HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

# Moodle Plugin StudentQuiz

# Bachelorarbeit

**Abteilung Informatik**
**Hochschule für Technik Rapperswil**

## Herbstsemester 2018


| | |
|---|---|
| **Autor** | Philipp Albrecht |
| **Betreuer** | Prof. Frank Koch |
| **Experte** | Stephan Meier |
| **Gegenleser** | Thomas Corbat |

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Glossar

**apache2** ist einer der meistbenutzten Webserver im Internet in der Version 2.

**Array** beschreibt den Datentyp »Feld« und kann auch als Liste mit Index bezeichnet werden.

**Attempt** Im Kontext dieser Bachelorarbeit steht Attempt für einen Fragedurchlauf Versuch.

**Branch** Ein Entwicklungszweig in GIT.

**Button** Bezeichnet eine Schaltfläche und ist Steuerelement in grafischen Benutzeroberflächen.

**Commit** ist ein Ausdruck aus der Softwaretechnik und beschreibt die Freischaltung einer Änderung.

**Continuous Integration** steht für kontinuierliche Integration und bezeichnet das fortlaufende Zusammenfügen von Komponenten einer Anwendung mit laufenden Tests.

**Difficulty Bar** bezeichnet eine Grafik, die die durschnittliche und eigene Schwierigkeit darstellt.

**Durchlauf** Bezeichnet im Kontext dieser Arbeit das Durchspielen eines Quizzes.

**Execution Plan** Englisch für Ausführungsplan oder Auswertungsplan. Beschreibt in welchen Einzelschritten in einem relationalen Datenbankmanagementsystem eine Datenbankabfrage ausgeführt wird.

**Feld** Bezeichnet im Kontext dieser Arbeit häufig ein Datenfeld (zum Beispiel ein Datenfeld in einer Tabelle).

**Frage Typ** bezeichnet den Typ einer Frage (engl. Question) in Moodle. Beispiel: Fragetyp Mehrfachauswahl, FalschRichtig, usw.

**GDPR** ist eine Englische Abkürzung für General Data Protection Regulation und bezeichnet die Datenschutz-Grundverordnung der EU.

**HTML** steht für Hypertext Markup Language und ist eine textbasierte Auszeichnungssprache zur Strukturierung von Texten und wird im World Wide Web von Webbrowsern dargestellt.

**ID** steht für Identifikator und dient im Kontext dieser Arbeit zur eindeutigen Identifizierung von Objekten.

**IDE** Eine IDE ist eine Integrierte Entwicklungsumgebung und fasst alle für die Programmierung relevanten Werkzeuge in einer Applikation zusammen. Zentral sind Syntax Highlighting und Debugging Werkzeuge.

**Insert** Ein SQL Befehl um einen neuen Eintrag in der Datenbanktabelle anzulegen.

**JSON** steht für JavaScript Object Notation und ist ein Format für ein kompaktes Datenformat.

**Klasse** ,auch Objekttyp genannt, bezeichnet in der objektorientierten Programmierung ein abstraktes Modell für eine Reihe ähnliche Objekte.

**Mobile App** bezeichnet die Mobile Applikation für Smartphones und Tablets.

**Mobile Support** Beschreibung, ob Applikationen oder Plugins von mobilen Endgeräten wie Smartphones oder Tablets unterstützt werden.

**Moodle** Moodle steht für Modular Object-Oriented Dynamic Learning Environment und ist ein freies objektorientiertes Kursmanagementsystem, eine Lernplatform.

**Moodle Core** Bezeichnet den Kern der Moodle Web Version mit allen Standardkomponenten ohne installierte Plugins.

**Moodle Mobile** Die Mobile Version von Moodle für Smartphones.

**MySQL** ist ein relationales Datenbankverwaltungssystem und wird von der Oracle Corporation entwickelt.

**Performance** Bezeichnet das Leistungsverhalten der Rechenleistung.

**PHP** ist eine Skriptsprache die hauptsächlich zur Erstellung von dynamischen Webseiten verwendet wird.

**PhpStorm** ist eine Software zur verteilten Versionsverwaltung von Dateien.

**PhpStorm** Eine Entwicklungsumgebung (IDE) von JetBrains speziell für PHP.

**Plugin** bezeichnet ein Zusatzmodul oder eine Software-Erweiterung für eine bestehende Software.

**Push Request** bezeichnet in GIT einen Änderungsvorschlag der vom Maintainer angenommen oder abgelehnt werden kann.

**Query Log** bezeichnet die Protokollierung von SQL Querys.

**Question** steht englisch für Frage und steht im Kontext dieser Arbeit häufig für eine Frage innerhalb eines Quizzes.

**Question Type** Siehe Frage Typ.

**Quiz** ist englisch für Fragespiel.

**Redmine** ist eine Projekt Management Web Applikation.

**Rendern** bezeichnet das Erstellen einer Grafik aus Rohdaten.

**SE2** Ein Modul an der HSR mit dem Namen Software Engineering 2.

**Source Code** Englisch für Quellcode. Bezeichnet den Menschen lesbaren geschrieben Text eines Computerprogrammes.

**Star Bar** bezeichnet eine Grafik, die die durschnittliche und eigene Bewertung darstellt..

**SVG** steht für Scalable Vector Graphics und ist eine Spezifikation zur Beschreibung zweidimensionaler Vektorgrafiken.

**VirtualBox** ist eine Virtualisierungssoftware des Unternehmens Oracle.

# 1 Aufgabenstellung

Die folgende Aufgabenstellung wurde zu Beginn dieser Bachelorarbeit formuliert.

# Aufgabenstellung Bachelor Thesis „Moodle Plugin StudentQuiz“

## 1. Ausgangslage, Problembeschreibung

Studierende schätzen Trainings-Prüfungen zur Vorbereitung auf summative Assessments. Aus diesem Grunde entwickelte die Hochschule für Technik in Rapperswil das Moodle Plugin StudentQuiz. Mit StudentQuiz können Studierenden eigene Fragen erstellen und in einem Pool miteinander teilen. Auch wenn der Beitrag einzelner Studierender nur klein ist, entstehen bei grösseren Gruppen schnell beachtliche Fragen-Sammlungen.

Die gesammelten Fragen können in StudentQuiz nach einer Vielzahl von Kriterien zu einem Quiz gebündelt werden. Beim Durchspielen der Quizzes können die Studierenden die Fragen kommentieren und bewerten. Aus den Nutzungsdaten ermittelt StudentQuiz die Qualität der Fragen und honoriert Studierenden Punkte für Beiträge und richtige Antworten. Eine Personal Learning Assistance visualisiert den individuellen Fortschritt und vergleicht diesen mit der Community. Die erstellten Fragen können zudem in weiteren Moodle-Tests recycliert werden.

