HAW HAMBURG

BACHELORTHESIS
Steffen Hespe

Erstellung eines Konzepts mit prototypischer Umsetzung von Korrekturkomponenten als Microservices im Rahmen einer Online-Programmierplattform

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Steffen Hespe

Erstellung eines Konzepts mit prototypischer Umsetzung von Korrekturkomponenten als Microservices im Rahmen einer Online-Programmierplattform

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Jens von Pilgrim
Zweitgutachter: Prof. Dr. Axel Schmolitzky

Eingereicht am: 26. Juli 2021

Steffen Hespe

Thema der Arbeit

Erstellung eines Konzepts mit prototypischer Umsetzung von Korrekturkomponenten als Microservices im Rahmen einer Online-Programmierplattform

Stichworte

Korrekturkomponente, Online-Programmierplattform, Microservice, Prototyp, Anforderungsanalyse

Kurzzusammenfassung

In dieser Thesis wird ein Konzept für Korrekturkomponenten einer Online-Programmierplattform unter der Verwendung von Microservices entworfen. Die von Studierenden und Dozenten an das System gestellten Anforderungen werden analysiert und in den Systemkontext gesetzt. Abschließend wird durch eine prototypische Umsetzung der entworfenen Komponenten die Umsetzbarkeit der Architektur demonstriert.

Steffen Hespe

Title of Thesis

Creation of a concept with prototypical implementation of grading components using microservices as part of a online programming platform

Keywords

Grading Service, Online Programming Plattform, Microservice, Prototype, Requirement Analysis

Abstract

In this thesis, a concept for grading components of an online programming plattform using microservice is designed. The requirements placed on the system by students and lecturers are analyzed and put into the system's context. Finally, the feasibility of the architecture is shown by a prototypical implementation of the designed components.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die ersten Programmierkenntnisse zu erlernen, ist in den letzten Jahren immer einfacher geworden. Die Menge an kostenlosen Informationen scheint unendlich groß zu sein und steht einem deutlich größerem Publikum leicht zur Verfügung. Zusätzlich zu Büchern und Dokumentationen entstanden neue Dienste und Angebote, zwischen denen Interessierte wählen können. Darunter befinden sich unter anderem Online-Programmierplattformen, bei denen man in allen denkbaren Programmiersprachen Aufgaben gestellt bekommt. Zwischen den Plattformen gibt es geringfügige Unterschiede und Alleinstellungsmerkmale, um sich von Konkurrenzprodukten abzuheben. Grundsätzlich folgen aber alle dem selben Prinzip: Eine neue Programmiersprache soll gelernt werden, wofür dem Nutzer in unterschiedlichen Anforderungsniveaus Aufgaben zu Problemen und Algorithmen gegeben werden. Auch an Hochschulen ist das Programmierenlernen ein wichtiger Bestandteil des Grundstudiums und könnte eventuell auf derartige Plattformen zurückgreifen. Es gibt allerdings einige Gründe, warum die Verwendung einer existierenden Fremdsoftware nicht infrage kommt. Eigene Aufgabenstellungen bei Fremddiensten hinzuzufügen ist kaum möglich und falls man sich auf fremde Anfängeraufgaben festlegt, könnten diese jederzeit vom Betreiber wieder entfernt werden. Zusätzlich ist Gruppenarbeit ein wesentlicher Bestandteil des Lernkonzepts an Hochschulen, ein potentieller Dienst müsste dies ebenfalls unterstützen.

Damit genau auf eigene Bedürfnisse eingegangen werden kann und um über entstehende Daten selbst entscheiden zu dürfen, wurde das Projekt „Online Programming Plattform for Software Engineering Education“ (OPPSEE) ins Leben gerufen.

1.1 Ziel der Arbeit

Die zu erstellende Programmierplattform lässt sich in mehrere kleine Teilbereiche unterteilen. Darunter fallen beispielsweise das Verwalten der Aufgaben oder das Speichern

und Auswerten der abgegebenen Nutzerdaten. In dieser Arbeit werden alle notwendigen Bestandteile für ein Korrektursystem einer Online-Programmierplattform ermittelt und anschließend in Form eines Prototypen umgesetzt. Der Prototyp soll den vollständigen Ablauf einer Aufgabenabgabe bei OPPSEE ermöglichen und benötigt daher sowohl ein Backend für die Ausführung von Testfällen, als auch ein Frontend, um die ermittelten Ergebnisse in geeigneter Form dem Nutzer zu präsentieren. Im Prototypen sollen neue Korrekturmöglichkeiten einfach ergänzt werden können, ohne im gesamten System Änderungen vornehmen zu müssen. Um diese Austauschbarkeit zu ermöglichen, ist der Einsatz von Microservices ein geeigneter Ansatz. Diese stehen jeweils für einen eigenen Aufgabentyp, wie zum Beispiel die Auswertung von Multiple-Choice-Fragen.

1.2 Zielgruppe

Von dieser Arbeit sollen einige unterschiedliche Zielgruppen angesprochen werden. Für Lehrende, die nach der Fertigstellung von OPPSEE Studierende über diese Plattform betreuen wollen, sind Informationen zu Statistiken im Bezug auf Kursverständnis zu vermittelten Themen und eine Schwierigkeitsermittlung der gestellten Aufgaben relevant.

Aus einem anderen Blickwinkel sind die Erkenntnisse für zukünftige Bearbeiter der Online-Programmierplattform wichtig, wenn weitere Funktionen und Aufgabenbereiche in die Korrekturkomponenten aufgenommen werden sollen. Dabei können insbesondere aus der Dokumentation des Prototypen die Schnittstellen für das Andocken an den bestehenden Korrekturdienst genommen werden.

Zwischen dem Front- und Backend der Korrekturkomponente des Prototypen werden unabhängige Dienste liegen, mit denen kommuniziert wird. Bei einem Zugriff auf eine Komponente des Korrektursystems muss dem externen Dienst die Schnittstelle bekannt sein, weswegen auch für Entwickler des restlichen OPPSEE Projekts der Prototyp Informationen bietet.

1.3 Abgrenzungen

Der Prototyp soll an dieser Stelle nur die Grundfunktionalitäten darlegen und einen möglichen Programmablauf zeigen. Dafür können zunächst einige Aspekte vernachlässigt werden, die in einem vollständigen System notwendig wären. Unter anderem ist die

komplette Sicherheit ignoriert worden. Dies betrifft beispielsweise die Ausführung von Java-Code, welcher ungeprüft aus der Nutzereingabe innerhalb von Testfällen ausgeführt wird. Ein großer Bestandteil von OPPSEE sind die verschiedenen Nutzerrollen, die bereits in vorherigen Arbeiten beschrieben wurden. In der Veröffentlichung „Towards an Online Programming Platform Complementing Software Engineering Education“ [4] von Niels Gandraß, Torge Hinrichs und Axel Schmolitzky wurden bereits vier Personengruppen beschrieben. Aus den Gruppen Student, Lecturer, Developer und Administrator werden im Folgenden nur die ersten beiden betrachtet. Der Developer ist sowohl für die Plattformentwicklung, als auch für die Erstellung von neuen Aufgaben zuständig. Gerade der zweite Aspekt fällt in den Bereich der Aufgabenverwaltung, die in diesem Prototypen explizit ausgegrenzt wird. Der Administrator steht für die Instandhaltung des Services, welche für die Ausführung der geplanten Szenarien nicht notwendig ist.

Für die beiden vorhandenen Rollen Student und Lecturer werden notwendige Funktionen bereitgestellt, es wird allerdings nicht mit fester Rechtevergabe gearbeitet. Das heißt, dass beide Nutzergruppen auf die Funktionen der jeweils anderen Gruppe zugreifen können. In einem fertigen Produkt sollten diese unerwünschten Aufrufe nicht angeboten werden, zudem müsste man bei den Schnittstellen die Gruppenzugehörigkeit prüfen.

1.4 Struktur der Arbeit

Im Kapitel 2 dieser Arbeit wird anhand von mehreren zur Hochschule passenden Szenarien eine Anforderungsanalyse erstellt. Dabei sind die Perspektiven der Student- und Lecturersicht gezeigt, welche in derselben Situation unterschiedliche Bedürfnisse haben. Die in der Anforderungsanalyse ermittelten Anforderungen müssen in den vorgegebenen Systemkontext gestellt werden, wodurch sich bereits vor Implementierung des Prototypen Einschränkungen ergeben.

Das Kapitel 3 beinhaltet die Beschreibung des erstellten Prototypen. Dabei wird zunächst der Gesamtaufbau diskutiert und anschließend die Struktur der einzelnen Komponenten genauer betrachtet. Unter anderem wird gezeigt, in welchen Bereichen bestimmte Daten, beispielsweise in Form von Statistiken, verarbeitet werden und wie die Schnittstellen zwischen den Komponenten aufgebaut sind.

Abschließend werden in Kapitel 4 die Ergebnisse zusammengefasst und gezeigt, welche Erkenntnisse durch die Arbeit erreicht wurden. Dabei gibt es einen Ausblick in die

nahe Zukunft, an welchen Stellen innerhalb der Korrekturkomponenten weitergearbeitet werden kann. Außerdem gibt es Installationshinweise für die Ausführung des erstellten Prototyps.

1.5 Definitionen und Konventionen

Die bereits verwendeten Rollenbezeichnungen *Student* und *Lecturer* [4] werden in dieser Arbeit weiterverwendet. Nutzer aus der Student Gruppe möchten ihre eigenen Programmierfähigkeiten verbessern und versuchen daher vorgeschlagene Aufgaben zu lösen. Der Lecturer betreut Kurse und möchte seinen Schülern die Möglichkeit bieten, anhand von begleitenden Aufgaben den Unterrichtsstoff zu vertiefen oder mit ähnlichen Problemstellungen zu experimentieren.

Im Folgenden werden für OPPSEE mehrere Abschnitte beschrieben, die ebenfalls bereits in der vorangehenden Literatur definiert wurden. Hauptbestandteil dieser Arbeit wird der *Grading Service* sein, der für die Verwaltung der eingereichten Nutzerlösungen zuständig ist. Davon zu differenzieren sind die jeweiligen *Grader*, welche im Anschluss zu den Aufgaben gehörige Tests ausführen und die Abgabe bewerten. Die Tests können abhängig vom Grader beispielsweise ein JUnit Test oder Multiple-Choice-Fragen sein. Benannt sind die Grader nach dem jeweiligen Aufgabentyp, der mit ihnen bearbeitet werden kann. Außerdem gibt es noch den *Assignment Service*, der für Aufgabenstellungen und zusätzlichen Informationen zu den Aufgaben verantwortlich ist. Dieser wird hier nur nebensächlich behandelt, ist allerdings im Prototyp notwendig, um Aufgaben darzustellen.

2 Konzept

Für die Erstellung eines Prototyps der Online-Programmierplattform müssen zunächst die gewünschten Anforderungen ermittelt werden. Mithilfe einer Anforderungsanalyse über vier Szenarien, die möglichst diversifiziert verschiedene Handlungsabläufe beschreiben, werden diese Anforderungen für die beiden existierenden Rollen Student und Lecturer ermittelt. Anschließend folgt eine Beobachtung des Systemkontext, der bestimmte Anforderungen einschränkt oder gewisse Grundlagen aufstellt, denen das System folgen muss. Dafür wird das Architekturmodell der Microservices eingeführt und auf den Entwurf bezogen. Das abschließende Ergebnis bietet die Grundlage für den folgenden Prototypen.

2.1 Anforderungsanalyse

Jedes der vier Szenarien soll einen eigenen Anwendungsfall beschreiben. Da die Online-Programmierplattform mehrere Aufgabentypen unterstützen soll, müssen diese bereits in den Szenarien aufgegriffen werden. Die möglichen Korrekturfunktionen sollten sich stark untereinander unterscheiden, damit möglich viele Anforderungen daraus gezogen werden können. Die drei Aufgabentypen sind eine Java Programmieraufgabe, ein Multiple Choice Fragebogen und eine Modellierung von UML Klassendiagrammen. In einem weiterem Szenario wird die Java Aufgabe um ein weiteres Bewertungskriterium ergänzt, dass die Anzahl an Zeilen der abgegeben Lösung zählt. Es wird jeweils sowohl die Student-, als auch die Lecturerperspektive betrachtet.

2.1.1 JUnit, Szenario 1

Ein Student aus dem zweiten Semester der Angewandten Informatik an der HAW Hamburg möchte als Hilfestellung zu den normalen Vorlesungen und Praktikumsaufgaben im Fach Programmiermethodik 2 als Klausurvorbereitung die eigenen Java Programmierkenntnisse verbessern. Der Student wählt eine einfache Aufgabe und soll dafür eine Funktion zur

rekursiven Berechnung der Gaußschen Summenformel erstellen. Ihm steht eine Java-Datei mit einem bereits vorhandenen Funktionsprototypen zur Verfügung, in welchen die eigene Funktion ergänzt werden soll. Bereits vor der Abgabe sind zwei Testfälle dargestellt, von denen noch keiner erfolgreich ausgeführt wurde. Dem Student fällt auf, dass in den angezeigten Testfällen die Funktion nur für einen einzigen legitimen Eingabewert geprüft wird, weshalb er ohne sinnvollen Code sofort das richtige Ergebnis zurückgibt. Den zweiten Testfall, in dem eine ungültige negative Eingabe benutzt wird, fängt der Student zusätzlich korrekt ab. Der Nutzer ist mit seinem Ergebnis zufrieden und reicht die implementierte Lösung ein.

Für den abgegebenen Quellcode werden die hinterlegten Tests in einer ausgelagerten Komponente ausgeführt. Zusätzlich zu den in der Entwicklungsumgebung sichtbaren Testfällen, gibt es noch weitere versteckte Tests, die für mehrere Grenzwerte ebenfalls die abgegebene Lösung auf Korrektheit prüfen. Aus allen durchgeführten Tests wird anschließend eine Note bestimmt. Da der Student nur die sichtbaren Testfälle behandelt hat, ergibt sich für ihn eine schlechte Note und die Aufgabe wurde nicht bestanden. Es handelt sich hierbei allerdings nicht um eine Aufgabe aus einer Prüfungssituation, weswegen eine erneute Abgabe erlaubt ist. Dies unterscheidet sich zwischen den einzelnen Aufgaben. Bei einem erneutem Versuch kann auf die vorherige Lösung zugegriffen werden. Der Student ändert seinen ursprünglichen Quellcode zu einem funktionierenden Programm und gibt diesen erneut ab. Allerdings wurde anstatt der gewünschten rekursiven Implementierung, eine iterative Lösung für die Gaußsche Summenformel erstellt. Es findet keine Static Code Analysis statt, die auf eine Verwendung von Rekursion prüfen könnte, weshalb der Student trotzdem die volle Punktzahl erhält.

In Abbildung 2.1 ist der Ablauf beim Abgeben einer Aufgabe entlang der beteiligten Akteure durch ein Sequenzdiagramm abgebildet. Nur die Rolle Student ist an dieser Stelle beteiligt und kommuniziert über die eigene Entwicklungsumgebung mit der Programmierplattform. Es entstehen zwei Anfragen: Das Auswählen der Aufgabe und das anschließende Abgeben. Die weitere Kommunikation ist für den Student nicht einsehbar. Der Auftrag wird zur zugehörigen Komponente weitergeleitet und dort behandelt. Abschließend wird das Ergebnis visualisiert.

