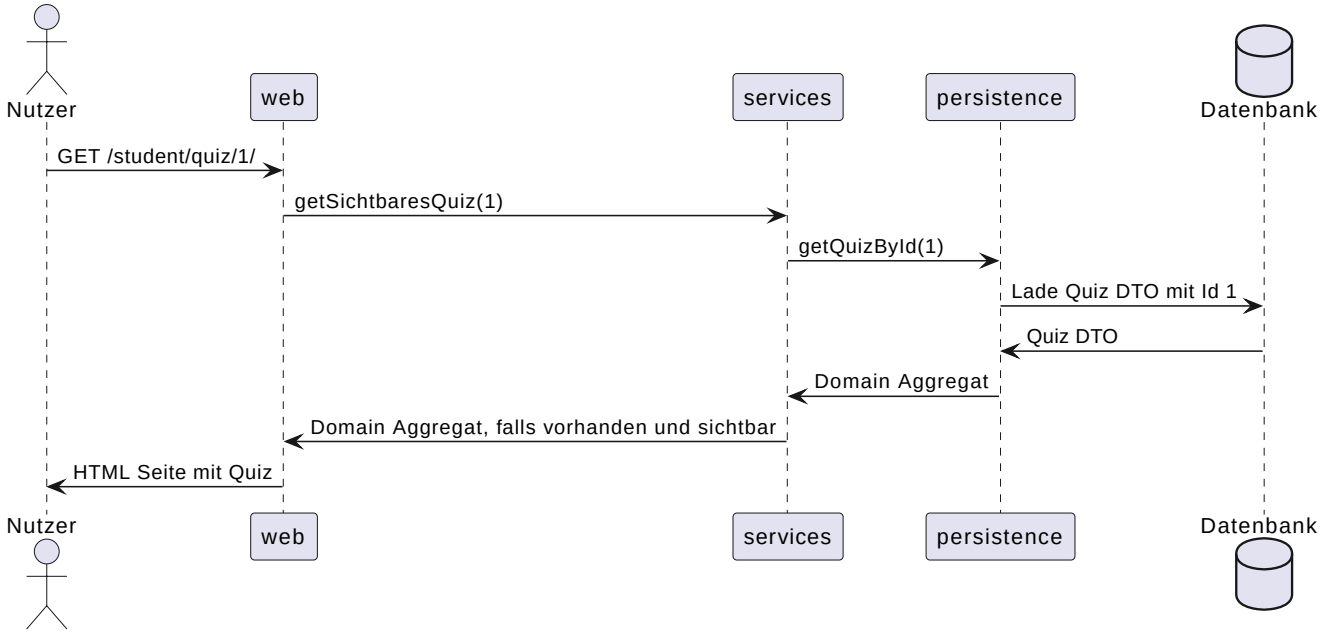

Abbildung 5. Studi Quiz, Laufzeitsicht, Student lädt Quiz Detailansicht

Die meisten Nutzerinteraktionen bestehen daraus, dass ein (oder mehrere) Domain Aggregate geladen werden und die zugehörigen Daten angezeigt werden. Dieses Szenario zeigt beispielsweise, was passiert, wenn ein Nutzer die Detailansicht eines Quiz lädt.