StudentQuiz wurde zuletzt von der Diplomanden-Gruppe Denis Manente und Simon Schäfer entwickelt und aktuell von circa 350 Schulen eingesetzt, natürlich auch an der HSR. Als Ausgangsbasis für diese Arbeit steht das Plugin in der Version 3.1.1 auf GitHub zur Verfügung.

## 2. Aufgabenstellung

Es handelt sich um eine GPLv3 Entwicklung im Umfeld von PHP und JavaScript für Moodle ab Version 3.4. Auf der inhaltlichen Seite geht es um kollaboratives Peer-Quizzing als innovative Lern- und Prüfungsform. Folgende Aufgaben sollen analysiert bzw. umgesetzt werden:

### 2.1 Funktionale Anforderungen

Die funktionalen Anforderungen bestehen im Wesentlichen aus den nachstehenden Punkten. Detailliert werden diese in der Anlage «Todo StudentQuiz 4.0», Kapitel „Improvements“ und «New Features» beschrieben.

- Moodle Mobile App / Moodle Desktop
  Quizzes sollten innerhalb der Moodle App online, ggf auch offline durchgeführt werden können.
- Question States
  Die Question States sollen von aktuell {Approve, Disapprove} auf {New, Approve, Disapprove, Changed, Hidden, Deleted} erweitert warden
- Bereinigung der Question Types
  Entfernen von allen Question Types, die nicht in das Konzept von StudentQuiz passen.

### 2.2 Verbesserung bestehender Funktionen

Die angestrebten Verbesserungen bestehen im Wesentlichen aus den nachstehenden Punkten. Detailliert werden diese in der Anlage «Todo StudentQuiz 4.0», Kapitel «Improvement Features» beschrieben.

- Performance-Verbesserung
  Einige Sichten (z.B. Der Aufbau der Haupt-View, Filtern von Fragen, Start von Quizzes) sollen laufzeitmässig

optimiert werden. Dabei geht es insbesondere um die Optimierung der DB-Zugriffe und das Verwalten der Aggregatswerte. Dieses Ziel überschneidet sich mit der ebenfalls angestrebten Bereinigung der Datenbank.

- Bereinigung Datenbank
  Die Vorgängerarbeit hinterliess noch einige Problembereiche in der DB, z.B. studentquiz_practice und studentquiz_pogress. Auch sollten gelöschte Fragen inklusive der dazu gemachten Antworten, Ratings, Kommentare auch in der Db gelöscht werden. Dieses Ziel überschneidet sich mit der ebenfalls angestrebten Performance-Verbesserung.
- Test-Coverage
  Die Testbarkeit von StudentQuiz soll verbessert werden.

## 2.3 Bugs

- Leider wurden in der Vorgänger-Software (v3.1.1) einige Bugs bekannt. Diese sollten möglichst zu Beginn der Arbeit bereinigt werden und dienen damit auch zur Einarbeitung. Die Bugs werden in der Anlage «Todo StudentQuiz 4.0», beschrieben.

## 2.4 Nicht funktionale Anforderungen

- Usability
  StudentQuiz sollte unter allen Core-Themes laufen, inklusive Boost. Auf mobiles Handling wird Wert gelegt.
- Scalability
  StudentQuiz erzeugt höchste Nutzungsraten. Im Pilotversuch WI1 im FS18 mit ca 80 Studierenden und 450 Fragen wurden circa 21‘000 Zugriffe erzeugt, was extrem hoch ist. StudentQuiz soll aber auch Gruppen mit bis zu 500 Studierenden und 5'000 Fragen bedienen können. Dann ist mit circa 500'000 Zugriffen innerhalb eines Semesters zu rechnen.  Das System soll so getestet werden, dass es mindestens 50 Zugriffe pro Minute bedienen kann.
- Mehrsprachigkeit
  StudentQuiz ist in vielen Sprachen verfügbar. Änderungen im UI sind auf Deutsch und Französisch zu übersetzen.
- Dokumentation
  Da das Projekt nach der Bachelorarbeit weiterentwickelt wird, wird Wert auf eine gute Dokumentation für die verschiedenen Anwender und Entwickler gelegt. Die englischsprachigen Manuals für Administrator, Teacher und Student sind in Absprache mit dem Betreuer zu aktualisieren.

# 3. Zur Durchführung

Interesse an Design und Entwicklung einer Open Source Web-Anwendung im Bereich E-Learning wird vorausgesetzt.

Mit dem Betreuer finden Besprechungen gemäss Absprache statt. Die Besprechungen sind von den Studenten mit einer Traktandenliste vorzubereiten und die Ergebnisse sind in einem Protokoll zu dokumentieren, das dem Betreuer per E-Mail zugestellt wird. Die Projektsprache ist Englisch.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben.

# 4. Dokumentation und Abgabe

Wegen der beabsichtigten Weiterverwendung der Ergebnisse wird auf Vollständigkeit und Qualität der Dokumentation in englischer Sprache erhöhter Wert gelegt.

Die Dokumentation zur Projektplanung und -verfolgung ist gemäss den Richtlinien der Abteilung Informatik anzufertigen. Die Detailanforderungen an die Dokumentation der Recherche- und Entwicklungsergebnisse werden entsprechend dem konkreten Arbeitsplan festgelegt.

Die Dokumentation ist vollständig in drei Exemplaren abzugeben.

Neben der Dokumentation sind abzugeben:

- ein Poster zur Präsentation der Arbeit
- alle zum Nachvollziehen der Arbeit notwendigen Ergebnisse und Daten (Quellcode, Buildskripte, Testcode, Testdaten usw.)
- Material für eine Abschlusspräsentation (ca. 20’)

# 5. Termine

| | |
|---|---|
| 17.09.18 | Beginn der Bachelorarbeit |
| 17.12.18 | Die Studierenden geben den Abstract für die Diplomarbeitsbroschüre zur Kontrolle an ihren Betreuer/Examinator frei. Die Studierenden erhalten vorgängig vom Studiengangsekretariat die Aufforderung mit den Zugangsdaten zur Online-Erfassung des Abstracts für die Broschüre.<br><br>Die Studierenden senden per Email das A0-Poster zur Prüfung an ihren Examinator/Betreuer.<br><br>Vorlagen sowie eine ausführliche Anleitung betreffend Dokumentation stehen unter den allgemeinen Infos Diplom-, Bachelor- und Studienarbeiten zur Verfügung. |
| 18.12.18 | Der Betreuer/Examinator gibt das Dokument mit dem korrekten und vollständigen Abstract der Broschüre zur Weiterverarbeitung an das Studiengangsekretariat frei. |
| 21.12.18 bis 12:00 | Abgabe des Berichtes an den Betreuer bis 12.00 Uhr.<br>Fertigstellung des A0-Posters bis 12.00 Uhr. |
| 07.01.19 – 01.02.19 | Mündliche BA-Prüfung. |
| 01.03.19 | Bachelorfeier und Ausstellung Bachelorarbeiten |