Nachdem die Aufgabe bereits einige Stunden für Kursteilnehmer freigeschaltet ist, möchte der zugehörige Lecturer Informationen abrufen, um die Aufgabenqualität, sowie die Leistung der Studierenden zu beurteilen. Dabei fällt ihm auf, dass bei der Erstellung einer rekursiven Gaußschen Summenformel ein deutlich besserer Kursschnitt als in vergleich-

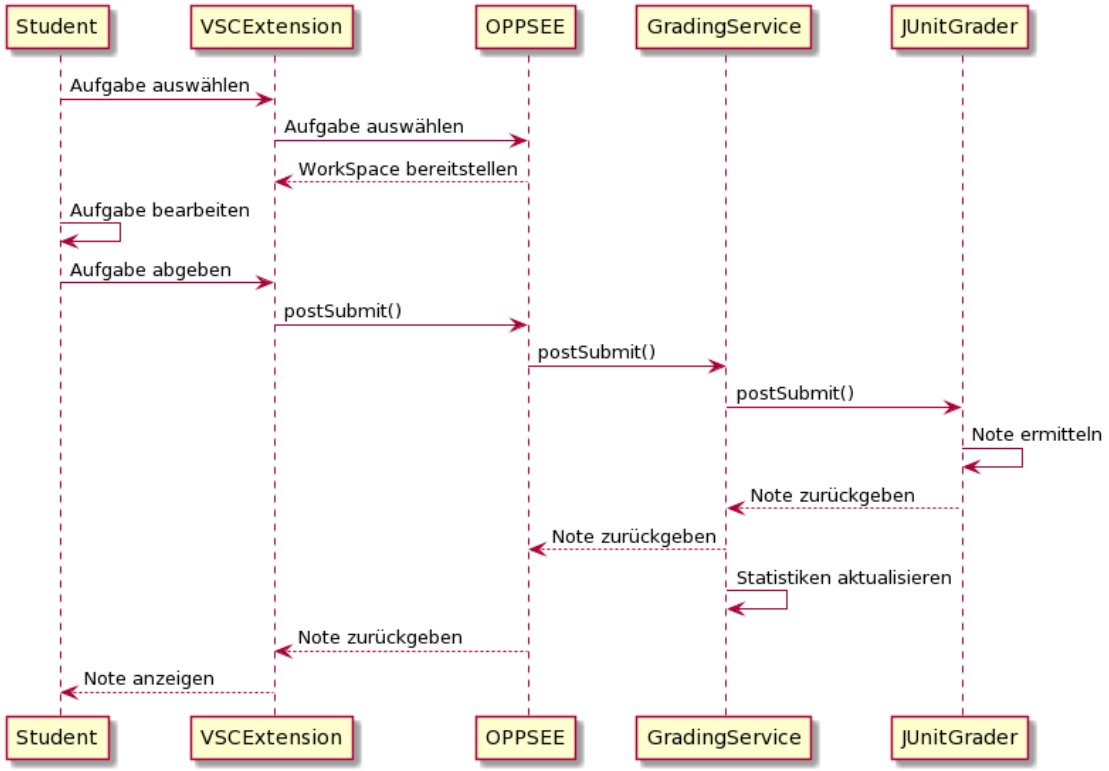


Abbildung 2.1: Sequenzdiagramm JUnit Grader, Student, Aufgabenabgabe

baren Aufgaben erzielt wurde und überprüft die vorhandenen Testfälle. Der Lecturer findet den ausgenutzten Fehler und versucht ihn zu beheben. Aus dem ursprünglichen Test für einen festen Wert wird stattdessen eine zufällige Eingabe zwischen eins und 20. Zukünftige Ausführungen werden mit den veränderten Testfällen durchgeführt. Nach der Bearbeitung erhalten alle Studierende, die bereits eine Lösung mit den alten Tests eingereicht haben, eine Benachrichtigung, die auf eine Änderung der Aufgabe hinweist.

Das Sequenzdiagramm 2.2 stellt den Ablauf der Statistikabfrage dar. Der Lecturer fragt ebenfalls in der Entwicklungsumgebung die Statistiken der Summenformelaufgabe ab und erhält im Anschluss einen Bericht. Die Weiterleitung bis zur Bewertungskomponente verläuft wie im vorherigen Beispiel des Studierenden. Damit die zurückgegebenen Informationen aktuell bleiben, wurden in Grafik 2.1 beim Auswerten der abgegebenen Lösung bestimmte Daten gespeichert. Dies geschieht bei jedem Abgabevorgang.

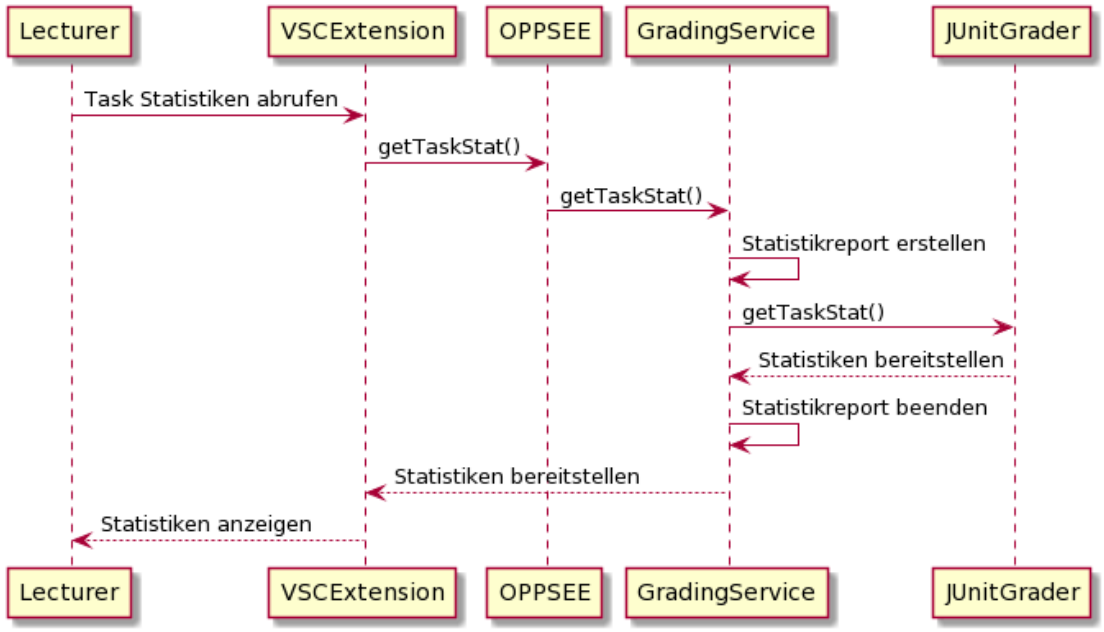


Abbildung 2.2: Sequenzdiagramm JUnit Grader, Lecturer, Statistiken abrufen

Im dritten Fall des ersten Szenarios (Abbildung 2.3) sind erstmals beide Nutzerrollen beteiligt. Nachdem der Auftrag zum Editieren des Tests gegeben wurde, werden alle betroffenen Teilnehmer über die vollzogene Änderung benachrichtigt.

Die beiden Aktivitätsdiagramme in Abbildung 2.4a und 2.4b zeigen ebenfalls den Ablauf des ersten Szenarios der beiden Nutzerrollen. In dieser Darstellung ist nur die Sicht der Akteure sichtbar. Alles Interne wurde bewusst ausgeblendet, um nur die Entscheidungen der Nutzer zu betrachten. Durch die folgenden Szenarien werden die Aktivitätsdiagramme ergänzt, damit abschließend der Verlauf durch alle Entscheidungen aus einem vollständigen Diagramm abzulesen ist.

2.1.2 Lines of Code, Szenario 2

Nachdem derselbe Student das Ergebnis für seine Abgabe erhalten hat, fragt er sofort die nächste Aufgabe an. Er bleibt im selben Themenbereich und lernt weiterhin Java für das Fach Programmiermethodik 2. Die Aufgabenstellung lautet, mehrere Werte aus einer vorgegebenen Textdatei zu importieren. In dieser Datei befinden sich bereits formatierte Personendaten, die in einer Java Liste gespeichert werden sollen. Die Elemente der Liste

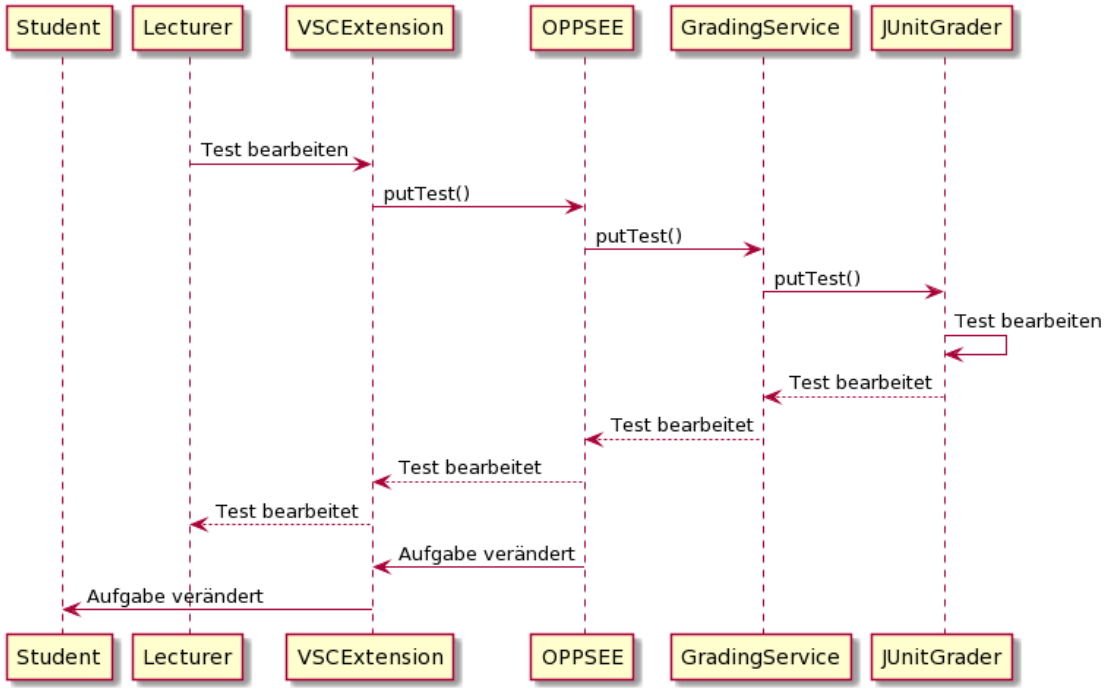
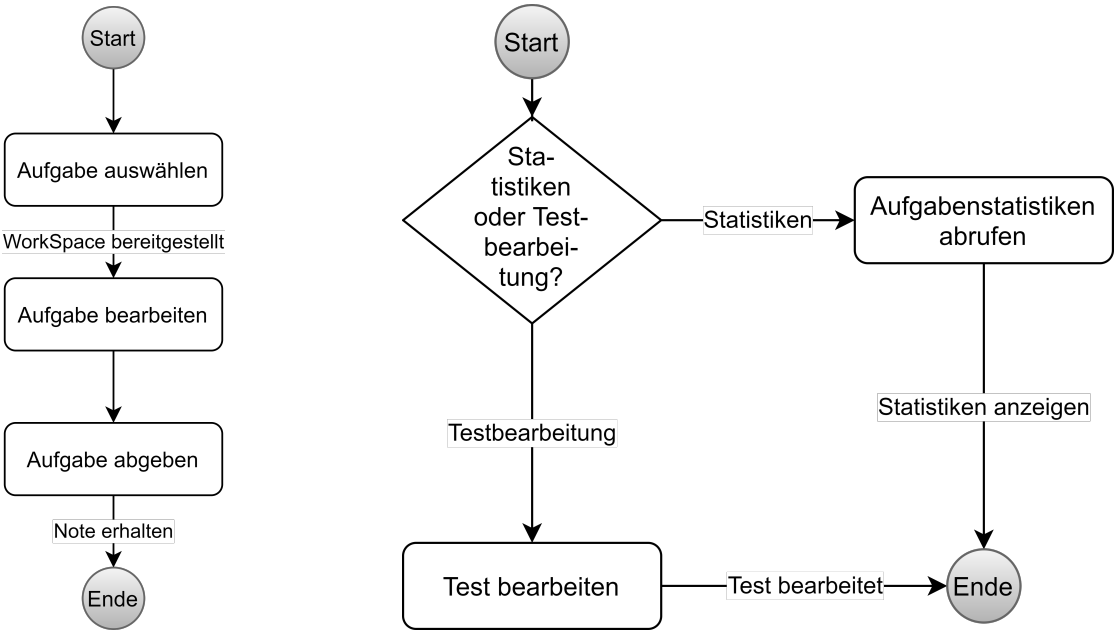


Abbildung 2.3: Sequenzdiagramm JUnit Grader, Lecturer, Test bearbeiten

sind jeweils Objekte aus einer vordefinierten Klasse Mensch, die mit den Daten aus der Textdatei erzeugt werden. Jede Zeile beschreibt ein Element der Liste. Die zu erstellende Methode soll die erzeugte Liste anschließend zurückgeben.

Für die Bearbeitung dieser Aufgabe gibt es zusätzliche Bedingungen, die verpflichtend zum Bestehen zu erfüllen sind. Erstens darf die Datei, in der die Aufgabe gelöst werden soll, höchstens 50 Zeilen umfassen. Zweitens stehen dem Student nur 30 Minuten für die Implementierung zur Verfügung. Bei einer verspäteten Abgabe wird die Aufgabe mit der Minimalnote bewertet. Dem Nutzer wird in der Entwicklungsumgebung die verbleibende Zeit angezeigt.

Der Student ist noch am Anfang der Vorbereitung und schafft es deswegen nicht, in den 30 Minuten eine funktionierende Lösung zu implementieren. Er verpasst trotz Anzeige der übrigen Zeit die letztmögliche Abgabemöglichkeit. Trotzdem gibt der Student noch die Aufgabe ab, obwohl er mit seiner Lösung aufgrund des Zeitmangels nicht zufrieden ist. In dem anschließend angezeigten Bewertungsbogen erfährt der Student, dass seine Lösung mit der Minimalpunktzahl bewertet wurde. Da er die erlaubte Zeit überschritten



(a) Student (b) Lecturer

Abbildung 2.4: Aktivitätsdiagramme JUnit Grader, Student und Lecturer, Szenario 1

hat, gibt es keine Informationen über die Qualität des abgegebenen Quellcodes. Die Tests wurden nicht ausgeführt.

Im Vergleich zum ersten Szenario, ist für diese Aufgabe eine Zeitmessung ergänzt worden. Im Sequenzdiagramm 2.5 wird gezeigt, dass die Abgabe trotzdem bis zur Bewertungskomponente weitergereicht wird, dort allerdings aufgrund der Zeitüberschreitung keine Testausführung stattfindet. Bevor die Aufgabe überhaupt ausgewertet wird, muss ebenfalls an die Bewertungskomponente das Starten der Aufgabe signalisiert werden. Ab diesem Zeitpunkt läuft die Zeitmessung.

Beim erneuten Abfragen der Aufgabenstatistik fällt dem zuständigen Lecturer die Durchschnittsbearbeitungsdauer der Teilnehmer auf. Diese liegt bei etwa 28 Minuten und damit nur leicht unter der maximalen Zeit. In Verbindung mit der geringen Durchschnittspunktzahl beschließt der Lecturer die Bearbeitungsdauer von 30 auf 45 Minuten zu erhöhen. Es handelt sich schließlich nicht um eine Prüfungsleistung und es soll daher ausreichend Zeit zur Verfügung stehen, um gegebenenfalls bei Problemen bei der Bearbeitung in den Vorlesungsmaterialien nachzuschlagen. Durch das Anheben der maximal verfügbaren Zeit werden wieder alle Studierende benachrichtigt, die diese Aufgabe bereits bearbeitet hatten.