6.2. Student startet neuen Quizdurchlauf

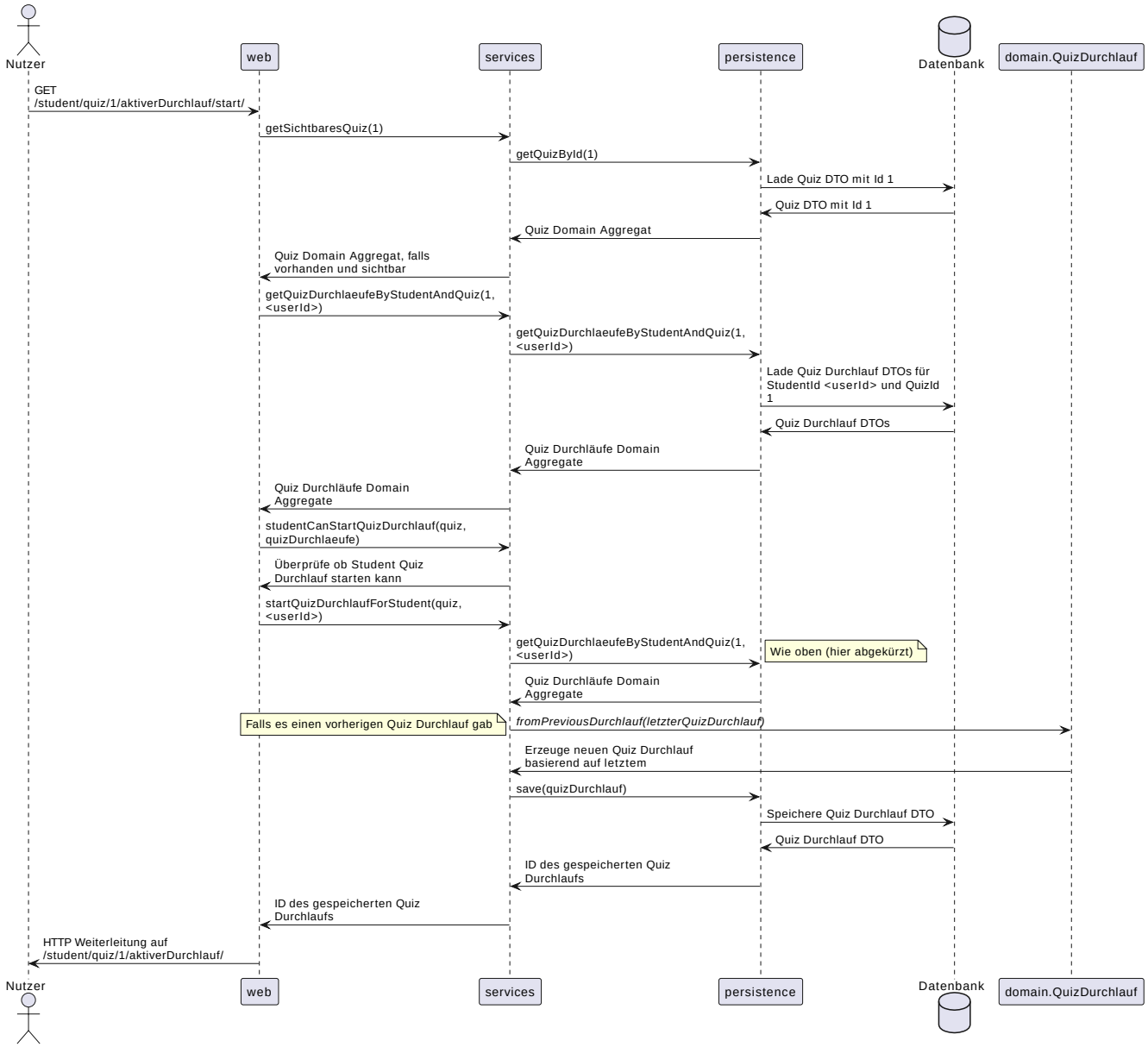

Abbildung 6. Studi Quiz, Laufzeitsicht, Student startet neuen Quizdurchlauf

Beim Starten eines neuen Quiz Durchlaufs wird (nach laden der erforderlichen Aggregaten) überprüft, ob der Nutzer überhaupt berechtigt ist, ein Quiz Druchlauf zu starten. Anschließend wird ein neuer Quizdurchlauf erstellt. Falls ein vorheriger Quizdurchlauf existiert, werden die Antworten des vorherigen Quiz Durchlaufs kopiert (das schließt bereits existierende Bewertungen mit ein), andernfalls wird ein Quiz Durchlauf ohne Antworten erstellt. Dieser Quizdurchlauf wird dann gespeichert und der Nutzer wird zur ersten Frage weitergeleitet.

6.3. Student beantwortet Frage

Für dieses Szenario werden zwei Vorgänge betrachtet: Das Laden/Anzeigen der Frage und das Speichern der Antwort.

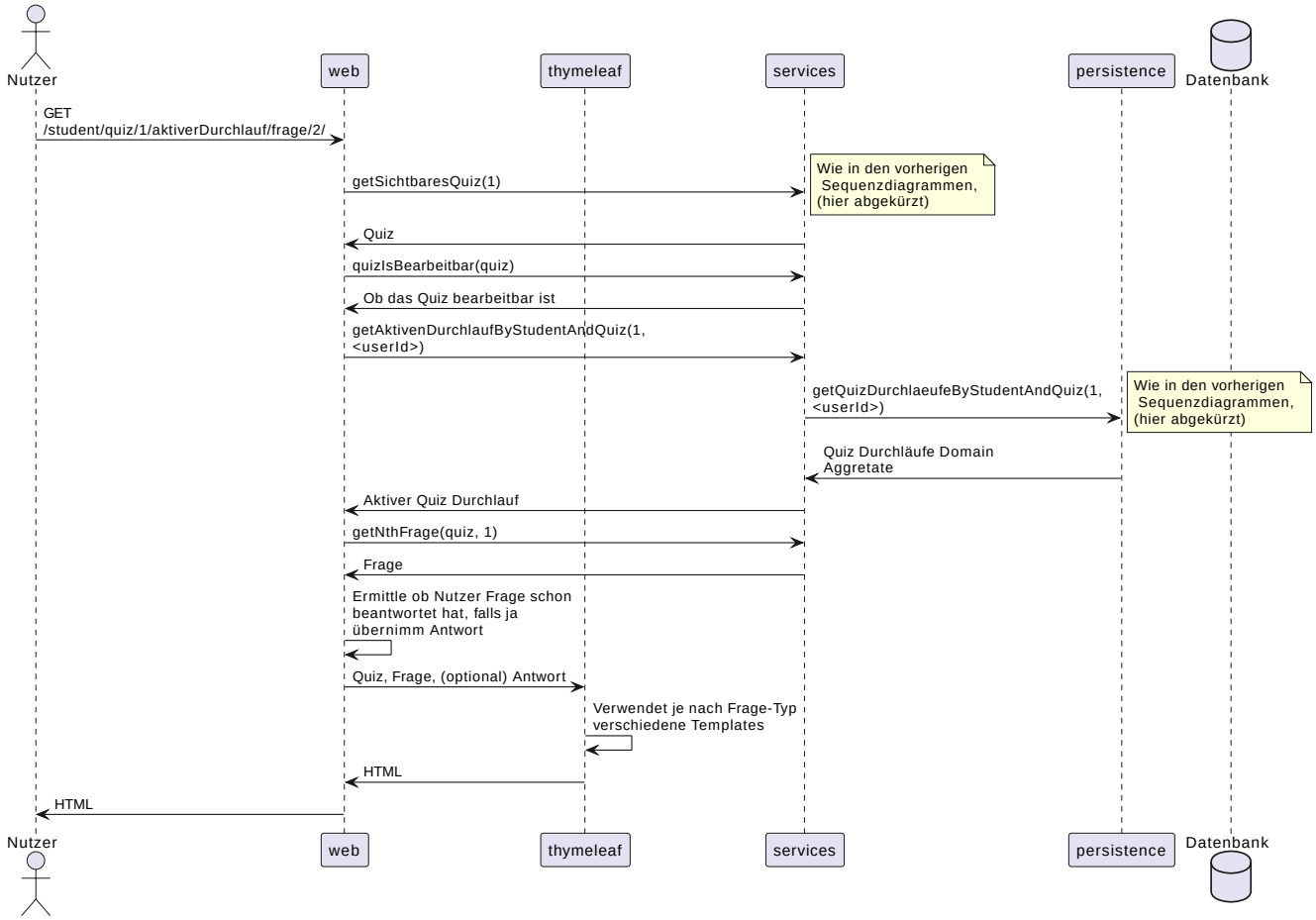

Abbildung 7. Studi Quiz, Laufzeitsicht, Student lädt Frage

Beim Laden einer Frage wird das zugehörige Quiz geladen und die Frage wird angezeigt. Wichtig hierbei ist, dass im Thymleaf Template entschieden wird, welches Fragment geladen wird, basierend auf dem Typ der Frage.