## 6. Beurteilung

Eine erfolgreiche Bachelorarbeit erhält 12 ECTS-Punkte (1 ECTS Punkt entspricht einer Arbeitsleistung von ca. 25 bis 30 Stunden). Die Bewertung erfolgt gemäss nachstehender Kriterien:

| Gesichtspunkt | Gewicht |
|---|---|
| 1. Organisation, Durchführung (Projektplanung u. Nachführung Arbeit gemäss Projektplan, Selbstständigkeit, Einsatz, Zusammenarbeit mit Auftraggeber, Betreuer) | 1/6 |
| 2. Bericht (Inhalt des Projektschlussberichts, Gliederung, Darstellung, Sprache der gesamten Dokumentation) | 1/6 |
| 3. Inhalt | 1/2 |
| 3.1 Problemanalyse (Vorstudie, Literaturstudium, Anforderungsspezifikation, Anforderungsanalyse, Domainanalyse) | 1/6 |
| 3.2 Lösungsentwurf (Lösungsvarianten und deren Beurteilung, Variantenentscheid, Konzept, Entwurf) | 1/6 |
| 3.3 Realisierung und Test | 1/6 |
| 4. Präsentation und Mündliche Prüfung zur Bachelorarbeit | 1/6 |
| | |

Im Übrigen gelten die Abläufe und Regelungen Studien- und Bachelorarbeiten im Studiengang Informatik.

## 7. Betreuer

Prof. Frank Koch, Dozent für Wirtschaftsinformatik

## 8. Anlagen

Dokument „Todo StudentQuiz 4.0“ auf geshartem GoogleDrive

# 2 Abstract

StudentQuiz ist eine frei verfügbare Erweiterung für die Lernplattform Moodle. Moodle ist eine Open Source Software, welche in vielen Ausbildungsstätten wie der Open University UK oder der HSR, sowohl von Studierenden als auch von Lehrpersonen rege genutzt wird. Die Software funktioniert modular. Bereits in der Grundausstattung existieren verschiedenste Module wie Forum, Wiki, Glossar, Datenbank, etc.

Die Idee hinter der Erweiterung StudentQuiz ist, dass Studierende gemeinsam Lernfragen und Antworten zusammentragen und diese danach angemeldeten Nutzern zum Üben von Inhalten in Form eines Quizes zur Verfügung stellen. Zentral ist dabei der Qualitätssicherungsprozess, welcher sowohl auf der Prüfung von Fragen und Antworten durch die Lehrpersonen als auch auf Bewertungen und Kommentare der Studierenden setzt. Eine weitere wichtige Funktion von StudentQuiz ist die Möglichkeit, Fragen umfangreich zu filtern und je nach Bedürfnis unterschiedlich zu einem Quiz zu bündeln.

Im Rahmen dieser Bachelorarbeit wird zu verschiedenen Entwicklungen der Applikation beigetragen. Nach der Einführung folgen Analysen und Lösungsansätzen, die in drei technischen Berichten detailliert beschrieben werden. Im ersten Bericht werden bestehende Probleme der Erweiterung diskutiert und Lösungen dafür erarbeitet, sowie Verbesserungen vorgeschlagen und deren Umsetzung aufgezeigt. Im zweiten Bericht steht die Konzipierung der Implementation von StudentQuiz in die bestehende Moodle Mobile App im Vordergrund, womit die Erweiterung auch auf mobilen Geräten genutzt werden könnte. Im dritten Bericht steht die Performance der Software im Vordergrund. Hier werden verschiedene Engpässe in der Leistung untersucht. Die Performance-Optimierungen mit der grössten Wirkung wurde dann innerhalb dieser Bachelorarbeit implementiert.

# 3 Management Summary

## 3.1 Ausgangslage

Die Erweiterung StudentQuiz für die Lernplattform Moodle existiert und steht bereits erfolgreich produktiv im Einsatz. Es besteht allerdings Verbesserungsbedarf. Sowohl in Form von kleineren Problemen, für die Lösungen erarbeitet werden sollten, als auch im Bezug auf die gesamte Performance, welche verbessert werden kann. Des Weiteren muss eine Möglichkeit zur Implementierung der Erweiterung in Moodle Mobile, die Version von Moodle für Smartphones und Tablets, gefunden werden. Diesen drei Problemen widmet sich diese Bachelorarbeit.

Sie baut dabei auf mehrere vorgängig verfasste Arbeiten auf. Neben zwei Studentenarbeiten ist vor allem die Bachelorarbeit "Moodle Plugin StudentQuiz"von Denis Manente und Simon Schaefer grundlegend.[1] Als besonders hilfreicher Abschnitt hat sich das umfangreiche Continuous Integration Testing erwiesen, das von der erwähnten Bachelorarbeit implementiert wurde. Ebenfalls wichtig ist der bereits aufgesetzte Entwicklungsserver, auf den auch diese Arbeit zurückgreift.

Zu beachten ist die Tatsache, dass StudentQuiz eine bereits weitverbreitete Applikation ist, die aktiv im Einsatz ist. Sie wird dementsprechend rege genutzt. Dies ermöglicht es den Entwicklern, Rückmeldungen und Vorschläge aus der Community zu beachten und in die Weiterentwicklung mit einzubeziehen.

## 3.2 Vorgehen

Langfristig wünschenswert ist die Integration von StudentQuiz in Moodle Mobile, sowie die Aufnahme des Plugins als einen festen Bestandteil im Moodle core. Um diesen Zielen einen Schritt näher zu kommen, setzt sich diese Arbeit intensiv mit funktionellen Erweiterungen und Verbesserungen von Fehlern und Performanceproblemen in Moodle Web auseinander.

Die Integration von StudentQuiz in Moodle Mobile benötigt das Durcharbeiten von Dokumentationen, die Analyse von bestehenden Implementierungen und das Ausarbeiten der Anforderungen an die Funktionalität.

Für die Fehler- und Performanceverbesserungen sind aufwändige Codeanalysen, das umfangreiche Nachvollziehen von Strukturen, Debugging und der Einsatz von verschiedenen Messmethoden für die effiziente Lokalisierung der relevanten Problemstellen von Nöten.

## 3.3 Ergebnisse

In allen drei Teilen der Arbeit wurden signifikante Fortschritte für StudentQuiz erzielt. Mit der Analyse und Konzipierung der Erweiterung für Moodle Mobile existiert nun ein Lösungsvorschlag für die Implementation. Daneben konnten mehrere kritische Fehler in der Punkteberechnung der Ranglisten von Fragen innerhalb von StudentQuiz behoben werden. Die Analyse der Performance-Engpässe hat deutliche Verbesserungen ermöglicht, die im Rahmen dieser Arbeit auch bereits implementiert wurden. Darüber hinaus wurde ein Konzept für die Erweiterung der Zustände für eine effiziente Qualitätssicherung der Fragequalität erarbeitet.