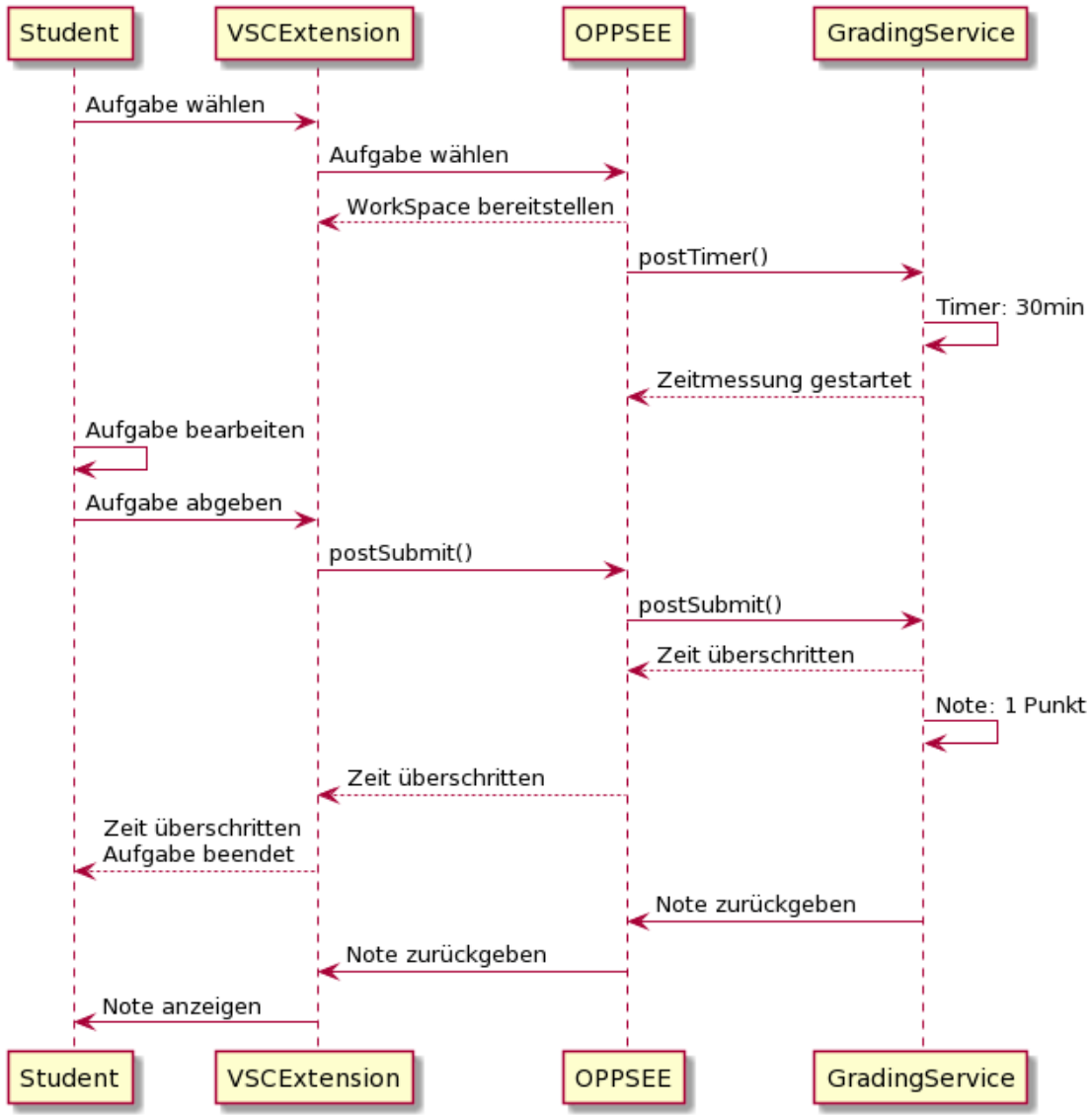


Abbildung 2.5: Sequenzdiagramm Lines of Code Grader, Student, Zeit abgelaufen

Der Vorgang des Lecturers ist dem Bearbeiten eines normalen Testfalls sehr ähnlich (siehe Abbildung 2.3). Allerdings wird hier kein einzelner Test bearbeitet, sondern in der Aufgabe die neue Maximalbearbeitungszeit eingetragen. Zusätzlich ist eine Bearbeitung der Aufgabenstellung im Assignment Service notwendig, damit die Nutzer bei einer späteren Bearbeitung die neue Zeit einsehen können.

Nachdem der Student die Benachrichtigung über eine Erhöhung des Zeitlimits erhalten hat, versucht er die Aufgabe erneut zu bearbeiten. Diesmal gelingt ihm die Bearbeitung in der vorgegebenen Zeit. Allerdings hat der Student nicht daran gedacht, die Anzahl der benutzten Zeilen auf die geforderte Maximalzahl von 50 zu reduzieren und liegt daher mit 55 Zeilen über der erlaubten Grenze. Deswegen fällt der zurückgegebene Bewertungsbogen erneut mangelhaft aus. Da das Zeitlimit eingehalten wurde und nur Mängel im Programmcode existieren, sind diesmal die Ergebnisse der eigentlichen JUnit Testfälle in der Bewertung enthalten. Mehr als die Hälfte der Tests sind erfolgreich durchgeführt worden, weswegen der Student die Aufgabe ohne eine Zeilenbeschränkung bestanden hätte.

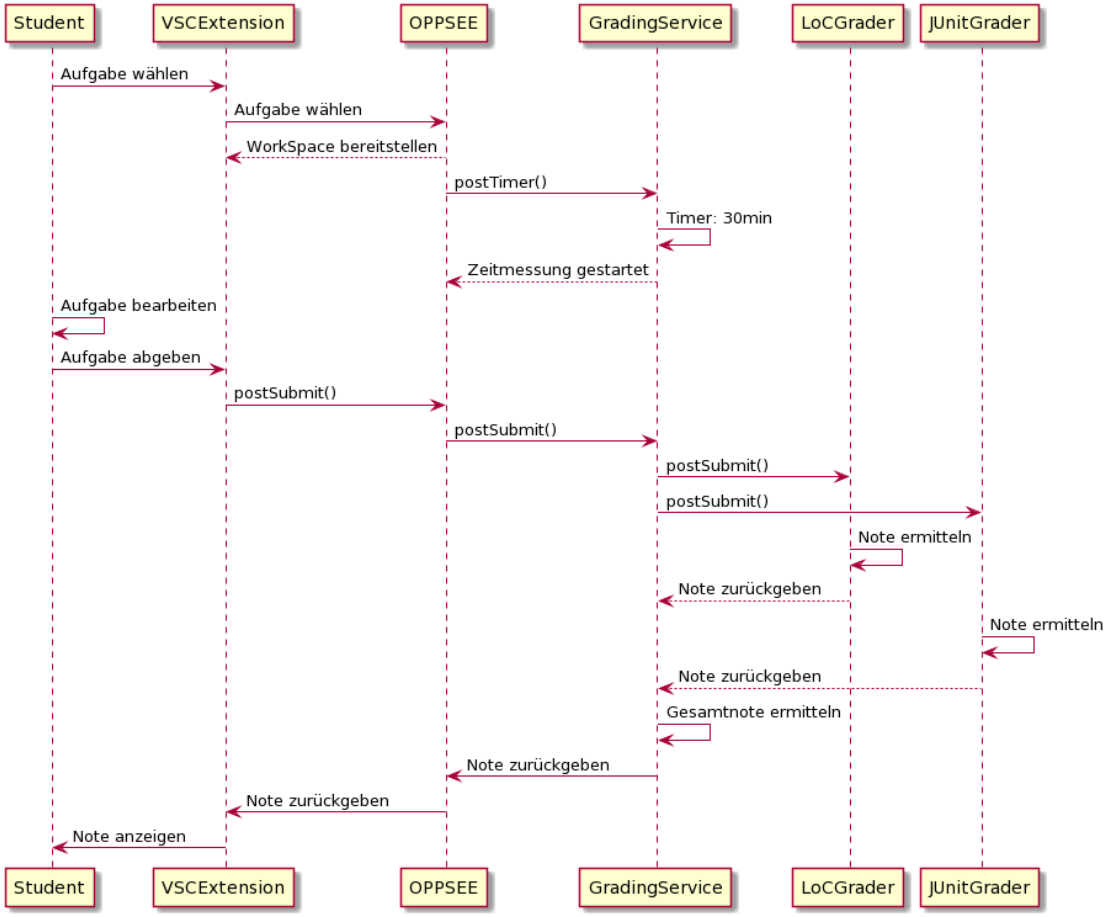


Abbildung 2.6: Sequenzdiagramm Lines of Code Grader, Student, zwei Grader

Dieser Fall (Abbildung 2.6) ähnelt dem ersten Abgabeversuch, bei dem eine Zeitüberschreitung vorlag. Wieder wird intern eine Zeitmessung gestartet, allerdings diesmal die Aufgabe rechtzeitig abgegeben. Vom ersten Szenario unterscheidet sich dieser Ablauf durch die Verwendung von sowohl JUnit Tests, als auch der Zeilenprüfung durch die Korrekturkomponente. Beide Aspekte werden bewertet und anschließend zu einer Gesamtnote zusammengefasst. Darauf folgt die Benachrichtigung des Students durch Anzeige des Bewertungsbogens.

2.1.3 Multiple Choice, Szenario 3

Nachdem der Student an seinen technischen Programmierkenntnissen gearbeitet hat, will er jetzt für die Theoriefragen desselben Kurses Programmiermethodik 2 lernen. Dazu wählt er einen Multiple-Choice Fragebogen zum Thema Objektorientierte Programmierung aus. Dieser lässt sich ähnlich wie die beiden bisherigen Aufgaben innerhalb der Entwicklungsumgebung absolvieren. Dieser Fragebogen besteht aus zehn Fragen mit jeweils fünf Antwortmöglichkeiten, wobei Mehrfachantworten erlaubt sind. Jede Frage hat mindestens eine richtige Lösung. Bei falschen Antworten gibt es einen Punktabzug. Zwischenzeitlich kann ein Student negative Punkte haben, bei der Auswertung würde dies allerdings auf die Mindestpunktzahl null erhöht werden. Es gibt bei dieser Aufgabe nur die einmalige Möglichkeit eine Lösung einzureichen. Die Lösungen und dazugehörigen Fragen können allerdings durchgängig nach der erstmaligen Bearbeitung eingesehen werden.

Der Student beantwortet neun der zehn Fragen und kreuzt im Schnitt zwei Möglichkeiten an. Jedoch lässt er eine der Fragen unausgefüllt, da er sich bei der Antwort unsicher ist und keinen Punktabzug riskieren möchte. Anschließend erhält er vom Bewertungssystem eine Note, die sich aus der Anzahl richtiger Antworten abzüglich der falschen Markierungen im Verhältnis zur Maximalpunktzahl ergibt.

Ein erfolgreicher Ablauf durch dieses Szenario läuft ähnlich wie die bisherigen Beispiele ab. Versucht jedoch der Student eine Aufgabe erneut abzugeben, wird dies wie in Abbildung 2.7 abgelehnt. Dies geschieht bereits zu Beginn der Aufgabe, beim Starten der Zeitmessung. Diese wird zwar nicht aufgrund einer Maximaldauer benötigt, wird allerdings trotzdem ausgeführt um später in den Statistiken dem Lecturer bessere Informationen bieten zu können.

Beim Abrufen der Statistiken für diese Multiple-Choice Aufgabe bemerkt der Lecturer, dass es bei einer der Fragen auffällig viele Fehler gibt. Er überprüft die Frage und bemerkt

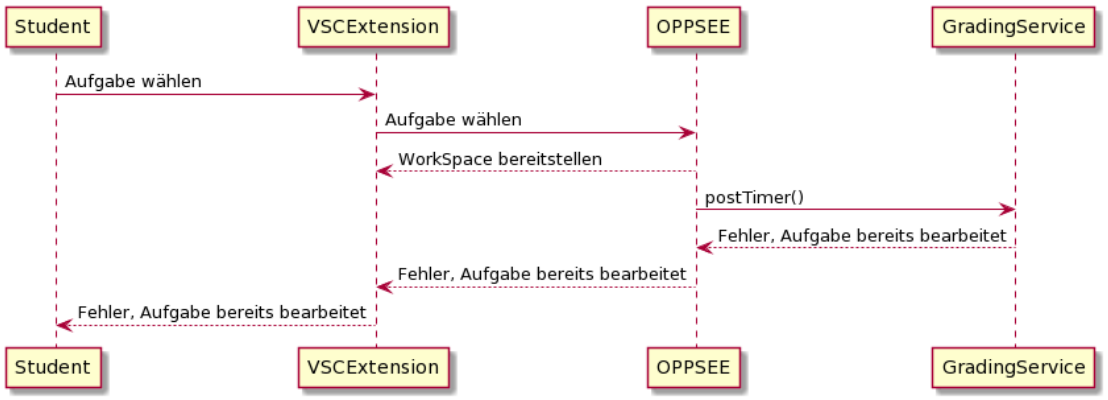


Abbildung 2.7: Sequenzdiagramm Multiple Choice Grader, Student, bereits abgegeben

eine undeutliche Fragestellung. Daher passt er sowohl die gestellte Frage, als auch zwei der möglichen Antwortmöglichkeiten an. Da der Lecturer den Multiple-Choice Fragenblock aus einem vorherigen Semester übernommen hat und eine der gestellten Fragen gar nicht mehr in seiner Vorlesung behandelt wird, löscht er die nicht mehr für die Prüfung relevante Frage. Studierende, die bereits den Fragebogen beantwortet haben, werden wieder benachrichtigt und ihre Sperre für eine erneute Durchführung wird aufgehoben.

Das Diagramm 2.8 stellt einen ähnlichen Vorgang wie das Bearbeiten von Testfällen dar. Der einzige Unterschied ist, dass diesmal sogar ein Testfall aus einer Multiple-Choice Aufgabe gelöscht wird.

2.1.4 UML Klassendiagramme, Szenario 4

Nach der Prüfungsphase mit erfolgreichen Bestehen der Programmiermethodik 2 Klausur, möchte sich der Student schon für das kommende Semester vorbereiten. Für den Kurs Software Engineering 1 will sich der Student bereits mit UML-Klassendiagrammen beschäftigen. Für diese Aufgabe werden ihm in vorgegebenen schreibgeschützten Java Code mehrere Klassen gezeigt. Die Beziehungen zwischen den Klassen, sowie sonstige Elemente eines UML-Klassendiagramms, wie zum Beispiel Klassenvariablen, Attribute und Sichtbarkeiten, sollen in einem bereitgestellten UML-Editor modelliert werden. Die Korrektheit dieser Eigenschaften führt zur späteren Bewertung der abgegebenen Lösung. Das zu erstellende Modell soll verschiedene Fahrzeugtypen beschreiben, die dafür vorgegebenen Klassen sind Fahrzeug, Auto und Motorrad.

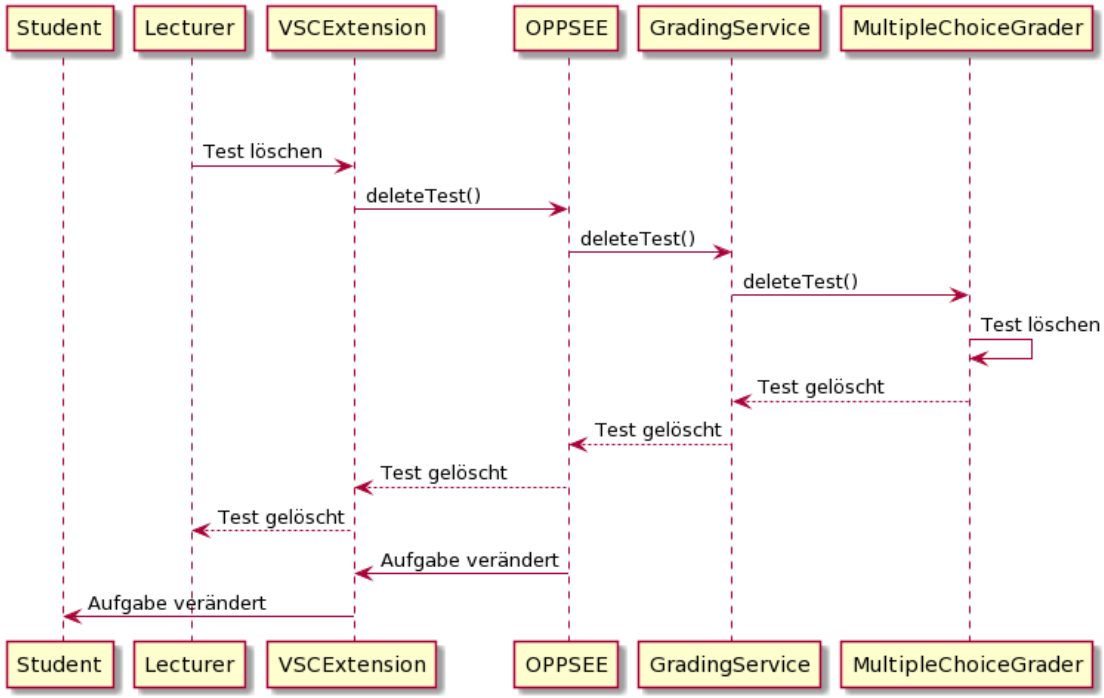


Abbildung 2.8: Sequenzdiagramm Multiple Choice Grader, Lecturer, Test löschen

Der Student modelliert korrekte Beziehungen zwischen den Klassen, vergisst jedoch eine Klassenvariable aus der Personen Klasse. Die Bewertung fällt sehr gut aus, da nur ein einziges Element nicht korrekt beschrieben wurde.