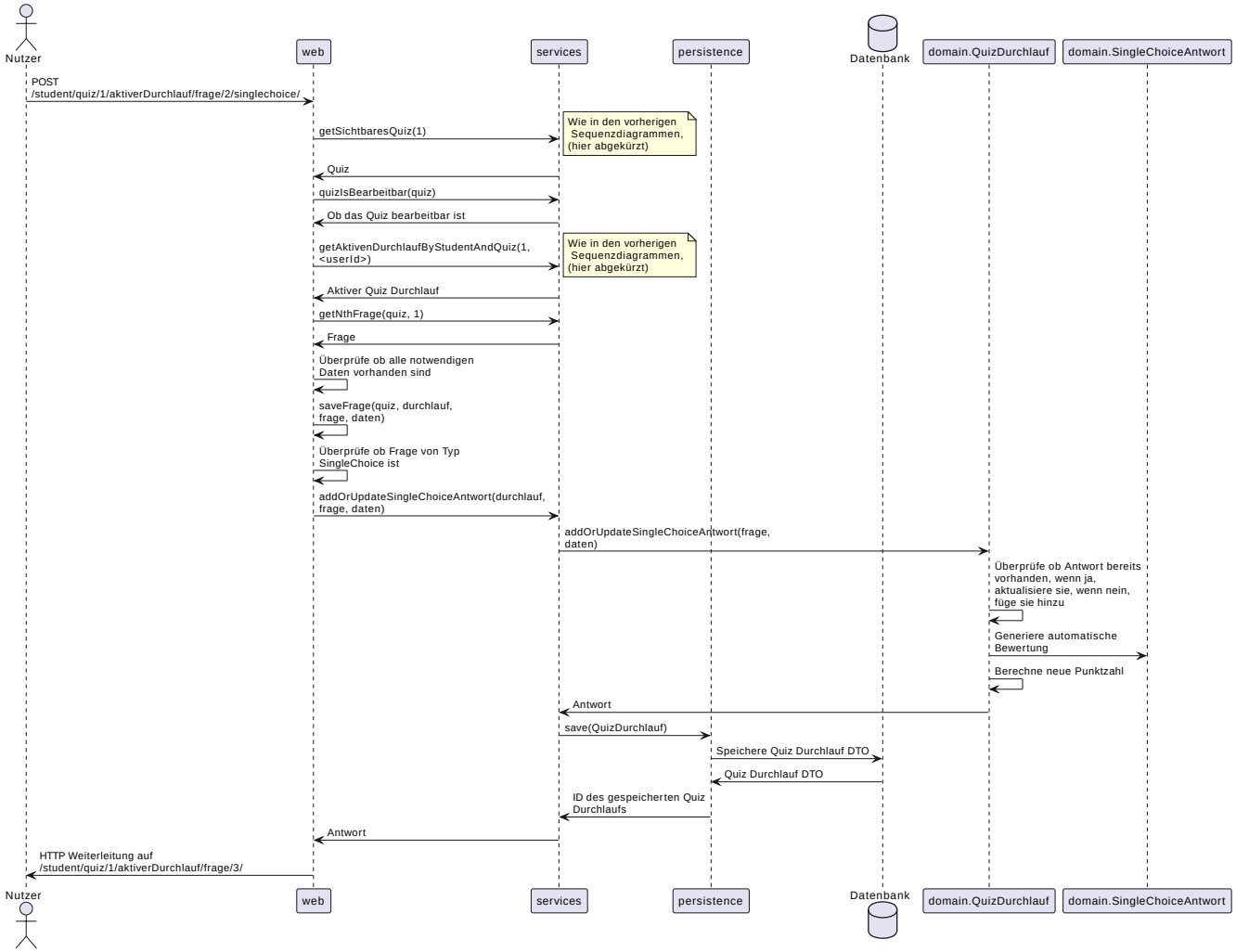

Abbildung 8. Studi Quiz, Laufzeitsicht, Student speichert Antwort

Beim Speichern einer Antwort wird das zugehörige Quiz sowie der zugehörige Quiz Durchlauf geladen. Anschließend wird die Antwort gespeichert, also entweder eine neue Antwort erstellt oder eine bereits existierende Antwort aktualisiert. Danach wird eine automatische Bewertung generiert. Für Fragen, die eine manuelle Bewertung erfordern, passiert dabei nichts. Anschließend wird der Quiz Durchlauf gespeichert und der Nutzer wird zur nächsten Frage weitergeleitet.