Die umgesetzten Lösungen für die gefundenen Fehler und die Verbesserung der Performance resultierten in dem neuen Release 3.2.1. `https://moodle.org/plugins/pluginversion.php?id=18578`

# 4 Einführung in den technischen Bericht

## 4.1 Moodle

Moodle ist der Name einer frei verfügbaren Open Source Software für Lernmanagement. Sie wird als Lernplattform seit fast 20 Jahren an verschiedenen Schulen, Hochschulen, Universitäten und weiteren Ausbildungsstätten genutzt und zwar sowohl von Schülern und Studierenden als auch von Lehrpersonen, welche über die Plattform online Unterrichtsmaterialien verfügbar machen können. In der Schweiz ist dies beispielsweise an der Open University UK und der HSR der Fall. Weltweit wird Moodle allerdings in über 200 Ländern in über 100‘000 Installationen genutzt.

Die aktuellste Version ist Moodle Version 3.6. Neue Versionen erscheinen halbjährlich jeweils im Mai und im November. Für diese Arbeit ist Version 3.5 relevant. Die Erweiterung StudentQuiz, um die sich diese Arbeit dreht, ist mit Moodle Version 3.4 und höher kompatibel.

Der Name Moodle steht für sich aus „modular object-oriented dynamic learning environment“. Wie der Name besagt, ist die Software modular aufgebaut. Aufbauend auf einer Standardausführung (Moodle Core) können etliche Erweiterungen zusätzlich installiert werden, welche als Aktivitäten (engl. Activity) bezeichnet werden. Bereits in der Standardausführung sind jedoch Aktivitäten integriert, die je nach Bedarf installiert werden können.

## 4.2 StudentQuiz

StudentQuiz ist eine Erweiterung für Moodle, welche (noch) nicht in die Standardausführung aufgenommen wurde, jedoch als Plugin zusätzlich installiert werden kann. Die Erweiterung stellt den Moodle Nutzern eine kollaborative Fragedatenbank zur Verfügung. Sie ermöglicht es den Teilnehmern eines Kurses, Fragen zum behandelten Stoff mit dazugehörigen Antworten zu formulieren, um damit die Inhalte zu üben. Zentral ist dabei die Qualitätssicherung der Fragen in Form von deren Bestätigung durch Dozierende und Bewertung anderer Nutzer.

## 4.3 Questionbank

Das Kernelement von StudentQuiz ist die sogenannte QuestionBank, Fragedatenbanken zu verschiedenen Themen, welche hierarchisch kategorisiert werden können.

StudentQuiz zeigt nach der Beantwortung einer Frage an, ob diese richtig beantwortet wurde und bietet allenfalls weitere Hinweise. Im Gegensatz dazu zeigt das in Moodle integrierte Standardmodul Quiz die Resultate erst am Schluss an.

Bei StudentQuiz liefern die eingeschriebenen Schüler oder Studenten die Fragen, wobei durch die Qualitätskontrolle der Lehrpersonen gesichert werden kann, dass keine inkorrekten Fragen oder Antworten Zugelassen und fälschlicherweise fürs Üben genutzt werden. Die Fragen können durch die Dozenten entweder angenommen oder abgelehnt werden, wobei abgelehnte Fragen von den Studierenden nochmals neu formuliert und eingegeben werden können.

## 4.4 Legitimation für eine Bachelorarbeit

Vor Beginn der Bachelorarbeit muss sichergestellt werden, ob die Problemstellung genug Bereiche für einen konzeptionellen, theoretischen und praktischen Teil hergibt.

**Praktischer Teil** Diese Bachelorarbeit ist in erster Linie ein Programmierprojekt an einem bestehenden etablierten Produkt im produktiven Einsatz. Der Betreuer Prof. Frank Koch spielt dabei die Rolle des Auftraggebers, der mich mit der Weiterentwicklung nach seinen Vorstellungen beauftragt.

**Konzeptioneller Teil** Um die Applikation um Funktionen zu erweitern und bestehende Fehler zu verbessern ist eine ausführliche Analyse durchzuführen und eine entsprechende Lösung auszuarbeiten. Alle Vorschläge sollen zu einer Weiterentwicklung des Projektes beitragen.

**Theoretischer Teil** Die in der Bachelorarbeit angewandten Methoden und Einsichten greifen auf theoretische Grundlagen zurück, die mir während meiner Ausbildung vermittelt wurden.

## 4.5 Gliederung des technischen Berichtes

In dieser Arbeit wurde das Lokalisieren und Lösen verschiedenen Problemen, das Hinzufügen von neuen Funktionen, sowie die Implementation der Erweiterung für Moodle Mobile behandelt. Die wichtigsten drei Themen sind also: Das erarbeiten von Lösungen für Fehler, sowie kleinere Verbesserungen, die Entwicklung der Erweiterung für die Mobile App und die Optimierung der Performance. Für diese drei unterschiedlichen Bereiche wurden verschiedene Analysemethoden angewandt und voneinander unabhängige Resultat erzielt. Die praktische Arbeit, sowie der technische Bericht sind dementsprechend ebenfalls in drei Teile gegliedert.

## 4.6 Wichtige Begriffe

In der Tabelle 1 werden die wichtigsten Begriffe aufgelistet und nochmals erklärt.

| **Begriff** | **Bedeutung** |
|---|---|
| MoodleMoodle core | Beinhaltet die gesamte Webapplikation Moodle in der »Basisversion« ohne installierte Plugins.[10] |
| (Moodle) Plugin(Moodle) Activitymod | Eine Erweiterung für Moodle, die Funktionalität von Moodle core nutzt. |
| Moodle Web | Bezeichnet spezifisch Moodle core. Der Begriff wird verwendet, um besser von Moodle mobile unterscheiden zu können. |
| Moodle mobile | Moodle mobile ist eine auf dem ionic Framework basierte Applikation für den Einsatz auf Smartphones. Für den Betrieb von Moodle mobile wird Moodle Web benötigt. |

Tabelle 1: Wichtigste Begriff für das Verständnis der Arbeit

## 4.7 Nutzung der Vorarbeit

Für diese Bachelorarbeit war vor allem die Bachelorarbeit »Moodle Plugin StudentQuiz« von Denis Manente und Simon Schaefer[1] relevant. Sie hat zentrale Informationen über die neue Struktur von StudentQuiz 3.0 geliefert und war sehr hilfreich in Bezug auf die Entwicklungsumgebung.