Die Abläufe zwischen den Komponenten von OPPSEE verhalten sich wie in den bisherigen Szenarien. Das abgegebene modellierte UML-Diagramms wird an die Korrekturkomponente weitergeleitet und dort ausgewertet. Nur intern wird wieder eine unterschiedliche Auswertung vollzogen, für den Nutzer verhält sich das System jedoch identisch.

Der Lecturer überlegt sich später nach Freigabe der Aufgabe noch eine weitere Klasse Bus in das Modell hinzuzufügen. Dafür muss er sowohl eine schreibgeschützte Klasse Bus zu den bestehenden drei Dateien hinzufügen und zusätzlich die Diagrammmusterlösung um den Bus, inklusive aller Beziehungen, ergänzen.

Im Gegensatz zu Änderungen in den bisherigen Szenarien, verändern sich an dieser Stelle nicht nur Testfälle von der Korrekturkomponente, sondern auch die Aufgabenstellung. Das Anpassen der Testfälle verläuft genau wie in Abbildung 2.3. Beim Bearbeiten der Aufgabenstellung (siehe Abbildung 2.9) wird nicht die Korrekturkomponente sondern

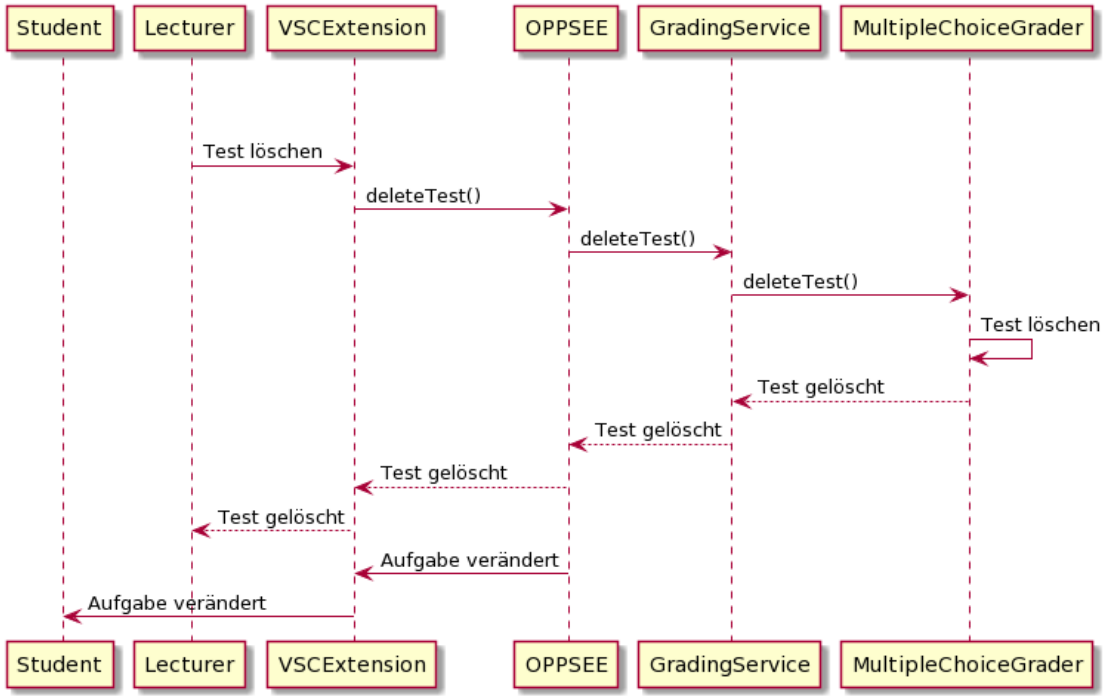


Abbildung 2.9: Sequenzdiagramm, Lecturer, Aufgabenstellung bearbeiten

der Assignment Service angesprochen. Der genaue Ablauf des Assignment Services liegt allerdings nicht im Bereich dieser Arbeit, weswegen dies im folgenden ignoriert wird.

2.1.5 Anforderungen

Aus den vier beispielhaften Szenarien aus Sicht der Student und Lecturer Rolle lassen sich Anforderungen für die Bewertungskomponenten ablesen. Wichtig für die Einordnung ist, dass vier verschiedene Aufgabentypen in den Szenarien vorkamen und daher nicht alle Funktionen der Bewertungseinheit daher gleichzeitig benutzt werden müssen. Es bietet sich allerdings an, grundsätzliche Bedürfnisse, die von allen derzeitig geplanten und zukünftigen Aufgaben benötigt werden, unabhängig von dem speziellen Anwendungsfall zur Verfügung zu stellen.

In Abbildung 2.10 ist mithilfe eines Aktivitätsdiagramms der gesamte Verlauf durch die Szenarien zusammengefasst dargestellt. Der blaue und gelbe Pfad beschreibt die Möglichkeiten des Lecturers mit Aufgaben zu interagieren, wie zum Beispiel in Szenario

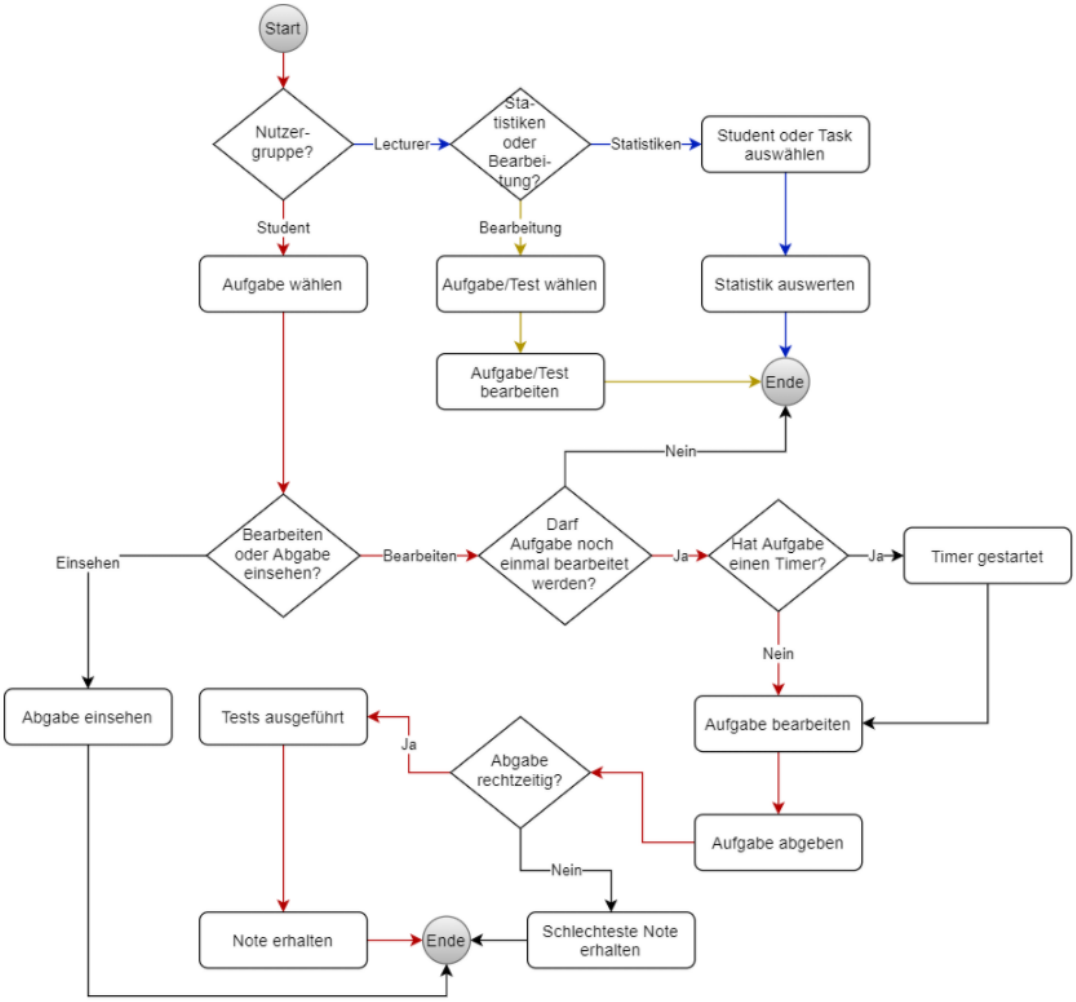


Abbildung 2.10: Aktivitätsdiagramm Alle Szenarien

1. Dabei steht das Bearbeiten für alle Funktionen an Aufgaben, also auch das Erstellen oder Löschen. Ein Student kann eine Aufgabe auswählen und diese entweder nur einsehen oder bearbeiten. Vor der Ausführung von Tests wird die Bearbeitungsdauer geprüft.

Der Lecturer muss neue Aufgaben inklusive dazugehöriger Testfälle erstellen können. Er benötigt Rechte zum Löschen und Bearbeiten von Tests, die zu selbsterstellten Aufgaben gehören. Das Recht alles bearbeiten zu können, liegt bei der Nutzerrolle des Assignment Creators. Trotzdem überschneiden sich dadurch die Aufgabenbereiche beider Rollen. Es sollen entweder wie im ersten Szenario direkt JUnit Tests, aber auch Grader spezifische

Eigenschaften wie die Maximallänge einer Datei oder eine Zeitbegrenzung veränderbar sein. Das Bearbeiten der Maximalzeit wie in Szenario 2 ist jedoch kein expliziter Testfall in der Bewertungseinheit, sondern wird im Assignment Service gesetzt. Manche Testfälle, die zum Beispiel eine Umgehung der eigentlichen Aufgabenstellung verhindern, können vom Nutzer versteckt werden. Dieser sieht eventuell nur eine bestimmte Menge der gesamten Tests. Als Lecturer soll die Sichtbarkeit der Testfälle verändert werden können. Auch dies ist im Assignment Service vermerkt.

In manchen Aufgaben wird explizit eine bestimmte Methode zum Lösen einer Aufgabe gefordert, wie beispielsweise die rekursive Lösung aus Szenario 1. Dies wird allerdings nicht überprüft und dient daher nur als Empfehlung an den Studierenden, um genau diesen Bereich zu lernen. Die volle Punktzahl lässt sich daher auch ohne eine solche Umsetzung erreichen. Da die Plattform aber nur zu Lernzwecken genutzt wird und nicht zu Extrapunkten oder der Prüfungsvorleistung führt, hat ein Umgehen der Aufgabenstellung keine Vorteile für den Studierenden und verhindern den Lernerfolg. Im Anschluss an diese Arbeit könnte mithilfe von Static Code Analysis versucht werden, verwendete Techniken wie eine Rekursion zu erkennen und vorauszusetzen. Dies wäre besonders wichtig, wenn OPPSEE in Zukunft auch für prüfungsrelevante Tests verwendet werden würde. Dies wird allerdings im folgenden nicht weiter beachtet.

Wie in den Szenarien mehrfach vorkam, muss es dem Lecturer ermöglicht werden, zu den einzelnen Aufgaben Statistiken einzusehen. Dabei ist es wichtig, für die Gesamtheit der abgegebenen Lösungen Durchschnittswerte angezeigt zu bekommen. Dies kann beispielsweise die Durchschnittsbearbeitungsdauer oder die Durchschnittspunktzahl einer Aufgabe oder eines Students sein. Mithilfe solcher Werte wird verdeutlicht, an welchen Aufgaben vermehrt Probleme auftreten oder welche womöglich gar nicht zu lösen sind. Das Einsehen von einzelnen Studentdaten ist dem Lecturer vorbehalten, jedoch könnten Nutzer ihre eigenen Daten abfragen und anonymisiert mit den restlichen Teilnehmern einer Aufgabe verglichen werden. Dadurch entsteht trotzdem ein Gefühl für das eigene Verständnis, ohne gezielt fremde Nutzerdaten abzugreifen.

Da zusätzlich in manchen Fällen Dateien vorgegeben sein können, muss ein Lecturer Zugriff und Möglichkeiten zur Änderung dieser haben. Dies war zum Beispiel wichtig im vierten Szenario, als eine komplette Datei mit Klasse ergänzt werden sollte. Wie im zweiten Fall kann eine unterschiedlich starke Gewichtung der Tests vorliegen. Dort war die Maximallänge des abgegebenen Codes eine Voraussetzung, die auf jeden Fall erfüllt werden musste. Dies muss ebenfalls vom Lecturer editierbar sein.

Aus Sicht der Student-Rolle muss es trivialerweise Zugriff auf die vorhandenen Dateien geben. Dies soll entweder über die eigene Entwicklungsumgebung oder über ein Programm im Browser möglich sein. Es sollten sich Änderungen vornehmen und der bearbeitete Stand mit dem eigenen Gitlab-Konto der Hochschule sichern lassen. In späteren Aufgaben kann es notwendig sein, dass mehrere Students an einer Aufgabe zusammenarbeiten und dementsprechend auf den selben Code zugreifen können müssen. Außerdem soll eine endgültige Einreichung des Codes mit anschließender Bewertung möglich sein. Die Benotung setzt sich je nach Aufgabe aus unterschiedlichen bekannten und unbekannten Tests zusammen. Im Falle der Multiple-Choice-Fragen muss der Student die Möglichkeit erhalten, einzelne Antworten zunächst auszulassen und erst am Ende zu beantworten.

Wie im ersten Szenario kann es vorkommen, dass nur bestimmte Dateien oder sogar nur Funktionen einer einzelnen Klasse verändert werden sollen. Dafür muss es Möglichkeiten geben, nur in bestimmten Klassen Schreibrechte an die Studentrolle zu verteilen. Auch nicht für den Studierenden vorgesehene Bereiche des Projekts sollten verborgen bleiben, damit beispielsweise kein Zugriff auf die pom.xml Datei eines Maven-Prozesses existiert. Der nur vorhandene Lesezugriff kann in Fällen wie dem UML-Szenario genutzt werden. Eventuell muss im Nachhinein die Sichtbarkeit und Bearbeitbarkeit einer Datei verändert werden, sollte aus bestimmten Gründen dem Studierenden doch mehr freigegeben werden sollen.

2.2 Microservices

Für die Online-Programmierplattform, und insbesondere die im Prototypen relevanten Grader, sollen die Komponenten als eigene Microservices umgesetzt werden. Unter einer Microservice Architektur versteht man das Trennen einer Anwendung in mehrere unabhängig voneinander betriebene Dienste [6]. Alle der durch die Trennung entstandenen Microservices erfüllen einen möglichst kleinen eigenständigen Zweck. Dies bewirkt, dass alle Dienste, bei Befolgung von Regeln bei den Schnittstellen, getrennt entwickelt und gewartet werden können. Diese Architektur schafft große Flexibilität und Wartbarkeit. Treten beispielsweise gravierende Fehler in einem einzelnen Microservice auf, lassen sich diese gezielt behandeln, ohne die Funktionalität der anderen Systeme einzuschränken. Fehler verteilen sich dadurch nicht im kompletten System und müssen nicht überall in einer einzigen großen Komponente gesucht werden.

Im Prototypen lassen sich die Grader jeweils als eigener Microservice realisieren. Dadurch können, nach der Erstellung einer allgemeingültigen Schnittstelle zum restlichen System, beliebig neue Grader integriert werden, ohne auf die bisherigen Rücksicht nehmen zu müssen. Verwaltet werden die Grader durch einen weiteren Microservice, den Grading Service. Dieser bekommt sämtliche Anfragen, die an einen Grader gerichtet sind, und verarbeitet diese. Aspekte, die alle Grader nutzen würden, können in den Grading Service ausgelagert werden, um Duplikationen zu minimieren und gegebenenfalls Anfragen zu sparen. Ein mögliches Problem bei der Microservice-Architektur ist die vermehrte Kommunikation zwischen den Diensten, die durch eine Trennung zwangsläufig entsteht. Diese ist im hier geplanten System allerdings weniger relevant, da zwischen den einzelnen Gradern keine Kommunikation stattfinden sollte. Es wird nur eine ohnehin notwendige Schnittstelle zum restlichen System benötigt, die nun von mehreren Gradern anstelle eines allgemeingültigen Graders angesprochen wird.