Chapter 7. Verteilungssicht

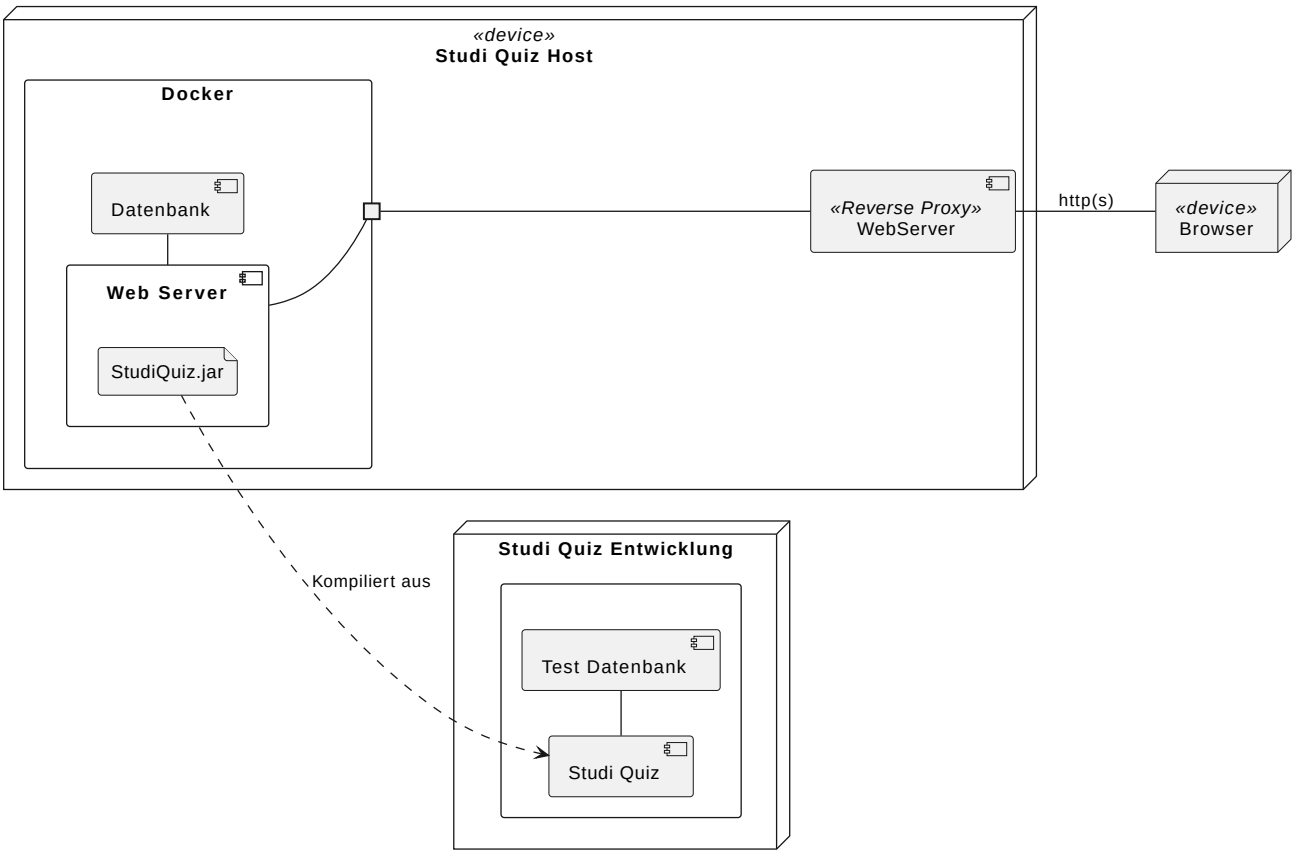

Abbildung 9. Studi Quiz, Verteilungssicht

Komponente	**Beschreibung**
Studi Quiz Develpment	Entwicklungsumgebung für die Studi Quiz Anwendung, Standard System mit JDK 21, Gradle installiert.
Test Datenbank	Datenbank in der Entwicklungsumgebung, PostgreSQL
Studi Quiz Host	Host für die Studi Quiz Anwendung, System, dass Docker sowie einen Webserver (beispielsweise Apache oder Nginx) installiert hat.
Docker	Docker Umgebung auf dem Host, um die Studi Quiz Anwendung und die Datenbank in Containern zu betreiben. Gibt nach außen den Port der Studi Quiz Anwendung frei.
StudiQuiz.jar	Eine "fat jar" Datei, die alle Abhängigkeiten enthält und die Studi Quiz Anwendung startet. Läuft abgekappselt in einem Docker Container.
Datenbank	Datenbank in der Produktivumgebung, PostgreSQL. Läuft abgekapselt in einem Docker Container.
WebServer	Web Server auf dem Host, um die Studi Quiz Anwendung nach außen zu veröffentlichen. Fungiert als Reverse Proxy.
Browser	Browser über den die Nutzer auf die Studi Quiz Anwendung zugreifen. Kommuniziert über das Internet mit dem Web Server.

Chapter 8. Querschnittliche Konzepte

8.1. Domain Modelle

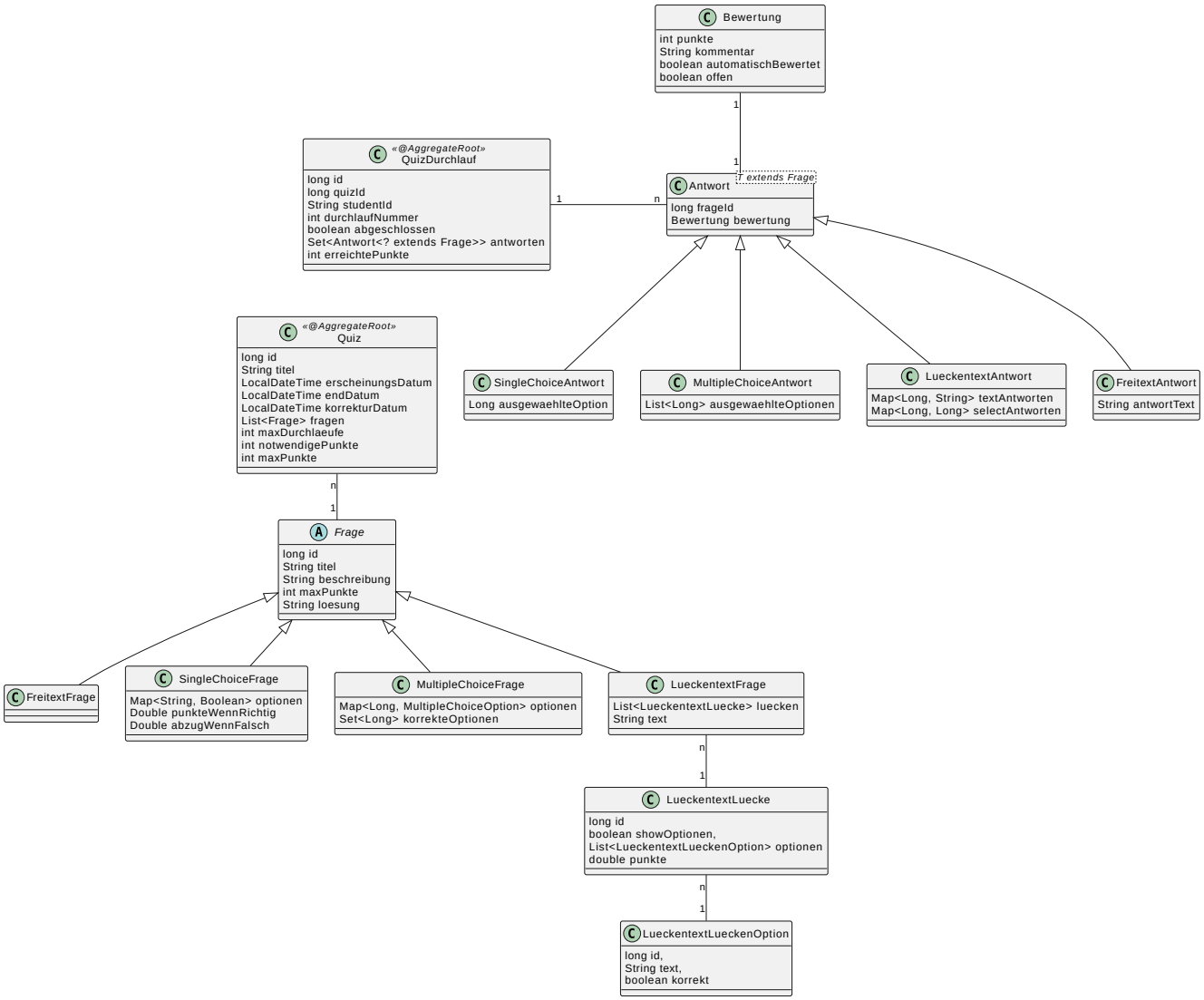

Abbildung 10. Studi Quiz, Domain Modelle

8.2. Persistenz

Zur Persistierung von Daten wird eine PostgreSQL Datenbank verwendet.

Für alle Domain Modelle existieren entsprechende Tabellen in der Datenbank. Die Datenbank wird über Spring Data JDBC angesprochen. Dazu werden für jedes Domain Modell entsprechende Repositories erstellt, die die CRUD-Operationen bereitstellen.

Die Datenbankverbindung in der Konfiguration definiert. Dabei müssen die Datenbankurl, der Datenbankname, der Benutzername und das Passwort angegeben werden. Als Port wird der Standardport 5432 verwendet.

8.2.1. Migrationen

Die Datenbankstruktur wird über Flyway verwaltet. Die Migrationen sind im Ordner `src/main/resources/db/migration` abgelegt. Die Migrationen werden automatisch beim Starten der Anwendung durch Flyway ausgeführt.