Konkret wurden folgende Punkte übernommen:

**User Stories**
Die Bachelorarbeit »Moodle Plugin StudentQuiz« von Denis Manente und Simon Schaefer[1] hat alle User Stories detailliert aufgelistet. In meiner Arbeit hat es dazu keine wesentlichen Änderungen an der Grundfunktionalität gegeben. Deshalb sind diese User Stories weiterhin gültig (siehe Abschnitt 5.1.2[1]).

**Linux-Server auf Basis des SE2-Image der HSR**
Die Bachelorarbeit »Moodle Plugin StudentQuiz« von Denis Manente und Simon Schaefer[1] hat einen Testserver zurückgelassen. Dieser war zusätzlich zu der lokalen Entwicklungsumgebung sehr hilfreich um manuelle Tests durchzuführen.

## 4.8 Lokale Entwicklungsumgebung

Die lokale Entwicklungsumgebung basierte auf einer Virtuellen Maschine Lubuntu in VirtualBox. Für die Installation von Moodle wurde php7.2, MySQL und apache2 installiert. Darauf wurde Moodle mit der StudentQuiz Erweiterung installiert (Ubuntu Softwarepakete siehe Abschnitt 9). Als IDE wurde PhpStorm verwendet. Mithilfe von Xdebug (Siehe Unterabschnitt 9.7) konnten Fehlerstellen schnell ausfindig gemacht werden.

Für die Virtuelle Maschine standen vier geteilte Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz und 4 GB RAM zur Verfügung.

## 4.9 Repository

Alle Codeänderungen wurden in den *develop4 Branch* als User `muqiuq` oder `p1albrec` auf GitHub commited. Es wurde darauf verzichtet den StudentQuiz Code dieser Arbeit beizulegen. Stattdessen wird auf das Onlinerepository verwiesen. Der Link zu den entsprechenden Commits lautet: `https://github.com/frankkoch/moodle-mod_studentquiz/commits/develop4`.

# 5 Technischer Bericht - Teil 1

Der erste Teil des technischen Berichtes beinhaltet verschiedene kleinere Entwicklungen. Es werden technische Fehler der Software gelöst und konzeptionelle Probleme durch Verbesserungen behoben. Die Änderungen im Code sind dementsprechend eher klein, der Grossteil des Aufwands entstand vor allem bei der Rekonstruktion des eigentlichen Problems.

## 5.1 Probleme mit dem Filtern von Fragen

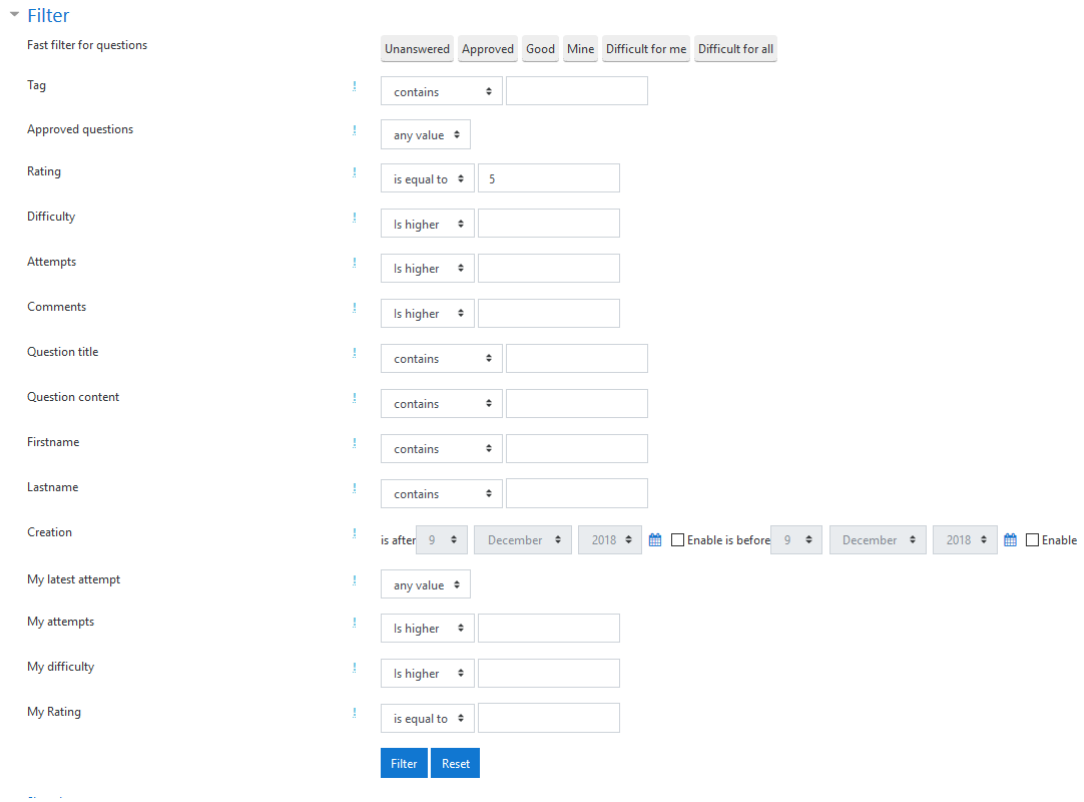


Abbildung 1: view.php - Filterformular für die Fragen

Zwei bekannte Probleme von StudentQuiz liegen in der Filterung von Fragen. Hier tut sich die Software schwer mit der Verarbeitung von Vor- und Nachnamen sowie mit der Eingabe von Dezimalzahlen im Feld „Difficulty“, wenn „is equal to“ ausgewählt wird. Beide Probleme betreffen das Formular in Abbildung 1.

### 5.1.1 Lösung

Bei allen Fehlern handelte es sich um Programmierfehler. Teilweise waren SQL Abfragen falsch und es gab Probleme mit den PHP Datentypen. Die effektiven Änderungen sind im Code nachzuvollziehen.