2.3 Randbedingungen des Systems

Zusätzlich zu den Anforderungen für die Korrekturkomponenten gibt es Einschränkungen des Systems, die eine Umsetzung erschweren könnten oder diese sogar komplett ausschließt.

Der Dienst OPPSEE soll vollständig im Browser benutzt werden können. Dafür muss bei der Nutzerinteraktion auf einen Dienst zurückgegriffen werden, der eine Online Entwicklungsumgebung bietet, die sich zusätzlich durch Erweiterungen modifizieren lässt. Dies ist durch die Open-Source Entwicklungsumgebung Theia gegeben, die auf Visual Studio Code (VSC) basiert. Theia wurde bereits in vorherigen Entwürfen für OPPSEE verwendet [5].

Dadurch sind alle Schnittstellen zur Erstellung von VSC Erweiterungen nutzbar. Durch die mit Visual Studio Code kompatible Erweiterung wäre theoretisch sogar eine Nutzung von OPPSEE außerhalb des Browsers möglich.

Als grundlegende Voraussetzung an die Bewertungskomponenten der Online-Programmierplattform ist eine leichte Erweiterbarkeit der Funktionalität gefordert. Dies führt dazu, dass jeder Aufgabentyp aus den vier Szenarien der Anforderungsanalyse in eine eigene Komponente ausgelagert werden sollte. Funktionen, die alle Aufgaben benötigen,

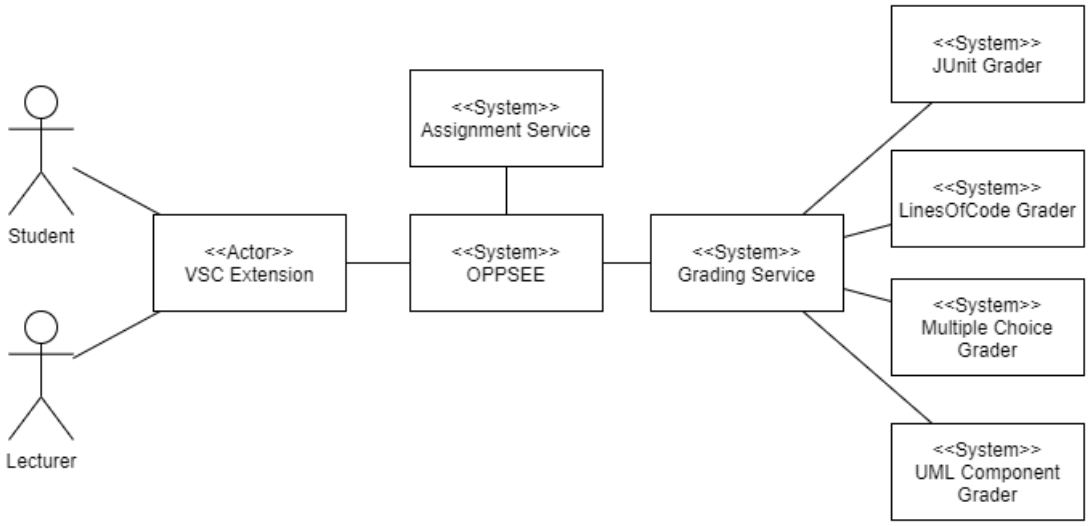


Abbildung 2.11: Systemkontextdiagramm

müssen dementsprechend in einer neuen gemeinsam genutzten Komponente implementiert sein. Darüber soll später auch die Kommunikation mit dem restlichen System ablaufen.

In dem Systemkontextdiagramm aus Abbildung 2.11 sind die Komponenten sowie beteiligte Akteure eingezeichnet. Die beiden Nutzerrollen Student und Lecturer greifen nur über die Visual Studio Code Extension auf das System zu. Zugriffe werden durch die Erweiterung bis zur Korrekturkomponente weitergeleitet. Die Anfragen behandeln immer eine bestimmte Aufgabe, wodurch bestimmt wird an welchen Microservice weitergeleitet wird. Anschließend findet die Auswertung der übergebenen Daten im jeweiligen Grader statt. Das Endergebnis wird dem Akteur über die VSC Extension in seiner Entwicklungsumgebung angezeigt. Der Assignment Service lässt sich ebenfalls anfragen, der Aufbau dieses Dienstes wird im folgenden allerdings nicht weiter betrachtet.

Der Grading und Assignment Service könnte jeweils direkt mit der Visual Studio Code Extension verbunden werden. Dies würde eine direkte Verbindung schaffen und Informationen müssten nicht über Umwege zum Nutzer geschickt werden. Wird dagegen eine weitere Komponente zwischen die Kommunikation geschoben, wird von der VSC Extension nur eine einzige Schnittstelle angesprochen. Von diesem Dienst lassen sich die eigentlichen Komponenten benachrichtigen, wodurch bei einer möglichen Änderung der Schnittstellen nur an einer Stelle im gesamten System der Zugriff angepasst werden muss. In der folgenden Implementierung dient OPPSEE daher nur als Weiterleitung, im

vollständigen Projekt würden noch andere Dienste, wie zum Beispiel das Speichern und Laden der abgegebenen Daten, integriert werden.

3 Prototyp

Die im vorherigen Kapitel ermittelten Anforderungen sollen durch mehrere Microservices umgesetzt werden. In diesem Prototyp lassen sich die beschriebenen Szenarien durchführen. Manche Anforderungen sind zunächst ausgelassen worden, aufgrund der gewählten Architektur und ähnlich funktionierenden, existierenden Implementierungen ist es jedoch leicht möglich diese anschließend hinzuzufügen. Der UML-Komponentendiagramm-Grader aus Szenario 4 ist im Prototypen nicht enthalten. Mithilfe des JUnit-, Lines-of-Code- und Multiple-Choice-Graders können die gewünschten Abläufe durch das System gezeigt werden. Auch eine Kombination von zwei verschiedenen Gradern ist mit dieser Auswahl möglich, wodurch ein weiterer Grader kaum Informationsgewinn bietet.

Der Grading-Service und die Grader wurden in Golang implementiert. Zum Anzeigen von Statistiken und Aufgabenauswertungen sind HTML-Templates innerhalb von Golang verwendet worden. Für die Kommunikation des Nutzers mit OPPSEE durch eine VSC Extension wird TypeScript benötigt. Damit die Komponenten auf allen Systemen funktionsfähig sind und ohne Mehraufwand gestartet werden können, laufen alle Komponenten über Docker und Docker-Compose. Zum Speichern von Daten wurden SQLite Datenbanken verwendet.

Durch die Verwendung von Microservices wurde eine Unabhängigkeit zwischen den Komponenten geschaffen. Dies ermöglicht es, dass neue Grader in anderen Sprachen entwickelt werden können. Einzig die Schnittstellen müssen korrekt ansprechbar sein.

3.1 Allgemeiner Aufbau

Die vollständige Korrektureinheit von OPPSEE soll gemäß den Anforderungen in mehrere Microservices geteilt sein. Die verwaltende Komponente der Korrekturkomponenten empfängt alle Anfragen, die an einen der Grader gesendet werden, und bearbeitet diese.

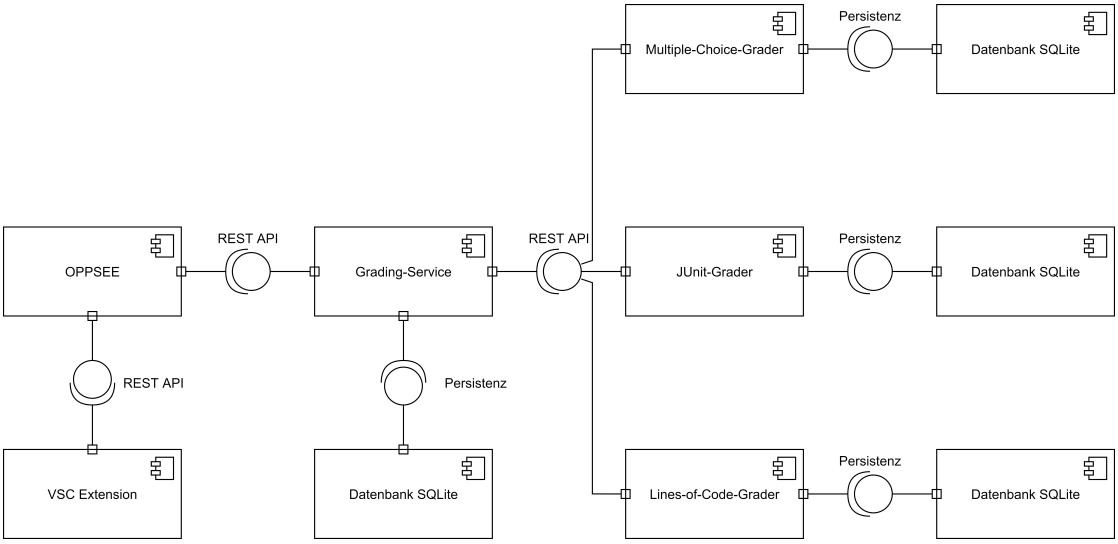


Abbildung 3.1: Komponentendiagramm

Im Komponentendiagramm in Abbildung 3.1 ist die einzige Schnittstelle zum Grading-Service dargestellt. Diese wird im Prototypen auch nur von der OPPSEE Komponente angesprochen. Damit Anfragen, die an den Grading-Service gestellt wurden, sinnvoll weitergeleitet werden können, müssen die Schnittstellen der Grader untereinander identisch sein. Ansonsten würde jeder neu hinzugefügte Grader eine Änderung im Grading-Service erzwingen. Die Grundfunktionen jedes Graders sind die Auswertung von Nutzerlösungen und das Bereitstellen von Statistiken.

Das Auslassen des Grading-Services und eine damit einhergehende direkte Kommunikation mit den Gradern würde zu Codeduplikation führen. So müsste jeder Grader selbst eine Zeitmessung durchführen oder komplett eigene Statistiken erstellen. Die Konsequenz daraus wäre eine höhere Fehleranfälligkeit, ein größerer Aufwand beim Erstellen von weiteren Gradern, sowie eine deutlich komplexere Fehlerbehebung. Außerdem wäre ein nachträgliches Bearbeiten der Grundfunktionen der kompletten Korrektureinheit von OPPSEE deutlich erschwert, weil jeder Grader individuell angepasst werden müsste.

Jeder Grader besitzt eine eigene Datenbank, um Testfälle und Ergebnisse von Testausführungen zu speichern. Aus den Ergebnissen können später die Nutzer- und Aufgabenstatistiken generiert werden. In der Datenbank des Grading-Services sind Informationen über Nutzerabgaben hinterlegt, die den beteiligten Gradern nicht bekannt sind, wie

beispielsweise die Bearbeitungsdauer oder die Gesamtnote einer Abgabe, welche im Falle von mehreren Gradern bei einer Aufgabe von den Teilnoten abweichen kann.

Die visuelle Darstellung in Form einer VSC Extension ist nicht direkt mit den restlichen Korrekturkomponenten verbunden. Die Erweiterung spricht wieder nur eine einzige Schnittstelle an, welche Anfragen abhängig vom Bedarf an den Grading-Service oder den Assignment Service weiterleitet. Der Assignment Service ist im Prototypen in die OPPSEE Komponente integriert.

3.2 Grading-Service

Im Prototypen kommuniziert der Grading-Service mit zwei verschiedenen Arten von Diensten. Zum einen mit Gradern, zum anderen mit dem restlichen System OPPSEEs. In Betracht auf den Gesamtablauf eines Szenarios aus der Anforderungsanalyse dient der Grading-Service daher als Weiterleitung des erhaltenen Auftrags an den zugehörigen Grader. Für die Ermittlung der richtigen Komponente werden vom Assignment Service Informationen über die jeweilige Aufgabe abgefragt. Dieser speichert zu jeder Aufgabe die zugehörigen Grader. Dementsprechend müssen nur für den Grading-Service relevante Informationen zu den jeweiligen Abgaben in der Datenbank gespeichert werden. Sobald ein Student eine Aufgabe beginnt, wird der Grading-Service benachrichtigt. In der Datenbank wird für den Nutzer und die Aufgabe ein neuer Eintrag mit dem aktuellen Zeitstempel als Startzeitpunkt der Aufgabe und dem Startwert 1 für die Anzahl an Versuchen angelegt. Falls es bereits einen Eintrag für diese Kombination aus Nutzer und Aufgaben gibt, wird der Startzeitpunkt überschrieben und die Versuchsanzahl inkrementiert. Gibt der Student anschließend seine Lösung ab, wird erneut die Zeit bestimmt und die Differenz seit Beginn der Aufgabe in der Datenbank als Bearbeitungszeit gespeichert. Existiert in der Aufgabenstellung eine Maximalbearbeitungszeit, wird an dieser Stelle geprüft, ob eine rechtzeitige Abgabe stattfand. Ist dies nicht der Fall, findet gemäß den Anforderungen keine Überprüfung statt und der Student besteht die Aufgabe nicht. Ansonsten wird der Quellcode zunächst zur Auswertung an die beteiligten Grader weitergeleitet. Die Ergebnisse aller Grader werden gesammelt und daraus eine gemeinsame Note erstellt. Gibt ein Grader die schlechtest mögliche Note zurück, ist automatisch die komplette Aufgabe nicht bestanden. Dies passiert beispielsweise in Szenario 2 beim nicht Erfüllen der Zeilenanforderung. Auch diese wird in der Datenbank vermerkt.

Da außerhalb des Grading-Services und der Grader keine Informationen über die Bewertung der Aufgaben existiert, muss in diesen Komponenten bereits alles für die visuelle Darstellung in der VSC Extension zusammengetragen werden. Dies geschieht über eine HTML-Datei, die bis zur Entwicklungsumgebung zum Nutzer weitergeleitet wird. In den dafür erstellten Grading-Report fließen die Ergebnisse der Grader mit ein. Es soll an dieser Stelle wieder möglichst viel vom Grading-Service übernommen werden, weswegen nicht die vollständige Datei vom jeweiligen Grader erstellt wird. Einen Teil der Informationen, wie zum Beispiel Aufgabenbezeichnung, die erhaltene Note und die Bearbeitungsdauer, ermittelt der Grading-Service selbstständig und trägt sie in den Grading-Report ein. Für die restlichen Informationen, die von den Gradern übertragen werden, sind im Template der HTML-Datei Platzhalter für HTML-Div-Blöcke vorgesehen. Jeder Grader liefert nach der Auswertung solch einen HTML-Div-Block, sowie eine ermittelte Note zurück. Da eine Aufgabe wie in Szenario 2, in dem sowohl der JUnit-Grader, als auch der Lines-of-Code-Grader verwendet wurde, die Bewertung von mehreren Komponenten benötigen kann, sammelt der Grading-Service alle Ergebnisse ein und platziert die Blöcke in der HTML-Datei. Es wird ein gewichtetes Mittel der Teilnoten berechnet und erst danach im Grading-Report eingetragen. Bei der Kombination des JUnit- und Lines-of-Code-Graders liegt kein gleiches Verhältnis vor, weil der Line-of-Code-Grader nur über das Bestehen entscheidet.

Aus der Anforderungsanalyse ergibt sich die Notwendigkeit Statistiken zu erheben und diese dem Student und Lecturer bereitzustellen. Aus den in der Datenbank gespeicherten Informationen lassen sich unterschiedliche Schlussfolgerungen ziehen. Der Grading-Service generiert für Nutzer- und Aufgabenstatistiken ähnlich wie beim Grading-Report eine HTML-Datei, die durch zusätzliche Informationen aus den Gradern ergänzt werden. Auch diese Datei wird bei Bedarf an die Entwicklungsumgebung des Nutzers gesendet.

3.2.1 Schnittstellen

Damit ein Nutzer über seine Entwicklungsumgebung Anfragen an den Grading-Service senden kann, muss dieser dafür Schnittstellen bereitstellen. Die Erste dient zur Benachrichtigung beim Starten einer Aufgabe. Diese akzeptiert nur eine eindeutige Nutzer- und Aufgaben-ID. Mit diesen Informationen kann der Grading-Service den Datenbankeintrag anlegen und sich den Startzeitpunkt merken. Als Rückgabe gibt es nur den HTTP-Statuscode 200 ohne weitere Daten, um einen erfolgreichen Erhalt der Anfrage zu signalisieren.