8.3. HTML Generierung sowie CSS und JavaScript

Die HTML Generierung erfolgt über Thymeleaf. Dabei werden die HTML Templates in den Ordnern `src/main/resources/templates` abgelegt. Thymeleaf ermöglicht es, dynamische Inhalte in die HTML Templates einzufügen. Dazu werden die Thymeleaf-Attribute in den HTML-Dateien verwendet. Greift ein Nutzer auf eine Seite zu, wird das entsprechende HTML Template gerendert und die dynamischen Inhalte eingefügt. Statische Ressourcen wie CSS-Dateien, JavaScript-Dateien und Bilder werden im Ordner `src/main/resources/static` abgelegt.

8.3.1. Code Blöcke

Zur Generierung von Code Blöcken wird das JavaScript-Plugin `prism.js` verwendet. Prism.js ermöglicht es, Code Blöcke in HTML-Seiten zu erstellen und zu formatieren. Dazu wird das Plugin in den HTML-Dateien eingebunden und die Code Blöcke entsprechend markiert.

8.4. Authentifizierung

Die Nutzerauthentifizierung wird über Spring Security realisiert. Nutzer loggen sich über GitHub ein. Dazu wird OAuth2 verwendet. Beim Login wird der Nutzer auf die GitHub-Login-Seite weitergeleitet. Nach erfolgreicher Anmeldung wird der Nutzer auf die Startseite der Anwendung weitergeleitet. Dabei werden GitHub ID sowie der GitHub Benutzername in der Datenbank gespeichert, um später die Zuordnung zu den Nutzerdaten zu ermöglichen.

OAuth2 wird in der Konfiguration konfiguriert. Dabei müssen die Client-ID und das Client-Secret angegeben werden. Zusätzlich müssen Korrektoren und Administratoren in der Konfiguration definiert werden. Dazu werden die GitHub IDs der Nutzer angegeben. Beim Login wird überprüft, ob der Nutzer in der Liste der Korrektoren oder Administratoren enthalten ist und entsprechende Rollen zugewiesen.

Die ID eines Nutzer lässt sich unter `https://api.github.com/users/github_name` (ohne abschließenden Schrägstrich) abrufen. Dabei wird die ID unter dem Feld `id` angezeigt.

Es ist möglich, die Art der Authentifizierung zu ändern, Studi Quiz erwartet lediglich, dass Nutzer über eine eindeutige ID identifiziert werden können und dass jedem Nutzer ein Benutzername zugeordnet ist.

8.5. Fehlerbehandlung

Fehler bezüglich inkonsitenter Daten, Zugriffe auf nicht existierende Ressourcen oder fehlerhafte Anfragen werden über Exceptions abgefangen. Diese werden zu HTTP-Statuscodes gemappt und an den Nutzer zurückgegeben. Technische Fehler, wie der Ausfall der Datenbank, werden nicht abgefangen und führen zu einem Serverfehler.

8.6. Internationalisierung

Studi Quiz unterstützt keine Internationalisierung. Die Anwendung ist nur auf Deutsch verfügbar.

8.7. Logging

Studi Quiz loggt Zugriffe auf die Anwendung. Dabei wird der Nutzername, die aufgerufene URL, der HTTP-Statuscode und die Uhrzeit des Zugriffs protokolliert. Die Logdateien werden im Ordner `logs` abgelegt.

8.8. Konfiguration

Die Konfiguration der Anwendung erfolgt über die Datei `application.yaml`. Dort können die Datenbankverbindung, die OAuth2-Konfiguration, die Liste der Korrektoren und Administratoren sowie der Port der Anwendung angegeben werden. Alternativ kann die Konfiguration auch über Umgebungsvariablen erfolgen.

Mehr Informationen zur Konfiguration finden sich im README der Anwendung.

Chapter 9. Architekturentscheidungen

9.1. Verwendung von Spring Data JDBC als Interface zur Datenbank

Problemstellung

Beim Zugriff auf eine Datenbank in einer Spring-Anwendung gibt es verschiedene Möglichkeiten.

Betrachtete Alternativen

- Spring Data JPA
- Spring Data JDBC

Entscheidung

Beide Alternativen sind grundsätzlich möglich. Spring Data JDBC bietet CRUD-Operationen und Aggregat-Unterstützung. SQL Ausdrücke werden dann ausgeführt, wenn man es erwartet. Spring Data JPA demgegenüber bietet eine höhere Abstraktionsebene und eine größere Menge an Features (z.B. Caching, oder Lazy Loading).

Für Spring Data JDBC wurde sich entschieden, da die Anwendung keine komplexen Anforderungen an die Datenbank hat und der Entwickler nur Spring Data JDBC beherrscht und keine Erfahrung mit Spring Data JPA hat, wodurch die Entwicklung im Zeitplan bleibt.

Konsequenzen

- Die Anwendung ist einfacher zu entwickeln, da der Entwickler bereits Erfahrung mit Spring Data JDBC hat.
- Das Laden von mehreren Aggregaten, welche Entitäten in einer weiteren Datenbanktabelle referenzieren, führt zu vielen SQL-Queries, da keine Lazy Loading-Unterstützung vorhanden ist. Dadurch muss manuell ein Join durchgeführt werden.

Risiken

Die Wahl von Spring Data JDBC stellt kein großen Risiko dar. Die `persistente` Schicht kann bei Bedarf ohne Änderungen an anderen Schichten bearbeitet werden, sodass ein Wechsel zu Spring Data JPA möglich ist.

9.2. Datenbank

Problemstellung

Die Anwendung benötigt eine Datenbank, um Daten zu speichern. Es gibt viele verschiedene Datenbanken, die sich in ihrer Funktionalität und Performance unterscheiden.

Betrachtete Alternativen

- MySQL
- PostgreSQL

Entscheidung

Bei der Auswahl der Datenbanken wurde sich auf relationalen Datenbanken beschränkt, da die Daten der Anwendung eine feste Struktur haben und somit NoSQL-Datenbanken nicht benötigt werden. Die Wahl fiel auf PostgreSQL. Für die Anwendung macht es keinen signifikanten Unterschied, ob MySQL oder PostgreSQL verwendet wird. PostgreSQL wurde gewählt, da es eine größere Menge an Features bietet und die Performance für die Anwendung ausreichend ist.

Konsequenzen

- Die Migrationen der Datenbank müssen mit PostgreSQL durchgeführt werden.

Risiken

Die Wahl von PostgreSQL stellt kein Risiko dar. Ein Wechsel zu MySQL ist ohne großen Aufwand möglich, lediglich die Migrationen müssen angepasst werden.