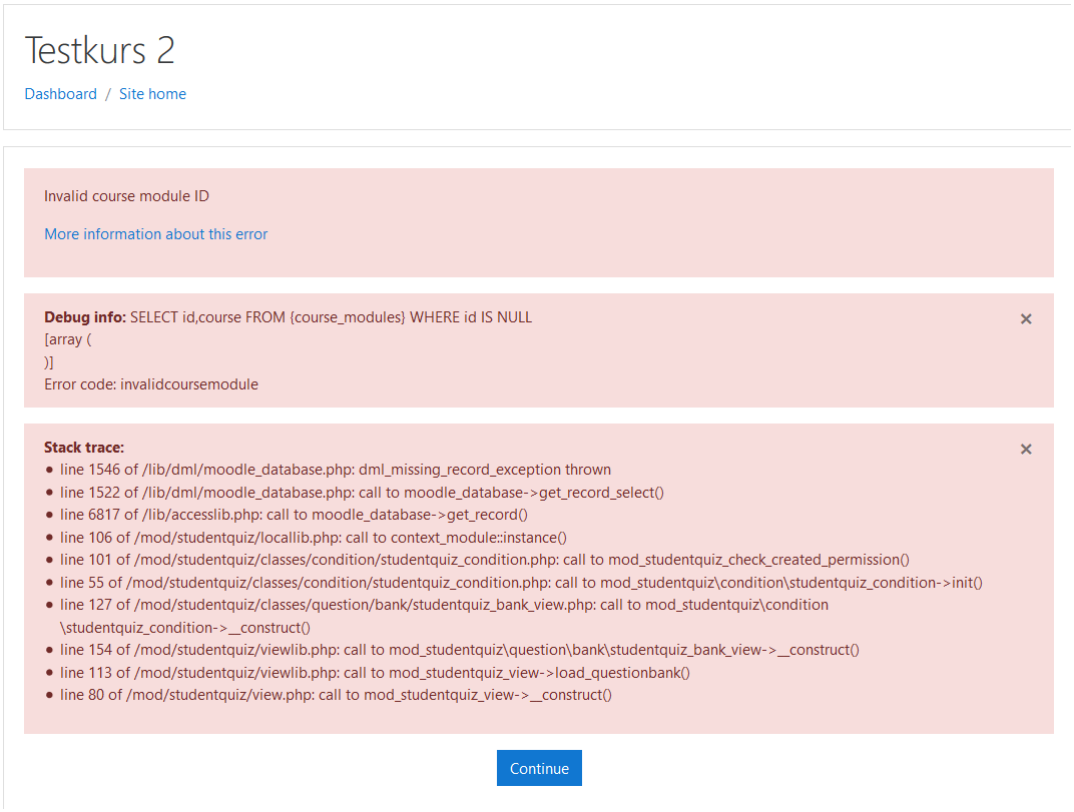


Abbildung 2: view.php - Beim Filtern nach Vorname oder Nachname tritt dieser Fehler auf

## 5.2 Fehlerhafte Punkteberechnung

Der Punkteberechnung liegen zwei generisch zusammengesetzte SQL Queries zu Grunde. Der Fehler dabei ist, dass gelöschte Fragen und nicht bewertete Fragen in die Punkteberechnung miteinbezogen werden, obwohl das nicht korrekt ist. Die Ursache des ersten Problems liegt darin, dass Fragen, welche gelöscht werden, zwar aus der Tabelle mit den Fragen entfernt werden, aber die Informationen zu den Antwortversuchen in der Datenbank verbleiben. Gelöst wird das Problem mit einer Anpassung der betroffenen SQL Query. Hier wird ein zusätzlicher *Join* auf die Fragen-Tabelle gemacht, um Daten gelöschter Fragen auszuschliessen. Die Ursache des zweiten Problems, dem Mitzählen von nicht bewerteten Fragen, liegt darin, dass als Faktor für den Bewertungsdurchschnitt die Anzahl der vom User erstellten Fragen verwendet wird, statt der Anzahl, der vom User nicht nur erstellten, sondern auch bewerteten Fragen. Auch bei diesem Problem reicht eine einfache Anpassung der SQL Query.

## 5.3 Neue Zustände

### 5.3.1 Ausgangslage

Die Qualitätssicherung durch den Dozenten geschieht dadurch, dass er Fragen genehmigen (*Approve*) oder ablehnen (*Disapprove*) kann. In der bisherigen Version von StudentQuiz ergeben sich mit diesen beiden Zuständen jedoch folgende Probleme:

- Neue Fragen sind automatisch *Disapproved*. Es kann nicht zwischen einer abgelehnten und einer neuen Frage unterschieden werden.
- Wird eine Frage geändert, verbleibt sie im ursprünglichen Zustand. Wenn diese also bereits genehmigt wurde, verbleibt diese weiterhin in diesem Zustand, ohne dass die Qualität der Änderung überprüft wird.
- Es gibt keinen Zustand für Fragen, die sich auf nicht besprochenen Stoff beziehen.
- Gelöschte Fragen werden unwiderruflich entfernt. Dies kann Beispielsweise bei ethischen Fragen wie Rassismus ein Problem sein, wenn Beweise verschwinden, welche zum Nachweis in der Strafverfolgung benötigt würden.

### 5.3.2 Lösung

**Question States**
Als Lösung für das Problem »Fragezustand« werden neben genehmigt und abgelehnt zwei neue Zustände eingeführt: Neu (*New*) und Geändert (*Changed*). Beide Zustände *New* und *Changed* können dabei ausschließlich Applikationsgesteuert auftreten. Die Zustände *Angenommen* und *Abgelehnt* hingegen können ausschliesslich manuell durch den Dozenten festgelegt werden. Wird eine Frage erstellt, kommt sie in den Zustand *New*. Erst nach einer Qualitätskontrolle kann sie in den Zustand *Angenommen* oder *Abgelehnt* kommen. Wird sie geändert, kommt sie wiederum in den Zustand *Changed*, sofern sie vorher in einem der Zustände *Approved* oder *Disapproved* war. In Abbildung 3 sind die verschiedenen Zustände (alt und neu) mit den dazugehörigen Symbolen dargestellt. In Abbildung 4 sind die neuen Zustände und alle möglichen Übergänge zwischen den Zuständen dargestellt.

**Question visibility**
Als Lösung für das Problem des unwiderruflichen Löschens können Fragen neu durch den Dozenten ausgeblendet und zu einem späteren Zeitpunkt allenfalls wieder eingeblendet werden. Für den Autor der Frage und den Dozenten wird die Frage anschliessend ausgegraut dargestellt. Hierzu wird ein zusätzliches Symbol eingeführt. Beim Löschen der Fragen werden diese in der Datenbank nur noch als gelöscht markiert, existieren aber weiterhin.

### 5.3.3 Umsetzung

Die neuen Zustände (*Question States*) werden im Rahmen dieser Bachelorarbeit nicht umgesetzt, weil sich mit der Open University UK `http://www.open.ac.uk/` eine dritte Partei sehr motiviert gezeigt hat, diese Umsetzung zu übernehmen. Dafür wurde gemeinsam mit dem Betreuer ein englisches Dokument mit den entsprechenden Anforderungen erstellt (siehe Unterabschnitt C.2).