Für die endgültige Abgabe gibt es einen weiteren Aufruf. Hierbei werden wieder die Nutzer- und Aufgabenkennung zur Zuordnung, sowie diesmal ein Feld zur Datenübertragung, benötigt. Dieser String hat keine weiteren Vorgaben und ermöglicht damit den Gradern eigene Datenformate zu verwenden, die erst im jeweiligen Grader ausgewertet werden. So können in den Beispielen je nach Aufgabe entweder die abgegebenen Antworten eines Multiple-Choice-Tests oder der fertige Quellcode für einen JUnit-Grader enthalten sein. Nachdem die Daten an alle beteiligten Grader weitergeleitet und von diesen ausgewertet wurden, wird als Ergebnis der HTTP-Statuscode 200, der HTML-Div-Block und die ermittelte Note gesendet.

Für das Auslesen von Statistiken gibt es zwei verschiedene Schnittstellen. Es können jeweils Daten einer Aufgabe oder eines Nutzers abgerufen werden. Das Vorgehen dabei ähnelt wieder dem der Aufgabenabgabe. Diesmal wird nur entweder die Aufgaben- oder Nutzerkennung benötigt, für welche die Statistiken gesammelt werden sollen. Nach der Bearbeitung der Grader wird ebenfalls die HTML-Datei als Endergebnis an den Auftragsgeber gesendet. Zwischen den beiden Varianten gibt es außer den unterschiedlichen Informationen keine Unterschiede. Es folgt, dass nur für den jeweiligen Nutzer oder die jeweilige Aufgabe relevante Grader angefragt werden.

Die Grader sollen nicht direkt, sondern nur über den Grading-Service angesprochen werden. Daraus folgt, dass alle Funktionen, die einen Grader betreffen, auch vom Grading-Service aufrufbar sein müssen, wie zum Beispiel alles zu gespeicherten Tests. Dies sind die gängigen Datenbankzugriffe: Das Erstellen, Löschen, Bearbeiten und Anzeigen von Tests. Hierbei reicht allerdings fast eine Weiterleitung des selben Funktionsaufrufs, den auch der Grader erwarten würde. Einzig muss zusätzlich der Name des betroffenen Graders übergeben werden, damit der Auftrag vom Grading-Service korrekt weitergeleitet wird. Die erhaltenen Rückgabewerte werden vollständig übernommen und ausgegeben.

3.2.2 Statistiken

Die Statistiken werden, wie im Bereich der Schnittstellen beschrieben, in Zusammenarbeit des Grading-Services mit den einzelnen Grader erhoben. Je nach Benutzergruppe verfolgen die Statistiken unterschiedliche Absichten. Ein Student kann sich zu den eigenen bisherigen Abgaben Informationen anzeigen lassen. So kann er beispielsweise im Nachhinein nachvollziehen, welche Aufgabe ihm besondere Schwierigkeiten bereitet hat und gezielt dieses Themengebiet für eine folgende Klausur lernen. Auch kann mithilfe der Statistiken

ein Vergleich mit anderen Nutzern der Plattform entstehen, wenn die eigene Leistung mit dem Durchschnitt aller Abgaben verglichen wird.

Doch auch für die Gruppe der Lecturer bieten die Statistiken wichtige Informationen. Besonders ohne Vergleichsaufgaben aus vorherigen Semestern kann das Erstellen von Klausur- und Praktikumsaufgaben besonders in Hinsicht auf das Abschätzen des Schwierigkeitsgrads schwerfallen. Schon während des Semesters könnte eine ungefähre Einschätzung des Wissensstands der Kursteilnehmer dabei große Abhilfe leisten. Falls die freiwilligen Aufgaben bereits innerhalb des Semesters absolviert werden, kann die Auswertung der Aufgaben außerdem Missverständnisse aus der Vorlesung aufzeigen. Erzielt eine Mehrheit des Kurses in einem bestimmten Themengebiet schlechte Ergebnisse, könnte in der Vorlesung dieses Thema gezielt wiederholt werden. Eine Abschätzung der Kursstärke lässt sich an mehreren Indikatoren abschätzten.

Einfluss der Zeit

Bei jeder Abgabe wird das Startdatum und die verwendete Bearbeitungszeit gespeichert und kann aufschlussreiche Informationen über das Abgabeverhalten aufzeigen. Zwischen der ermittelten Zeit und der resultierenden Benotung lässt sich eine schwache Abhängigkeit feststellen [3]. Dabei führt eine größere Bearbeitungszeit zu schlechteren Noten. Laut Erik Froekjaer sei dies zwar im Allgemeinen vernachlässigbar gering, die Unterschiede werden jedoch bei Aufgaben mit besserer Durchschnittsnote und geringerer Bearbeitungsdauer größer. Diese beiden Aspekte ließen auf leichtere Aufgaben, die vermehrt in Routine bearbeitet werden können, schließen. Es wird außerdem vorgeschlagen, zusätzlich zur Zeit und zum Erfolg, noch eine Stimmungsabfrage nach der Bearbeitung durchzuführen. Die Zufriedenheit der Nutzer sei ein weiterer unabhängiger Aspekt neben der Effektivität und Effizienz. Eine solche Umfrage könnte in der weiteren Entwicklung der Korrekturkomponenten ergänzt werden.

Gegenteilig zur Geschwindigkeit der Bearbeitung von einzelnen Aufgaben, bei denen eine kürzere Bearbeitungszeit auf leicht bessere Noten schließen lässt, steigt die Wahrscheinlichkeit auf die Bestnote mit einer höheren Verweildauer im kursbegleitenden Onlineangebot [1]. Diese könnte entweder zwischen Betreten und Verlassen der Plattform erhoben werden oder als Summe der Zeiten aller Aufgaben beschrieben sein. Ersteres würde eine zusätzliche Messung benötigen und zählt zusätzlich auch inaktive Zeit hinzu, in der kein produktives Lernen stattfindet. Die zweite Möglichkeit lässt sich aus bestehenden Daten ermitteln und

kann in der Gesamtstatistik eines Students angezeigt werden. Zusätzlich zu der eigenen Anzeige kann wieder ein Vergleich zur Gesamtheit gezogen werden. Eine Einteilung der Nutzer in relevante Gruppen, die beispielsweise einer Veranstaltung zugeordnet werden, wäre notwendig, damit solche Vergleiche zwischen Studierenden einen Mehrwert bieten. So würde nur Zeit gemessen werden, die in für den Kurs relevante Aufgaben investiert wurde. Eine solche Einteilung ist allerdings im Prototypen nicht umgesetzt und würde vermehrt auch im Assignment-Service relevant, um Aufgaben ebenfalls nur für bestimmte Gruppen freizuschalten.

Auch aus der Uhrzeit, beziehungsweise dem Datum, beim Bearbeiten einer Aufgabe lassen sich Informationen für den Lecturer ziehen. Je später Aufgaben in der zulässigen Bearbeitungsperiode abgegeben werden, desto schlechter fallen die Noten aus [8]. Dies hat bei der derzeitig angestrebten Umsetzung OPPSEEs kaum Aussagekraft, da es sich nur um freiwillige Aufgaben ohne Abgabefrist handelt. Es entsteht daher kein Druck, noch kurz vor Schluss noch die Lösung einzureichen. Trotzdem ließe sich voraussichtlich eine schwankende Aktivität zu unterschiedlichen Zeiten des Semesters feststellen. Womöglich ließen sich auch Aufgaben, die thematisch zu aktuellen Themen der Vorlesung oder zur bestehenden Praktikumsaufgabe passen, mit einer Empfehlung versehen, wodurch diese Statistik bedeutender werden würde. Dann wäre der Zeitraum messbar, wie nah die Teilnehmer eine empfohlene Aufgabe begleitend zum Unterricht absolvieren.

Auch die Tageszeit der Abgaben hat Auswirkungen auf die Noten [8]. Aufgaben, die in der Nacht abgegeben werden, fallen durchschnittlich schlechter aus als Abgaben während anderer Tageszeiten. Dies könnte durch hohe Arbeitszeiten außerhalb der Hochschule, die eine zusätzliche Lernleistung nur spät Abends ermöglichen, zustande kommen. In den Statistiken lässt sich anzeigen, in welchen Tagesabschnitten eine bestimmte Aufgabe abgegeben wurde, bzw. zu welcher Zeit ein Student generell Lösungen einreicht. Die Statistiken zum Tages- und Semesterverlauf könnten visuell dargestellt werden, um in Form eines Diagramms leichter und schneller verstanden zu werden.

Die Zeitmessung muss zusätzlich in Bezug auf die durchschnittliche Benotung aller Abgaben gestellt werden. Fallen die Ergebnisse vermehrt schlecht aus, könnte dies auf eine zu kurze Bearbeitungszeit hindeuten. Ein Zeitmangel führt nicht automatisch zu unfairen Bewertungen, besonders wenn diese auf den Kursschnitt bezogen wird [12]. Trotzdem ist gerade in einer freiwilligen Lernplattform wie OPPSEE ein solch entstehender Zeitmangel nicht förderlich, wenn es hauptsächlich um den Lernerfolg statt um eine Bewertung geht. In den jeweiligen Gradern lassen sich weitere Beobachtungen im Bezug auf die

Bearbeitungszeit treffen und in der Statistik anzeigen. So kann bei einer Multiple-Choice-Aufgabe die durchschnittliche Bearbeitungsquote bei der letzten mit der ersten angezeigten Frage verglichen werden. Dies würde wieder auf eine zu geringe Bearbeitungszeit hindeuten, sodass die Nutzer nicht alle Aufgaben vollständig bearbeiten können. Diese Messung benötigt allerdings eine zusätzliche Zeitmessung, da diese bewusst in den Grading-Service ausgelagert wurde. Statistiken, die von den Gradern erstellt werden, lassen sich durch den HTML-Div-Block des Graders bei der Abfrage nach Statistiken, an den Grading-Service senden.

Einfluss von Versuchsanzahl

Da OPPSEE als ergänzende Übungsplattform dienen soll, aber Prüfungen oder Prüfungsvorleistungen nicht ersetzt, besteht zunächst keine Limitierung an möglichen Versuchen beim Lösen einer Aufgabe. Für bestimmte Aufgaben könnte allerdings wie in Szenario 3 die Ausführung auf nur einen Versuch beschränkt werden. Dies hat aber keine direkten Auswirkungen auf den Student, der nicht wie in einer richtigen Prüfungssituation nach mehreren missglückten Versuchen von der Klausur ausgeschlossen wird. Trotzdem gibt es einige Gründe dafür, die Anzahl an verwendeten Versuchen bei jeder Aufgabe zu zählen. Durch die Messung entsteht kein zusätzlicher Aufwand beim Implementieren von neuen Gradern, da die Anzahl an Aufgabenstarts schon im Grading-Service vermerkt werden kann.

Falls es in einem Online-Test zwei Möglichkeiten zum Bestehen gibt, schneiden Teilnehmer mit nur einem verwendeten Versuch besser ab, als jene, die beide Versuche nutzten [10]. Dies könnte laut Ryan K. Orchard unter anderem durch das taktische Verschenken des Erstversuchs, um sich durch das Bestreiten der Prüfung besser auf den zweiten Test vorzubereiten, zustande kommen. Eine Antwort auf dieses Problem wäre das Erstellen einer Durchschnittsnote aller Versuche. Da es bei der Note in der hier besprochenen Lernplattform allerdings nicht um das Bewerten der gesamten Leistung, sondern nur um eine Form von Rückmeldung an den Nutzer geht, ist diese mögliche Problematik nicht relevant. Außerdem gibt es beim freiwilligen Ansatz der Lernplattform keinen direkten Anreiz eine möglichst gute Endnote zu erzielen und dafür bei einem vorherigen Versuch absichtlich durchzufallen. Zusätzlich benötigen Nutzer mit besseren Vorkenntnissen keinen Zweitversuch, da diese mit ihrem ersten Ergebnis bereits zufrieden sein werden. Erzielt ein Kurs mit wenigen Versuchen durchweg gute Noten, kann dies bei sinnvollem Aufgabenniveau für ein gutes Verständnis sprechen. Aus dem Versuch von Orchard ist ebenfalls

die Messung des Notenunterschieds zwischen Erst- und Zweitversuch anwendbar. Hier erzielten 21,4% Zweitprüfungen ein schlechteres Ergebnis als in ihrer Erstprüfung. Diese Metrik kann auf den Versuch des Ratens der Antworten ohne Lerneffekt zwischen den Versuchen deuten. Eine Kombination mit der Zeitmessung wäre an dieser Stelle relevant und könnte die Zeitspanne zwischen Versuchen einer einzelnen Aufgabe anzeigen, um zu beobachten, ob vor einem erneuten Versuch an anderer Stelle gelernt wurde, oder sofort ein neuer Versuch gestartet wird.

Fazit Statistiken

Der Nutzen für den Lecturer beim Einsehen von Nutzer- und Aufgabenstatistiken liegt in der Bewertung des Kursniveaus und der Schwierigkeit der gestellten Aufgaben. Aufgrund von allgemein zugänglichen Informationen ohne direkten Graderbezug lassen sich dazu bereits in dieser Komponente erste Aussagen treffen. Weitere Metriken können direkt bei der Auswertung des jeweiligen Graders berechnet und in den Bericht integriert werden. Aufgrund der Erstellung von weiteren Metriken direkt im Grader können speziellere Aspekte betrachtet werden, die auch einzelne Testfälle auswerten, anstatt nur die Gesamtnote zu betrachten.

3.3 Aufbau der Grader

Die einzelnen Grader können sich grundsätzlich voneinander unterscheiden. Jeder Grader hat allerdings als Mindestbedingung, die im Grading-Service beschriebenen Funktionen und vordefinierten Schnittstellen zu bedienen. Folglich sollten Tests eingepflegt, bearbeitet, gelöscht und angezeigt werden können. Außerdem lassen sich Aufgaben abgeben und später zugehörige Statistiken auslesen. Der Grading-Service hat keine Informationen darüber, wie im jeweiligen Grader die Auswertung stattfindet. Im allgemeingültigen Datenfeld beim Abgeben der Aufgabenbearbeitung muss nur zwischen Grader und der VSC-Extension eine Struktur festgelegt werden. Der Grading-Service greift auf diese Daten nicht zu und benötigt daher kein Wissen über das verwendete Format. Im Folgenden besitzen die Grader nur in der Datenverarbeitung und Visualisierung der Ergebnisse Unterschiede.

3.3.1 Multiple-Choice-Grader

Der Multiple-Choice-Grader soll dem Student mehrere nach Themen sortierte Fragen stellen. Diese werden in der Entwicklungsumgebung durch die VSC Extension anstelle des Editorfensters in Form einer HTML-Datei angezeigt. Da der Student nebenbei keinen Programmcode erstellt, müssen auch keine Dateien des Nutzers verwaltet werden. Multiple-Choice-Aufgaben lassen sich mit unterschiedlichen Formaten umsetzen. Für den Nutzer ändert sich bei der Bearbeitung wenig, jedoch kann sich die Bewertung deutlich unterscheiden. Ein verbreitetes Verfahren ist „numbers right scoring“ [7]. Dabei werden alle richtigen Antworten summiert und ergeben im Verhältnis zur Anzahl der Fragen das Endergebnis. Teilnehmer können dabei ohne Punktabzug raten, auch wenn ihnen die Antwort unbekannt ist. Auch hier gilt, dass OPPSEE als Lernplattform dieses Lösungsvorgehen nicht zwingend verhindern muss. Eine Alternative, die regelmäßig im Falle von Multiple-Choice-Fragen in Klausuren an der HAW Hamburg verwendet wird, ist „rights minus wrongs“ [7]. Dabei werden falsche von den richtigen Antworten nach der Gleichung

$$S = R - \frac{W}{k-1} \tag{3.1}$$

abgezogen, mit S als Endpunktzahl, R als Anzahl korrekter Antworten, W als Anzahl Fehler und k als Antwortmöglichkeiten pro Frage.