9.3. Onion Architektur als Architekturmuster

Problemstellung

Die Anwendung soll nach einem Architekturmuster entwickelt werden, um eine klare Struktur zu haben und die Anwendung besser zu warten.

Betrachtete Alternativen

- Onion Architektur
- Hexagonal Architektur

Entscheidung

Beide Architekturmuster sind für die Anwendung geeignet. Spring Boot bietet durch die automatische Injektion von Abhängigkeiten eine gute Unterstützung für beide Architekturmuster. Es wurde sich für die Onion Architektur entschieden, da sie eine klarere Struktur bietet und die Geschäftslogik besser von den technischen Details trennt. Zusätzlich hat der Entwickler bereits Erfahrung mit der Onion Architektur und sie wird in der Hochschule gelehrt und angewendet.

Risiken

Die Wahl der Onion Architektur stellt ein signifikantes Risiko dar. Ein Architekturwechsel skann nur mit großem Aufwand durchgeführt werden, da die gesamte Anwendung umstrukturiert werden muss.

9.4. Umsetzung von mehreren Durchläufen eines Quizzes

Problemstellung

Studierende können ein Quiz mehrmals bearbeiten. Dabei kann es vorkommen, dass der letzte Durchlauf nicht vollständig bearbeitet wurde. In diesem Fall sollen die Antworten des vorherigen Durchlaufs übernommen werden. Dabei kann es passieren, dass auch im vorherigen Durchlauf nicht alle Fragen beantwortet wurden.

Betrachtete Alternativen

- Bei der Erstellung eines neuen Durchlaufs werden die Antworten des vorherigen Durchlaufs kopiert und können bearbeitet werden.
- Jeder Durchlauf startet mit leeren Antworten und bei der Korrektur/Bewertung wird die letzte Antwort ermittelt.

Entscheidung

Es wurde sich für die erste Alternative entschieden. Dadurch können bei einer Wiederholung die alten Antworten angezeigt und nur die falschen Antworten korrigiert werden. Dies spart Zeit für die Studierenden und erhöht die Benutzerfreundlichkeit. Dies wäre auch bei der zweiten Alternative möglich, jedoch müsste dann jedes mal die letzte Antwort ermittelt werden.

Konsequenzen

- Die Anwendung ist benutzerfreundlicher, da Studierende nicht alle Fragen erneut beantworten müssen.
- Die Erstellung eines neuen Durchlaufs dauert länger, da die Antworten des vorherigen Durchlaufs kopiert werden müssen.
- Die Implementierung der Korrektur/Bewertung ist simpler, da nur der letzte Durchlauf betrachtet werden muss.
- Mehr Speicherplatz wird benötigt

Risiken

Die Wahl stellt nur ein kleines Risiko dar. Die Performance einbußen sind minimal und auch die Speicherplatzanforderungen sind nicht signifikant.

9.5. Dateiformat für den Import von Quizfragen

Problemstellung

Die Anwendung soll die Möglichkeit bieten, Quizfragen über eine Datei zu importieren. Dabei gibt es verschiedene Dateiformate, die verwendet werden können.

Betrachtete Alternativen

- CSV
- JSON
- XML
- TOML
- YAML

Entscheidung

Es wurde sich für das TOML Dateiformat entschieden. TOML ist ein einfaches und leicht lesbares Dateiformat, welches sich gut für die Speicherung von strukturierten Daten eignet. Die Entscheidung beruht primär auf die Verwendung von TOML in vorherigen (ähnlichen) Projekten.

Risiken

Die Wahl von TOML stellt kein Risiko dar. Ein Wechsel zu einem anderen Dateiformat ist ohne großen Aufwand möglich, da lediglich die `parser` Schicht angepasst werden muss.

Chapter 10. Qualitätsanforderungen

Dieser Abschnitt beinhaltet konkrete Qualitätsszenarien, welche die zentralen Qualitätsziele, aber auch andere geforderte Qualitätseigenschaften besser fassen. Sie ermöglichen es, Entscheidungsoptionen zu bewerten.