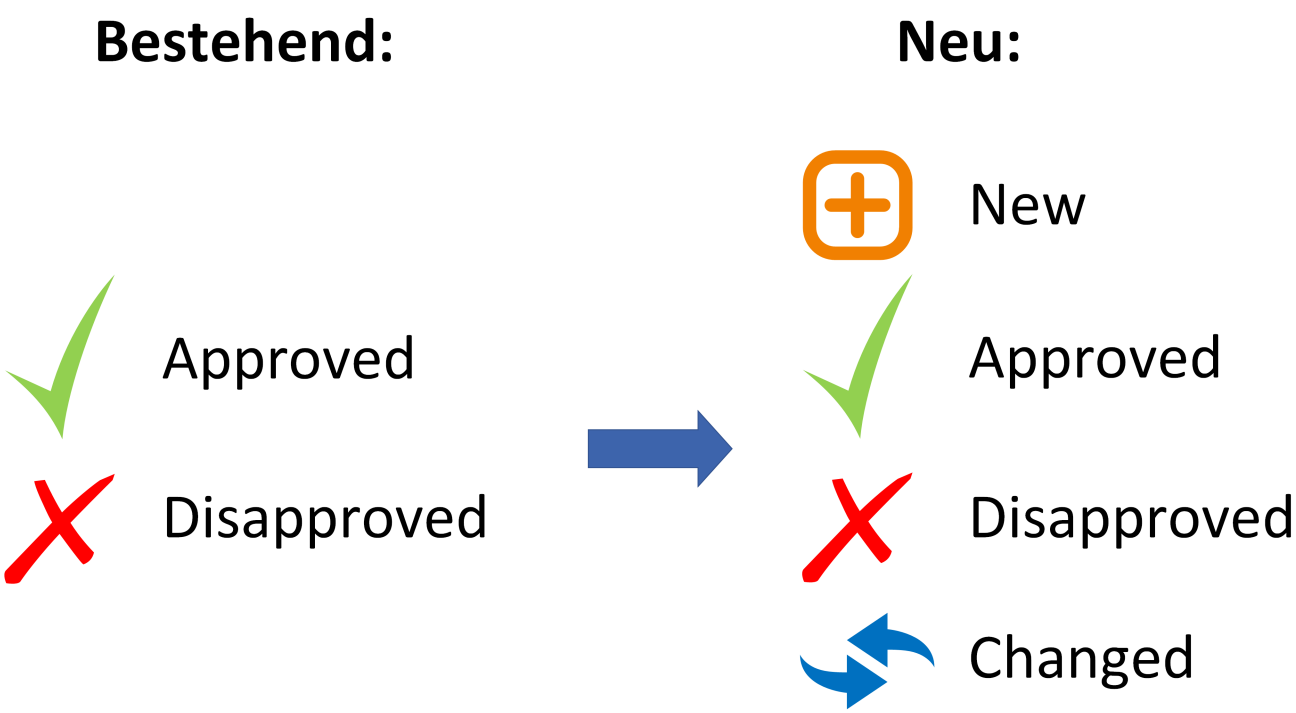


Abbildung 3: view.php - Die bestehenden und die neuen Zustände mit den dazugehörigen Symbolen

## 5.4 Essay Frage

**Ausgangslage**
Im StudentQuiz können verschiedene Fragetypen verwendet werden. Es gibt zwei Fragetypen, deren Idee von den anderen abweicht: *Essay* (engl. Aufsatz) und *Description* (engl. Beschreibung).

Beim Fragetyp *Essay* muss der Student einen »Aufsatz« als Antwort verfassen, der im Nachhinein vom Dozent ausgewertet wird. Beim Fragetyp *Description* handelt es sich nur um die Anzeige eines Infotextes.

Der Fragetyp *Essay* widerspricht aufgrund des notwendigen manuellen Eingriffs durch den Dozenten dem Konzept der automatischen Auswertung. Deshalb soll verhindert werden, dass dieser Fragetyp innerhalb von StudentQuiz verwendet wird.

**Analyse**
StudentQuiz implementiert die einzelnen Fragetypen mit den entsprechenden Bedienoberflächen nicht selber, sondern nutzt die Funktionalität von Moodle Core. Zudem können Fragen von StudentQuiz über die QuestionBank in ein Moodle Core Quiz übertragen werden. Um grössere Kompatibilitätsstörungen auszuschliessen, wurde deshalb beschlossen, lediglich das Hinzufügen von Fragen zu verhindern.

Nebst dem funktionellen Problem liegt bei den Fragetypen *Essay* und *Description* zusätzlich ein Softwarefehler vor. Wenn einer der beiden Fragetypen verwendet wird, zeigt StudentQuiz beim Durchgehen von Fragen die Schaltflächen `Back` und `Next` nicht an. Als Folge kann mit das Quiz nicht fortgesetzt werden. Dieses Problem soll ebenfalls behoben werden.