Dieses Verfahren wird vom Multiple-Choice-Grader verwendet, es lassen sich allerdings auch weitere Methoden im selben Grader realisieren. So könnten im übertragenen Datenfeld der Beginn der Zeichenkette die verwendete Methodik der Aufgabe ankündigen. Der Grader würde dann das genannte Verfahren für zur weiteren Auswertung verwenden. Falls mehrere Bewertungsmethoden verwendet werden, müssen diese auch in der Aufgabenstellung genannt sein, weil sich dadurch das Antwortverhalten des Nutzers ändert.

Eine weiteres Verfahren wäre ein System, in dem auch falsche Aussagen explizit gekennzeichnet werden müssen [2]. Studierende können dann jede Antwortmöglichkeit mit wahr, falsch oder Enthaltung beantworten. Dieses Verfahren würde Missverständnisse bei Studierenden aufdecken, wenn beispielsweise widersprüchliche Aussagen beide als korrekt eingestuft werden oder eine von beiden ausgelassen wird. Dieses Format fördert ebenfalls das Raten von Antworten, wodurch auch hier Bewertungsmethoden, die das Raten verhindern, notwendig sind.

Aufbau

Jeder Grader muss seine eigenen Testfälle verwalten. Beim Multiple-Choice-Grader ist jede Frage ein einzelner Test, der einer Aufgabe zugeordnet wird. In der Datenbank werden für Tests die zugehörige Aufgabe, die Anzahl an Antwortmöglichkeiten, sowie die korrekten Antworten gespeichert. Für den Zugriff auf die Tests werden die gängigen Datenbankzugriffe als Schnittstelle bereitgestellt. Es stehen mit Erstellung, Löschung, Bearbeitung und Anzeige der Daten die gleichen Funktionen zur Verfügung, die vom Grading-Service ebenfalls abgerufen werden.

Zusätzlich werden nach einer Abgabe, in einer weiteren Tabelle der Datenbank, Informationen über das erhaltene Ergebnis gespeichert. Das Merken der gegebenen Antworten eines Nutzers, sowie der erhaltenen Punktzahl, dient der Aufbereitung für Statistiken. Damit eine Abgabe eingereicht werden kann, muss auch der Multiple-Choice-Grader auf diese Anfrage reagieren. Dies geschieht mit demselben Aufbau wie beim Grading-Service. Einem Nutzer und einer Aufgabe wird ein Datenfeld zugeordnet, dass im Grader ausgewertet wird. Der String mit den Daten des Nutzers enthält mehrere aufeinanderfolgende Fragen, die durch einen Seperator getrennt werden. Jeder dieser Blöcke besteht aus einer eindeutigen Fragen-ID und allen vom Student gegebenen Antworten. Mithilfe der ID werden die korrekten Ergebnisse aus der Datenbank geladen und mit den Nutzerantworten verglichen. Dabei werden die korrekten und falschen Antworten gezählt und gemäß Gleichung (3.1) ausgewertet. Die ermittelte Punktzahl wird in eine Note umgerechnet und zurückgesendet. Nach der Auswertung wird ein HTML-Div-Block aus einer Vorlage generiert und dort sämtliche Fragen mit gegebenen und korrekten Antworten aufgelistet. Die dort verwendeten Daten, wie die Aufgabenstellung und die Antwortmöglichkeiten, müssen vom Assignment Service abgefragt werden. Die HTML-Datei und die Note werden daraufhin zurückgegeben.

Ähnliche HTML-Blöcke müssen auch bei Anfragen zu Nutzer- und Aufgabenstatistiken bereitgestellt werden. Allerdings können manche Grader dazu keine Informationen beitragen, weswegen dann HTTP-Statuscode 204 *no Content* zurückgegeben wird. Dies tritt beim Multiple-Choice-Grader bei den Nutzerstatistiken auf. Der Statuscode ist wichtig, damit der Grading-Service weiß, dass keine Daten zu erwarten sind.

Statistiken

Zusätzlich zu den allgemeingültigen Statistiken, die bereits im Grading-Service für alle Aufgabentypen erhoben werden, können in den Gradern speziellere Informationen gesammelt werden. Diese lassen sich zum einen in den Grading-Report direkt bei der Aufgabenabgabe integrieren, zum anderen bei der gezielten Abfrage von Nutzer- und Aufgabenstatistiken bereitstellen.

Für einen Multiple-Choice-Test mit großer Teilnehmerzahl, bei dem eine vorherige Abschätzung der Schwierigkeit notwendig ist, kann vor dem offiziellen Klausurtermin eine Probe mit ähnlichem Teilnehmerniveau durchgeführt werden, um eine Prüfung zu simulieren [11]. Dies ist allerdings bei den Onlinetests von OPPSEE aus mehreren Gründen nicht notwendig oder umsetzbar. Es ist aufgrund der fehlenden Relevanz der erhaltenen Note nicht wichtig genug, zu einfache oder zu schwere Fragen bereits vor einer ersten Prüfung zu filtern. Der Aufwand eine ausreichend große Probandengruppe zu finden, steht daher nicht im Verhältnis zum Nutzen für die Plattform.

Stattdessen lässt sich, nachdem die ersten Testergebnisse eingegangen sind, eine Bewertung anhand der erhaltenen Resultate vollziehen. Als erster sinnvoller Ansatz zur Bestimmung der Schwierigkeit ergibt sich das Auszählen der korrekten Antworten von jeder Frage der angefragten Aufgabe. Im Verhältnis zur Anzahl der Versuche ergibt sich die Quote an richtigen Lösungen. Vergleicht man die einzelnen Fragen der Aufgabe, können Tendenzen beim Aufgabenniveau und Verständnis der Teilnehmer erkannt werden. Diese Auszählung wirkt ergänzend zur vorherigen Messung im Grading-Service, in der die Durchschnittsnote ermittelt worden ist. Dadurch lassen sich Unterschiede innerhalb eines Themas feiner beobachten. Als Äquivalent zur Note können außerdem Durchschnittspunkte aller Fragen in der Statistik gezeigt werden. Diese lassen sich nicht vom Grading-Service ermitteln, können aber bei Verwendung mehrerer Grader gleichzeitig einen Mehrwert bieten. Fällt beispielsweise die Bewertung eines Graders auffällig schlecht aus, kann eine sehr tiefe Note entstehen, obwohl möglicherweise beim zweiten Grader eine gute Punktzahl erzielt worden ist.

Laut Ulrike Pado können Fragen auch über die Varianz der Antworten untereinander verglichen werden [11]. Im dort durchgeführten Experiment ließen sich frei formulierbare Antworten abgeben. Mithilfe von *Greedy String Tiling* wird geprüft, wie stark sich die Texte der Teilnehmer unterscheiden. Je größer die Ähnlichkeit bei einer Frage ist, desto leichter sei den Probanden die Bearbeitung gefallen. Dies lässt sich darauf zurückführen,

dass bei komplexeren Aufgaben, bei denen die Lösung nicht offensichtlich erscheint, eine größere Varianz an Antworten entsteht. Ist die Aufgabe hingegen trivial, wird immer ähnlich geantwortet. Das gleiche Konzept lässt sich auch für das „Negative Marking“ Multiple-Choice-Verfahren, das in dieser Komponente genutzt wird, anwenden. Anstelle des Greedy String Tiling wird für die Statistik des Graders die Anzahl an gleichen Antworten für jede Frage gemessen. Beim Abfragen der Aufgabenstatistik werden aus der Datenbank alle Antworten der gewünschten Aufgabe gesammelt und verglichen, wie oft die Antworten übereinstimmen. Dieser Wert bietet einen weiteren Bewertungsaspekt für die Schwierigkeit einer Aufgabe, es findet aber weiterhin keine automatische Aussortierung von zu leichten oder zu schweren Aufgaben statt. Diese Statistik bietet durch die hohe Zahl an Antwortmöglichkeiten bei Multiple-Choice-Fragen einen zusätzlichen Informationsgewinn im Vergleich zur Durchschnittspunktzahl. Werden alle Antwortmöglichkeiten gleichmäßig gewählt, liegt wahrscheinlich eine schwierigere Frage vor, bei der jede Antwort möglich wäre. Eventuell gibt es trotz nur einer korrekten Antwort von fünf Antwortmöglichkeiten auch nur drei häufig gewählte Lösungen. Dies ließe daraus schließen, dass zwei Antwortmöglichkeiten offensichtlich falsch waren. Außerdem werden bei dieser Messung im Gegensatz zur Erhebung der Gesamtpunktzahl auch die Teilfragen bewertet und nicht nur die komplette Fragensammlung.

In den Nutzerstatistiken kann äquivalent zu den Aufgabenstatistiken eine Durchschnittspunktzahl dargestellt werden. Aus den gleichen Gründen lässt sich dadurch zusätzlich zur Durchschnittsnote eine weitere Metrik betrachten, die ohne Einfluss von anderen Gradern auswertbar ist.

3.3.2 JUnit-Grader

Der JUnit-Grader ermöglicht das Testen von Javaprogrammen. Startet ein Student eine Java Aufgabe, werden vom Assignment Service die dafür benötigten Dateien geladen. Dies kann wie in den JUnit-Szenarien 1 und 2 nur eine einzelne Datei sein, oder wie bei der UML Modellierung ein ganzes Projekt mit mehreren Klassen. Generell werden nur Dateien geladen, die der Nutzer sehen soll, um mit ihnen zu interagieren. Dadurch wird die Anforderung nach Lesezugriff indirekt behandelt. Sollte ein Nutzer auf einer Datei keine Schreibrechte haben (wie in Szenario 4), werden diese Dateien bei der Abgabe nicht übermittelt. Für den Grader sieht es so aus, als hätte der Student nichts bearbeitet. Gibt der Student die Aufgabe ab, wird sein Fortschritt mithilfe des Datenfelds über den Grading-Service zum JUnit-Grader gesendet. Damit dies nicht zu Missverständnissen

führt, muss dem Studierenden angezeigt werden, welche Dateien für die Aufgabe relevant sind.

Der JUnit-Grader folgt dem Aufbau des Multiple-Choice-Graders. Es werden in der Datenbank nur die benötigten JUnit-Tests, sowie die ermittelten Ergebnisse der Nutzer gespeichert. Zusätzlich wurde im Prototypen die Benennung der Tests hinterlegt, um bei der Erstellung des Bewertungsbogens vereinfacht die Beschreibungen der Testfälle wiederzugeben. Diese müssten vom Assignment Service abgefragt werden. Zu jeder Aufgabe lassen sich beliebig viele Tests hinzufügen, die auf den abgegebenen Code ausgeführt werden. Die für den Benutzer relevanten Informationen sind dabei, ob der jeweilige Test bestanden wurde, und im Falle eines Fehlers, die erhaltene Fehlermeldung. Da der Sicherheitsaspekt in diesem Prototypen bewusst ausgelassen wird, kann die übergebene Lösung ohne Überprüfung ausgeführt werden.

Die Schnittstellen sind in allen Gradern gleich, die Funktionen zur Interaktion mit den Testfällen und zum Abrufen von Statistiken funktioniert daher genau wie beim Multiple-Choice-Grader.

Statistiken

Bei den Statistiken zu einzelnen Aufgaben kann ein ähnliches, aber vereinfachtes Verfahren wie im Multiple-Choice-Grader angewendet werden. Da die jeweiligen Testfälle immer nur eine binäre Antwort, bestanden oder durchgefallen, liefern, ist die Berechnung der Verteilung an erfolgreichen Tests allerdings nicht notwendig. Ein prozentualer Wert, der beschreibt wie viele Teilnehmer den jeweiligen Testfall erfüllt haben, reicht zum Bemerken von Auffälligkeiten aus. So könnte im Summenformel Beispiel aus der Anforderungsanalyse der Grenzwerttest deutlich häufiger fehlschlagen, als die anderen Testfälle. Dies führt zu einer genaueren Analyse der Fehler, als nur die Durchschnittsnote zu betrachten.

3.3.3 Lines-of-Code-Grader

Der Lines-of-Code-Grader des Prototypen stellt den einfachsten Grader dar. Er besitzt die gleichen Grundfunktionen wie die beiden vorher beschriebenen Komponenten. In einer eigenen Datenbank sind die Bedingungen zu den jeweiligen Aufgaben hinterlegt (siehe Komponentendiagramm Abbildung 3.1). Das Grundprinzip dieses Graders lautet die Länge einer gegebenen Datei zu kontrollieren. Dies ist eine absichtlich triviale Funktion,

die beispielhaft das Zusammenspiel mehrerer Grader innerhalb einer Aufgabe symbolisieren soll. Im Kontext des vollständigen Prototypen existiert daher die Aufgabe eine bestimmte Funktion mit maximal zehn Zeilen Code zu schreiben. Dabei wird sowohl der JUnit- als auch der Lines-of-Code-Grader verwendet. Die Verwaltungsarbeit liegt dabei vollständig auf Seiten des Grading-Services, der an beide beteiligten Komponenten die Aufgabe verteilen muss und die Ergebnisse einsammelt, um einen zusammenhängenden Grading-Report zu erstellen. Aus Sicht des Lines-of-Code-Graders verändert sich die Zusammenarbeit der Dienste nicht und es wird weiterhin nur die eigene Funktion erfüllt.

Aufgrund der geringen Komplexität hat diese Komponente eine Besonderheit bei der Benotung. Wird die geforderte Maximallänge von zehn Zeilen überschritten, wird die schlechtest mögliche Note zurückgegeben. In diesem Fall wird vom Grading-Service die komplette Aufgabe als nicht bestanden gewertet. Andererseits gibt es bei Einhaltung der Maximallänge die Bestnote, wodurch es bei Abgabe einer ausreichen kurzen Lösung, die womöglich sogar leer ist, zu einer Inflation der Note kommt. Dieses Problem könnte in Zukunft durch eine weitere Form der Notengebung gelöst werden, die nur zwei Zustände kennt. Eine gesonderte Auswertung des Grading-Services wäre notwendig, die Note würde dann zum Beispiel nur noch aus dem richtigen Bestandteil der Aufgabe, den JUnit-Testfällen bestehen. Der Lines-of-Code-Grader würde wie gewünscht nur zum sofortigen Nichtbestehen führen, falls die Regeln missachtet wurden.

Auch in diesem Grader werden die Tests, wie in den anderen Gradern, gespeichert und verwaltet. Es existieren die gleichen Schnittstellen, um Statistiken abzurufen. Da dieser Test aber nahezu keine weiteren Informationen sammeln kann, wird für beide Statistiktypen nur der 204 *no Content* Statuscode zurückgegeben.

3.4 VSC Extension

Um in diesem Prototypen den vollständigen Ablauf der Szenarien aus der Anforderungsanalyse darzustellen, fehlt noch die Schnittstelle zum Nutzer der Plattform. Diese soll gemäß der Vorgaben zur Umsetzung durch eine VSC Extension realisiert werden. Über diese müssen die beiden Nutzergruppen sämtliche Funktionen des Grading-Services aufrufen können. Die Gruppe Student kann Aufgaben starten, diese bearbeiten und anschließend abgeben. Daraufhin erhält der Nutzer den vom Grading-Service erstellten Bewertungsbogen. Dieser Vorgang lässt sich beliebig oft wiederholen. Der Lecturer kann

Statistiken von Nutzern und Aufgaben abrufen. Außerdem soll er sowohl Aufgabenstellungen und zugehörige Tests bearbeiten können. Im Prototypen wird an dieser Stelle nur die Aufgabenstatistik umgesetzt. Zum Anzeigen von Nutzerstatistiken müsste lediglich ein weiterer Aufruf an den Grading-Service gesendet werden. Ebenfalls gibt es in der VSC Extension, wie im gesamten Prototypen keine explizite Trennung der Nutzerrollen durch Rechtevergabe. Daher gibt es nur eine Extension, welche die Funktionen von beiden Rollen miteinander vereint.

3.4.1 Aufbau

Die VSC API bietet verschiedene Möglichkeiten, um dem Nutzer Menüs und grafische Darstellungen anzuzeigen. Als Vorlage für Designentscheidungen dient die VSC Extension einer vorherigen OPPSEE Version [5].