10.1. Qualitätsbaum

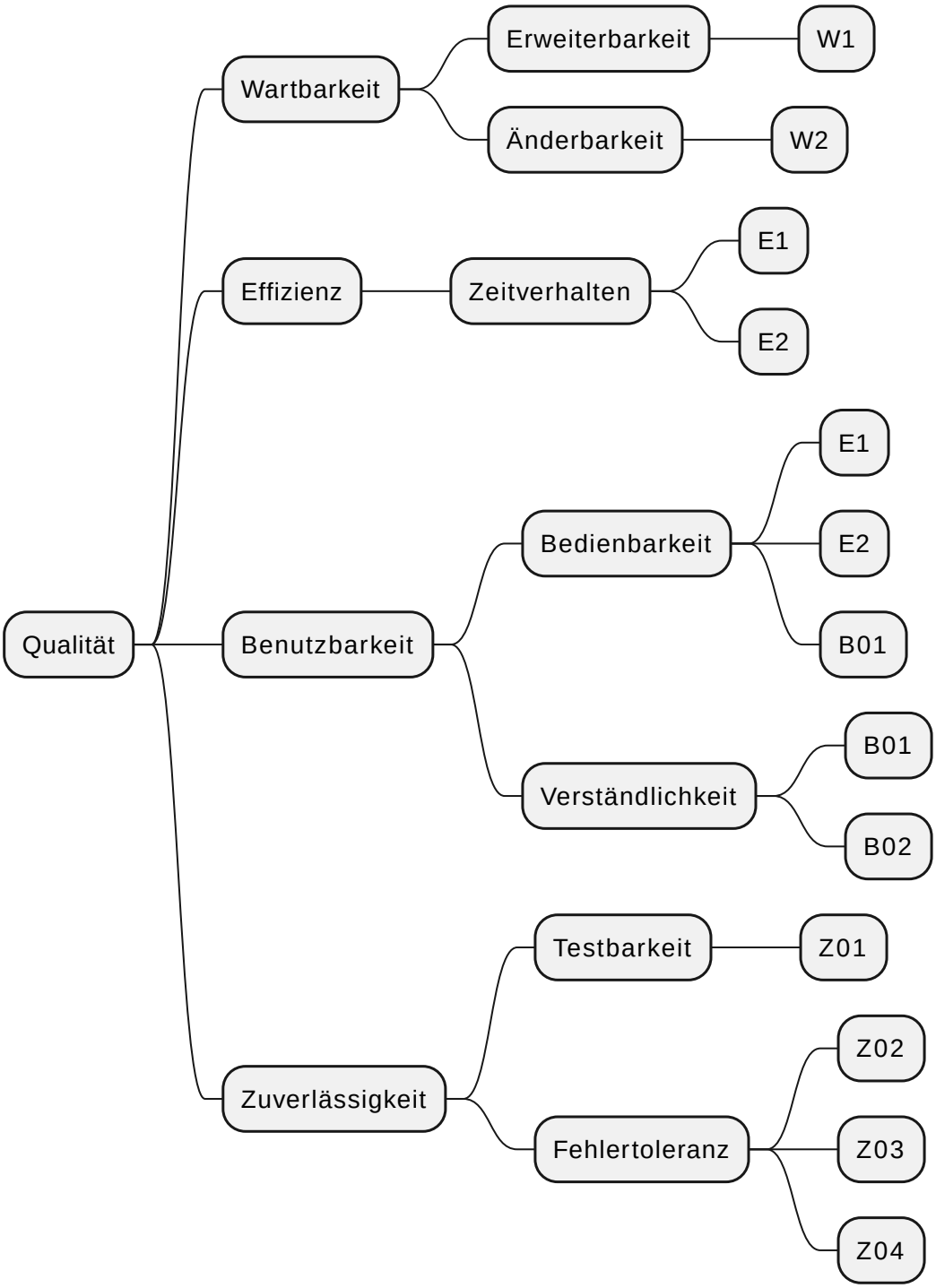

Abbildung 11. Studi Quiz, Qualitätsbaum

10.2. Qualitätsszenarien

Chapter 11. Risiken und technische Schulden

11.1. Datenverlust

Es ist möglich, dass bei unerwarteten Ereignissen die Datenbank beschädigt wird und Daten verloren gehen.

11.1.1. Risikobewertung

Wahrscheinlichkeit

niedrig

Schadensausmaß

hoch

11.1.2. Maßnahmen

Die Datenban sollte regelmäßig gesichert werden. Dies sollte automatisiert erfolgen, um menschliche Fehler zu vermeiden. Die Backups sollten an einem anderen Ort gespeichert werden, um bei einem Totalverlust des Servers die Daten wiederherstellen zu können.

11.2. Skalierung

Die Anwendung ist für eine bestimmte Anzahl an Nutzern ausgelegt. Bei einer steigenden Anzahl an Nutzern kann es zu Performanceproblemen kommen. Es konnte nicht getestet werden, ob die Anwendung bei einer großen Anzahl an gleichzeitigen Nutzern noch performant ist.

11.2.1. Risikobewertung

Wahrscheinlichkeit

mittel

Schadensausmaß

mittel

11.2.2. Maßnahmen

Es sollte ein Lasttest durchgeführt werden, um die Performance der Anwendung bei einer großen Anzahl an Nutzern zu testen.

11.3. GitHub für Authentifizierung

Die Anwendung authentifiziert Nutzer über GitHub. GitHub könnte in Zukunft die Authentifizierungsmethode ändern oder abschalten. Außerdem ist die Anwendung von GitHub abhängig, wenn GitHub offline ist, können sich Nutzer nicht anmelden.

11.3.1. Risikobewertung

Wahrscheinlichkeit

niedrig

Schadensausmaß

hoch

11.3.2. Maßnahmen

Es sollte sich frühzeitig nach Alternativen umgesehen werden, um im Fall der Fälle schnell reagieren zu können.

11.4. Kompatibilitätsprobleme mit Browsern

Die Anwendung wurde nur in den neuesten Versionen von Chrome und Firefox getestet. Es ist möglich, dass die Anwendung in anderen Browsern nicht korrekt funktioniert.

11.4.1. Risikobewertung

Wahrscheinlichkeit

niedrig, da keine spezifischen Browserfeatures verwendet werden

Schadensausmaß

mittel

Chapter 12. Glossar

12.1. Einführung

Quiz: Besteht aus einer Menge an **Fragen** und muss von **Studenten** `bearbeitet` werden.

- Ein **Quiz** hat ein **Erscheinungsdatum** und ein **Enddatum**, sowie ein **Korrekturdatum**.
- Ein **Quiz**, dessen **Erscheinungsdatum** noch nicht erreicht wurde, kann nicht `bearbeitet` und nicht `korrigiert` werden und wird als *nicht sichtbar* bezeichnet.
- Ein **Quiz**, dessen **Erscheinungsdatum** erreicht wurde, aber dessen **Enddatum** noch nicht erreicht wurde, kann `bearbeitet` aber nicht `korrigiert` werden und wird als *bearbeitbar* bezeichnet.
- Ein **Quiz**, dessen **Enddatum** erreicht wurde, aber dessen **Korrekturdatum** noch nicht erreicht wurde, kann nicht `bearbeitet` jedoch `korrigiert` werden und wird als *korrigierbar* bezeichnet.
- Ein **Quiz**, dessen **Korrekturdatum** erreicht wurde, kann nicht `bearbeitet` und nicht `korrigiert` werden und wird als *abgeschlossen* bezeichnet.
- Ein **Quiz**, dass *bearbeitbar*, *korrigierbar* oder *abgeschlossen* ist, kann von einem **Studenten** `betrachtet` werden und wird als *sichtbar* bezeichnet.
- Ein **Admin** oder **Lehrender** kann auch nach dem **Korrekturdatum** ein **Quiz** `korrigieren`.

Frage: Es gibt verschiedene **Fragetypen**. Eine **Frage** hat immer einen **Titel** und eine **Beschreibung** und optional einen **Lösung**stext.

Es gibt folgende **Fragetypen**:

- **Multiple Choice Frage**: Eine **Frage** mit einer Menge an **Optionen**, von denen eine oder mehrere *korrekt* sind, letztere werden als **korrekte Optionen** bezeichnet.
- **Single Choice Frage**: Eine **Multiple Choice Frage** bei der nur eine **korrekte Option** existiert.
- **Text Frage**: Eine **Frage** bei der die **Studenten** eine **Textantwort** in Textform eingeben müssen.
- **Lückentext Frage**: Eine **Frage** bei der die **Studenten** für eine oder mehrere Lücken im Text jeweils eine **Option** aus einer Menge von Optionen auswählen müssen oder einen **Text** eingeben müssen.

Student: Ein **Student** ist ein User, der **Fragen** in einem **Quiz** beantwortet, dazu muss er einen **Quiz Durchlauf** `starten`.

Quiz Durchlauf: Ein **Quiz Durchlauf** gehört zu einem **Quiz** und wird von einem **Student** `gestartet`. Zu einem **Quiz** können mehrere **Quiz Durchläufe** vom selben **Studenten** erstellt werden. Sie werden durch die **Durchlaufnummer** unterschieden. Der **Quiz Durchlauf** mit der höchsten **Durchlaufnummer** für einen **Studenten** ist der *letzte* **Quiz Durchlauf** (der einzige Quiz Durchlauf, der *aktiv* sein kann). Dieser wird auch zur ermittlung des **Ergebnisses** herangezogen.