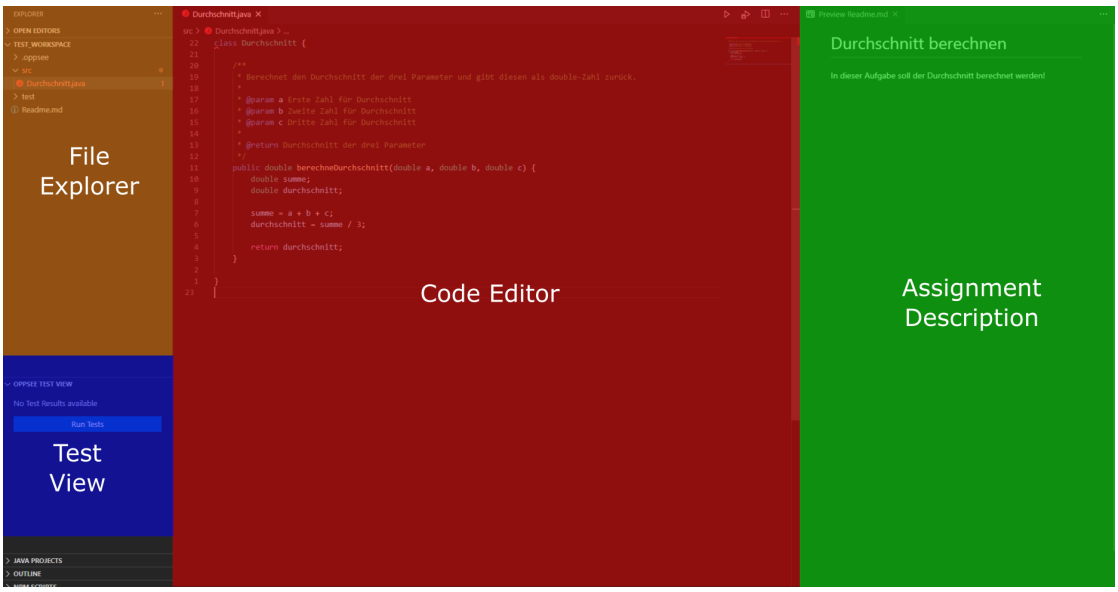


Abbildung 3.2: VSC Extension Design[5]

Die Anzeige der Testfälle ist mithilfe einer VSC TreeView [9] als aufklappbares Menü unterhalb des Dateiexplorers umgesetzt. Hierüber läuft in der Extension dieses Prototypen jegliche Kommunikation zum restlichen System OPPSEEs. Die ebenfalls in der Grafik enthaltene Aufgabenbeschreibung existiert hier im Prototypen nicht und müsste vom Assignment Service abgerufen werden. Es gäbe außerdem die Möglichkeit WebViews aus

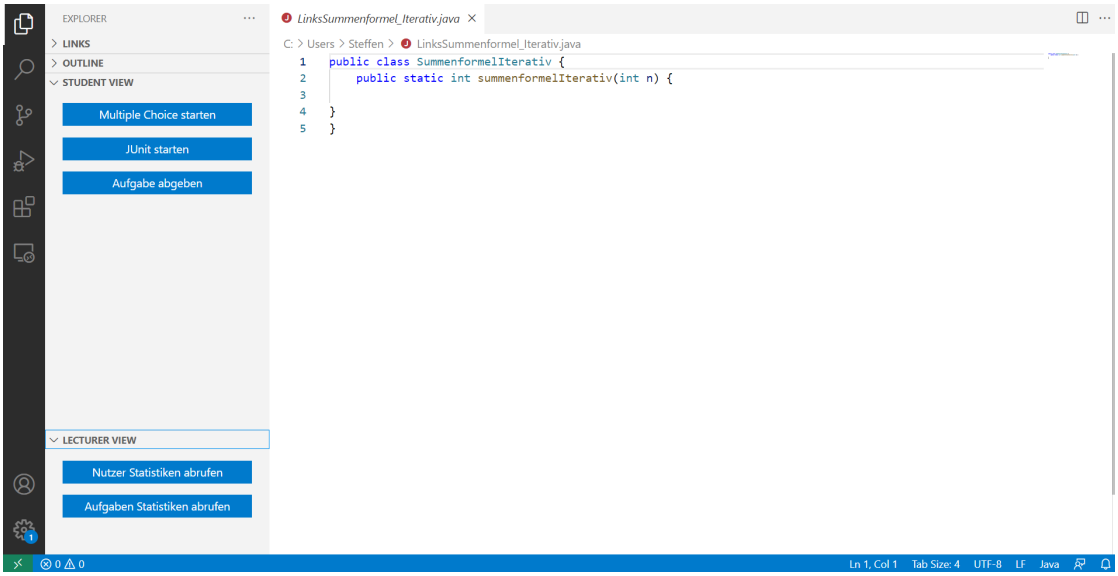


Abbildung 3.3: VSC Extension, Java Aufgabe gestartet

der Visual Studio Code API zum Starten der Tests und Abgabe der bearbeiteten Aufgabe zu benutzen. VSC WebViews ermöglichen durch das Anzeigen von HTML-Inhalten nahezu freie Gestaltungsmöglichkeiten. Allerdings ist die gewünschte Position im Editor in der Spalte des Dateiexplorers nicht nutzbar für WebViews, wodurch die Bedienung anstelle des Quellcodes dargestellt werden müsste. Im Gegensatz dazu ermöglicht eine TreeView die Nutzung dieser Position neben der Bearbeitung im Editor.

3.4.2 Student View

In der Student View sind alle Funktionen enthalten, um einen Zyklus aus der Anforderungsanalyse durchzuführen. Es kann zwischen einer Java und einer Multiple-Choice-Aufgabe gewählt werden. In der Java Aufgabe werden der JUnit- und der Lines-of-Code-Grader verwendet. Sobald eine Wahl getroffen wurde, sendet die Extension eine Nachricht an OPPSEE und signalisiert den Beginn dieser Aufgabe. Dies ist relevant für die Zeitmessung des Grading-Services, hat ansonsten aber keine Konsequenzen für den Nutzer. Gleichzeitig werden über die gleiche Schnittstelle relevante Informationen, wie zum Beispiel vorgegebener Quellcode oder die Multiple-Choice-Fragen, abgerufen. Im Falle einer vorherigen Abgabe muss eventuell ein alter Stand geladen werden, die Persistenz der persönlichen Daten liegt jedoch nicht im Aufgabenbereich des Grading-Systems. In anderen Aufgaben

könnten auch ganze Projekte geladen werden, falls dies für die Bearbeitung notwendig und vom Aufgabensteller gewollt ist. Zugriff auf die Daten erhält der Student über das normale VSC-Dateisystem. Anschließend gibt der Nutzer die Aufgabe über einen Button in der TreeView ab, wodurch die geöffneten Dateien versendet und gelöscht werden. Ein weiteres Arbeiten an der Aufgabe ist danach nicht mehr möglich. In Abbildung 3.3 ist die gestartete Java Aufgabe dargestellt. Links in der Student View, befindet sich der Knopf zum Abgeben der Datei. Vorher kann der Code beliebig im Editor bearbeitet werden.

Als Rückgabe erhält die Erweiterung die vom Grading-Service erstellte HTML-Datei mit Informationen zur Bewertung. Aufgrund der Unterstützung von HTML durch Web-Views innerhalb der VSC API bietet sich diese Art von Datenaustausch an, es entsteht daher kein weiterer Aufwand zum Anzeigen des Ergebnisses. Die angezeigte Datei wird dementsprechend auch nicht mehr innerhalb dieser Komponente editiert. Das Festlegen in Designfragen entsteht bereits beim Erstellen der HTML-Vorlage im Grading-Service und im jeweiligen Grader.

Anstatt eine Aufgabe zu starten, kann der Student auch im selben Menü Statistiken anzeigen lassen. Dies wären zum einen die Nutzerstatistiken, allerdings beschränkt auf Daten über den eigenen Account, sowie eine erneute Anzeige zu den gelösten Aufgaben. Statistiken haben für diese Nutzergruppe einen andere Zweck, beispielsweise als Selbstkontrolle und zum Verfolgen des eigenen Lernfortschritts. Auch ein Vergleich zu Durchschnittswerten der anderen Nutzer könnte dem jeweiligen Student seinen Wissensstand aufzeigen.

3.4.3 Lecturer View

Die Lecturer View würde auch bei einer Trennung von der Student View dem gleichen Aufbau folgen. Die Interaktion mit OPPSEE läuft über eine TreeView unter der Dateiauswahl ab und die angefragten Statistiken werden unverändert als HTML mithilfe einer WebView angezeigt. Der Lecturer soll sämtliche Statistiken zu den Aufgaben und Nutzern abrufen können.

4 Zusammenfassung

4.1 Fazit

Das Ziel der Thesis war die Erstellung einer Architektur für die Korrekturkomponenten einer Online-Programmierplattform in Form von Microservices. Anschließend sollte eine prototypische Umsetzung der Architektur erfolgen. Mithilfe von einer Anforderungsanalyse, die vier verschiedene Szenarien beschreibt, wurden Anforderungen an das geplante System erstellt. Im Prototypen sind die entworfenen Komponenten umgesetzt, um die Durchführbarkeit der geplanten Konzepte zu zeigen. Gezeigt wird die Nutzerinteraktion mit der VSC Extension und die folgende Kommunikation zwischen den Komponenten der Korrekturkomponenten bis zur abschließenden Anzeige der Testergebnisse.

Die grundlegenden Anforderungen wurden umgesetzt, sodass sich im Prototypen das JUnit-Szenario abspielen lässt. Dabei startet der Student eine Aufgabe in der Entwicklungsumgebung, bearbeitet diese und reicht die Lösung anschließend zur Korrektur ein. Über den Grading-Service wird der Quellcode bis zum JUnit-Grader weitergereicht und dort ausgewertet. Funktionen, wie das Erstellen eines Bewertungsbogens, die Messung der Bearbeitungszeit und das Zählen der Versuche, werden bereits im Grading-Service vollzogen, um Codeduplikation zwischen den Gradern zu reduzieren.

Aufgaben, wie beispielsweise die Java-Aufgabe im zweiten Szenario, können mehrere Grader gleichzeitig verwenden. Der erstellte Quellcode wird dabei an den JUnit- und den Lines-of-Code-Grader gesendet. Daher kann die endgültige Note einer Aufgabe erst im Grading-Service, und nicht bereits im genutzten Grader bestimmt werden.

Durch die gewählte Microservice Architektur wurden viele Funktionen der Grader in den Grading-Service ausgelagert. Dadurch ließen sich nach einer einmaligen Erstellung eines ersten Graders und des Grading-Services, weitere Grader leicht integrieren.

4.2 Ausblick

Durch die leichte Integration von neuen Gradern, lassen sich komplexere Grader, wie zum Beispiel für die Bewertung von UML-Klassendiagrammen aus Szenario 4 der Anforderungsanalyse, entwerfen. Die bisherigen Grader sind teilweise einfach gehalten und messen nur die Länge einer abgegebenen Datei, damit die Architektur des Gesamtsystems im Vordergrund steht.

Die bisherigen Grader können auch durch zusätzliche Funktionalitäten erweitert werden, wodurch sich mehr Anwendungsmöglichkeiten ergeben. So könnten im Multiple-Choice-Grader verschiedene Auswertungsmethoden umgesetzt werden. Auch die Problematik im ersten Szenario, dass bei der Auswertung nicht die Verwendung von einer Rekursion überprüft werden kann, ließe sich in Zukunft mit einer *Static Code Analysis* entfernen.

Nutzerrollen wurden zwar für die Funktionalitäten betrachtet, existieren allerdings im Prototypen nicht. Es gibt in den Schnittstellen keine Zugriffsrechte oder Prüfungen für eine Rollenzugehörigkeit. Würde OPPSEE in dieser Form den Studenten bereitgestellt werden, könnte jeder über die Schnittstellen der Lecturer alle Daten auslesen oder Aufgaben und Tests bearbeiten. Generell wurden Sicherheitsaspekte bisher komplett ignoriert und müssen in zukünftigen Architekturen beachtet werden.

Die Anforderung, bei einer Aufgabenänderung alle bisherigen Teilnehmer zu benachrichtigen, muss noch implementiert werden. Die Nutzerkennungen der betroffenen Nutzer wären verfügbar, es fehlt allerdings ein Benachrichtigungssystem, dass generell Informationen über neu hinzugefügte und freigeschaltete Aufgaben bietet.

Derzeit gibt es nur die Möglichkeit, Daten als HTML-Datei in der VSC Extension anzeigen zu lassen. Sinnvoll wäre es, zusätzlich die Daten maschinenlesbar als CSV-Datei zu exportieren, damit zusätzliche Auswertungen von Kursbetreuern und Plattformbetreibern leichter vollzogen werden können.

Literaturverzeichnis

[1] Carver, Lin B. ; Mukherjee, Keya ; Lucio, Robert: Relationship Between Grades Earned and Time in Online Courses. In: *Online Learning* 21 (2017), Nr. 4, S. 303–313

[2] Couch, Brian A. ; Hubbard, Joanna K. ; Brassil, Chad E.: Multiple–True–False Questions Reveal the Limits of the Multiple–Choice Format for Detecting Students with Incomplete Understandings. In: *BioScience* 68 (2018), 05, Nr. 6, S. 455–463. – URL https://doi.org/10.1093/biosci/biy037. – ISSN 0006-3568

[3] Frøkjær, Erik ; Hertzum, Morten ; Hornbæk, Kasper: Measuring Usability: Are Effectiveness, Efficiency, and Satisfaction Really Correlated? In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA : Association for Computing Machinery, 2000 (CHI '00), S. 345–352. – URL https://doi.org/10.1145/332040.332455. – ISBN 1581132166

[4] Gandrass, Niels ; Hinrichs, Torge ; Schmolitzky, Axel: Towards an Online Programming Platform Complementing Software Engineering Education. In: *SEUH*, 2020, S. 27–35

[5] Hinrichs, Torge ; Burau, Henri ; Pilgrim, Jens von ; Schmolitzky, Axel: A Scaleable Online Programming Platform for Software Engineering Education. (2021)

[6] Indrasiri, Kasun ; Siriwardena, Prabath: Microservices for the enterprise. In: *Apress, Berkeley* (2018)

[7] Kurz, Terri B.: A Review of Scoring Algorithms for Multiple-Choice Tests. (1999)

[8] Metz, Anneke M.: The effect of access time on online quiz performance in large biology lecture courses. In: *Biochemistry and Molecular Biology Education* 36 (2008), Nr. 3, S. 196–202

[9] Microsoft: *Visual Studio Code Extension API*. – URL https://code.visualstudio.com/api. – Zugriffsdatum: 2021-07-24

[10] Orchard, Ryan K.: Multiple attempts for online assessments in an operations management course: An exploration. In: *Journal of Education for Business* 91 (2016), Nr. 8, S. 427–433

[11] Padó, Ulrike: Question Difficulty – How to Estimate Without Norming, How to Use for Automated Grading. In: *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*. Copenhagen, Denmark : Association for Computational Linguistics, September 2017, S. 1–10. – URL https://www.aclweb.org/anthology/W17-5001

[12] Verleger, Matthew A. ; Beach, Daytona: Just Five More Minutes: The Relationship Between Timed and Untimed Performance on an Introductory Programming Exam. In: *2016 ASEE Annu. Conf. & Exposition*, 2016

A Anhang

Der Anhang befindet sich zusätzlich zur Thesis auf der beigelegten CD.

A.1 Inhalt der CD

1. Grading-Service
2. JUnit-Grader
3. Lines-of-Code-Grader
4. Multiple-Choice-Grader
5. OPPSEE
6. VSC Extension
7. docker-compose.yml

In der VSC Extension sind zur Verhinderung von zukünftigen Kompatibilitätsproblemen alle Node-modules mit enthalten.

A.2 Installationshinweise

Für die Installation und Ausführung des Prototyps werden Visual Studio Code, Node.js und Docker benötigt. Getestet wurde mit Visual Studio Code 1.58.2, Node.js v14.17.0 und Docker 20.10.6.

1. Zum Rootverzeichnis mit docker-compose.yaml navigieren
2. docker-compose up –build -d

3. cd vsc-extension
4. code .
5. In Visual Studio Code F5
6. Einen beliebigen Ordner im Visual Studio Code Debugger öffnen
7. Student View in der TreeView öffnen
8. JUnit Aufgabe starten, startet Timer
9. Aufgabe abgeben

Das Öffnen eines Ordners in Schritt 7 wird benötigt, damit Visual Studio Code die empfangene Datei zwischenspeichern kann. Nach der Abgabe wird die Datei wieder gelöscht.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original