- Zu einem **Quiz Durchlauf** gehören eine Menge an **Antworten**.
- Ein **Quiz Durchlauf** ist entweder *aktiv* oder *abgeschlossen*.

- Einem **Quiz Durchlauf** können nur **Antworten** hinzugefügt werden, wenn er *aktiv* ist und das zugehörige **Quiz** noch *bearbeitbar* ist.
- Ein **Quiz Durchlauf** wird von einem **Student** `beendet`.

Antwort: Eine **Antwort** gehört zu einem **Quiz Durchlauf** und beantwortet eine **Frage**.

- Eine **Antwort** kann eine **Bewertung** enthalten, die bei beim hinzufügen der **Antwort** *automatisch* generiert wird. Für einige **Fragetypen** ist eine nachträgliche **Bewertung** (durch eine *manuelle* **Korrektur**) notwendig.

Für die verschiedenen **Fragetypen** gibt es folgende **Antworttypen**:

- **Multiple Choice Antwort**: Eine **Antwort**, die eine Menge an **ausgewählten Optionen** enthält.
- **Single Choice Antwort**: Eine **Multiple Choice Antwort** die nur eine **ausgewählte Option** enthält.
- **Text Antwort**: Eine **Antwort**, die einen **Antworttext** enthält.

Bewertung: Eine **Bewertung** gehört zu einer **Antwort** und bewertet diese. Sie enthält die **Punkte** sowie einen **Kommentar**.

Aus den **Bewertungen** der **Antworten** eines **Quiz Durchlaufs** ergibt sich dann ein **Ergebnis**. Dieses beinhaltet die **Gesamtpunktzahl** und ob das **Quiz** *bestanden* bzw. *nicht bestanden* wurde.

Korrektor: Ein **Korrektor** ist ein User, der **Quizze** *manuell* `korrigiert`. Bei der **Korrektur** eines **Quiz Durchlaufs** werden die **Antworten** der **Studenten** *manuell* bewertet und als **Bewertung** gespeichert. Die **Bewertungen** der **Antworten** eines **Quiz Durchlaufs** ergeben dann zusammen ein **Ergebnis**.

Lehrender oder **Admin**: Ein **Lehrender** oder **Admin** ist ein User, der **Quizze** erstellen. Dieser Nutzer kann auch als **Korrektor** fungieren. Zur Erstellung eines **Quiz** muss der **Lehrende**/**Admin** eine TOML Datei hochladen, die die **Fragen** und **Antworten** enthält. Ein **Lehrender**/**Admin** kann die **Ergebnisse** sowie die **Antworten** der **Studenten** als CSV Datei exportieren.

12.2. Begriffe

Begriff	Synonym(s)	Definition
Quiz		Besteht aus einer Menge an **Fragen** und muss von **Studenten** `bearbeitet` werden.
Erscheinungsdatum		Datum ab dem ein **Quiz** für **Studenten** sichtbar ist.
Enddatum		Datum ab dem ein **Quiz** für **Studenten** nicht mehr bearbeitbar ist.
Korrekturdatum		Datum bis zu dem ein **Quiz** korrigiert werden kann. Der Korrekturzeitraum beginnt nach dem Enddatum.

Begriff	Synonym(s)	Definition
Frage		Aufgabe, die von Studierenden in einem **Quiz** beantwortet werden muss.
Fragetyp		Art einer **Frage**.
Multiple Choice Frage		**Frage** mit einer Menge an **Optionen**, von denen eine oder mehrere *korrekt* sind.
Single Choice Frage		**Multiple Choice Frage** bei der nur eine **korrekte Option** existiert.
Freitext Frage		**Frage** bei der die **Studenten** eine **Textantwort** in Textform eingeben müssen. Kann nicht automatisch bewertet werden.
Lückentext Frage		**Frage** bei der die **Studenten** für eine oder mehrere Lücken im Text jeweils eine **Option** aus einer Menge von Optionen auswählen müssen oder einen **Text** eingeben müssen.
Quiz Durchlauf		Zur Bearbeitung eines **Quiz** starten **Studenten** **Quiz Durchläufe**. Ein **Quiz** kann mehrere **Quiz Durchläufe** vom selben **Studenten** haben, falls das **Quiz** mehrmals bearbeitet werden kann. Der **letzte Quiz Durchlauf** wird zur Ermittlung des **Ergebnisses** herangezogen. Werden durch die **Durchlaufnummer** unterschieden.
Durchlaufnummer		Nummer, die einen **Quiz Durchlauf** eindeutig identifiziert. Inkrementell: Startet bei 1 und wird bei jedem neuen **Quiz Durchlauf** für einen **Studenten** für das selbe **Quiz** um 1 erhöht.
Antwort		Ergebniss der Beantwortung einer **Frage** in einem **Quiz**.
Bewertung		Punktezahl und Kommentar zur **Antwort** eines **Studenten**.
manuelle Bewertung		Bewertung einer **Antwort** durch einen **Korrektor**.
automatische Bewertung		Bewertung einer **Antwort** durch das System.
Student		Nutzer, der **Fragen** in einem **Quiz** beantwortet.
Korrektor		Nutzer, der **Antworten** von **Studenten** in einem **Quiz** bewertet.
Lehrender	Admin	Nutzer, der **Quizze** erstellt und **Ergebnisse** exportieren kann. Kann auch als **Korrektor** fungieren.

Begriff	Synonym(s)	Definition
Ergebnis		Gesamtpunktzahl und Bestehen eines **Quiz** für einen **Studenten**.
Java		Programmiersprache.
fat Jar		Ausführbare Jar Datei, die alle Abhängigkeiten enthält.
Jar Datei		Datei, die Java Code enthält und von Java ausgeführt werden kann.
Spring Boot		Framework für Java. Vereinfacht die Entwicklung von Java Anwendungen und die Verwendung von Spring.
CSV		Dateiformat das Daten in Tabellenform speichert. (Comma Separated Values)
TOML		Dateiformat für Konfigurationsdateien. (Tom’s Obvious, Minimal Language)

Chapter 13. Quellen

Für die Erstellung dieser Dokumentation wurde sich an folgenden Beispielen stark orientiert:

- https://www.dokchess.de/ von Stefan Zörner
- https://biking.michael-simons.eu/docs/ von Michael Simons

Zudem wurden folgende Quellen verwendet:

- https://docs.arc42.org/home/
- https://arc42.org/download (Verwendetes Template)