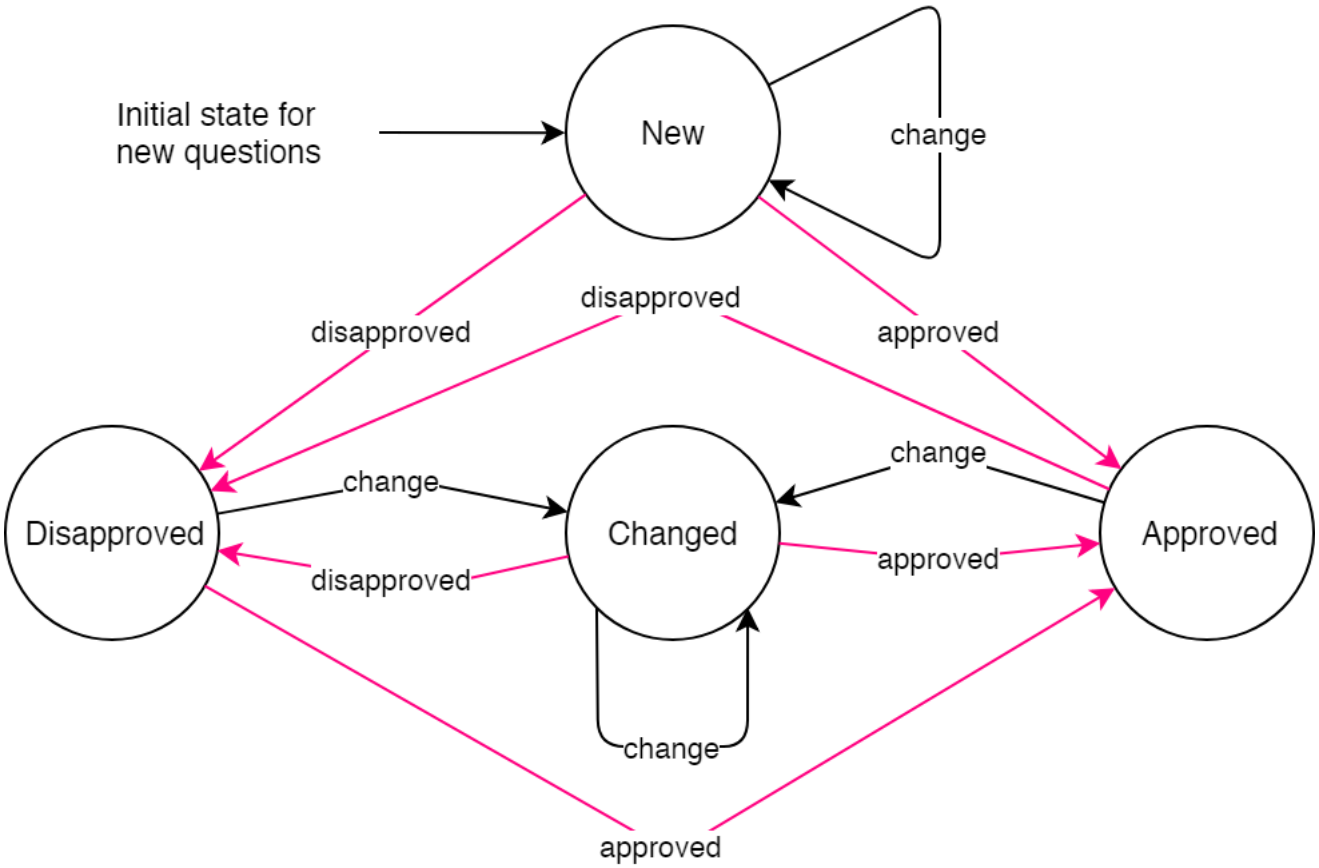


Abbildung 4: view.php - Alle Zustände und die möglichen Übergänge

**Implementation**
Ausgehend von der Analyse wurde folgende Implementierung im Rahmen des GitHub Issues #66 umgesetzt (`https://github.com/frankkoch/moodle-mod_studentquiz/issues/66`).

- Der Fragetyp *Essay* kann in den Einstellungen nicht mehr ausgewählt werden.
- Beim Hinzufügen einer neuen Frage (*Create new question*) wird der Fragetyp *Essay* nicht mehr angezeigt.
- Der *Next* und der *Back* Button werden beim Fragetyp *Essay* und *Description* angezeigt.
- Der Fragetyp *Essay* ist *readonly*. Das heisst: Der Inhalt der Frage wird anzeigt, aber es kann keine Antwort eingegeben werden.

## 5.5 Travis CI

Travis CI[15] ist eine Open Source Software für *Continuous Integration* und zugleich kostenlose Online Dienstleistung für Open Source Projekte. Ein *Push Request* des Benutzers *KietChan* hat dazu geführt, dass die *Continuous Integration* fehlschlug (`https://travis-ci.org/frankkoch/moodle-mod_studentquiz/builds/407142725?utm_source=github_status&utm_medium=notification`). Die Änderung implementierte GDPR[6] gemäss dem Implementationsrichtlinien von Moodle.[16] Die Änderungen brachte folgende

Probleme mit:

- Der neue Code war nur mit PHP >= 7.0 kompatibel
- Der neue Code war nicht mit moodle < 3.4 kompatibel
- Der neue Code entsprach nicht den Coding Richtlinien von moodle[7]
- Die Tabelle `studentquiz_progress` hatte kein `ID` Feld, dass jedoch zwingend notwendig ist.

### 5.5.1 Lösung und Umsetzung

- Für die neue StudentQuiz wurde der Support für moodle < 3.4 eingestellt
- PHP Versionen kleiner wie 7.0 mussten nicht mehr beachtet werden, weil moodle 3.4 mindestens PHP 7.0 als Mindestanforderung hat.
- Der neue Code wurde den Coding Richtlinien angepasst
- Die Tabelle `studentquiz_progress` wurde aus dem Code vorübergehend auskommentiert.

Als Folge der oben genannten Massnahmen funktionierten alle Unit und Integration Tests wieder.

# 6 Technischer Bericht - Teil 2 - Moodle Mobile

## 6.1 Ausgangslage

Die erste in den Release Notes erwähnte Version von Moodle Mobile ist im Juli 2015 erschienen.[2] Seitdem hat sich die mobile Internetnutzung weiter erhöht.[3] Es ist demnach für StudentQuiz unumgänglich auf der Mobile App integriert zu werden. Um die nachfolgende Analyse über die »Bereitmachung« von StudentQuiz für die mobile Welt verstehen zu können, müssen zuerst einige Moodle spezifische Details aufgezeigt werden. Moodle ist in erster Linie eine für den Desktop optimierte Webapplikation. Sie ist grösstenteils in PHP programmiert und bedient sich zur Darstellung im Web HTML, CSS und Javascript. Obwohl Moodle Web Resposive Design nutzt, ist das Benutzererlebnis auf einem Smartphone eher unattraktiv. Beim StudentQuiz selber sieht es nicht besser aus. Es kann gesagt werden, dass StudentQuiz auf dem Smartphone kein brauchbares Benutzererlebnis bietet. Aktuell erscheint in der Moodle Mobile App beim Aufruf einer StudentQuiz Activity die in Abbildung 5 gezeigte Fehlermeldung.

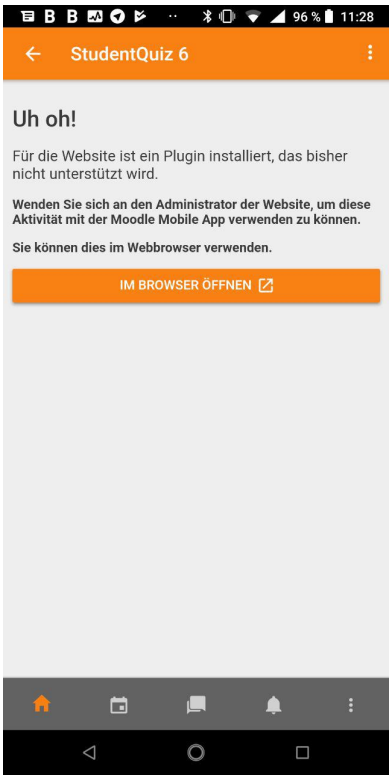


Abbildung 5: Fehlermeldung beim Aufruf von StudentQuiz in Moodle Mobile

### 6.1.1 Unterschiede Moodle und Moodle Mobile

Um zu verstehen, wie die Bereitstellung eines Plugins für die Mobilgeräte innerhalb von Moodle funktioniert, muss man zuerst wissen, dass Moodle und Mooble Mobile zwei verschiedene Applikationen sind. Moodle Mobile greift über eine definierte HTTP Schnittstelle auf Moodle zu. Alle Plugins, die Moodle Mobile unterstützen (von Moodle core oder von Dritthersteller) haben eine Schnittstelle für Moodle Mobile. Für die Umsetzung gibt es verschiedene Implementierungsvarianten. Ein Teil dieser Implementierungsvarianten

ist im Dokument »Moodle Mobile«[4] erklärt.

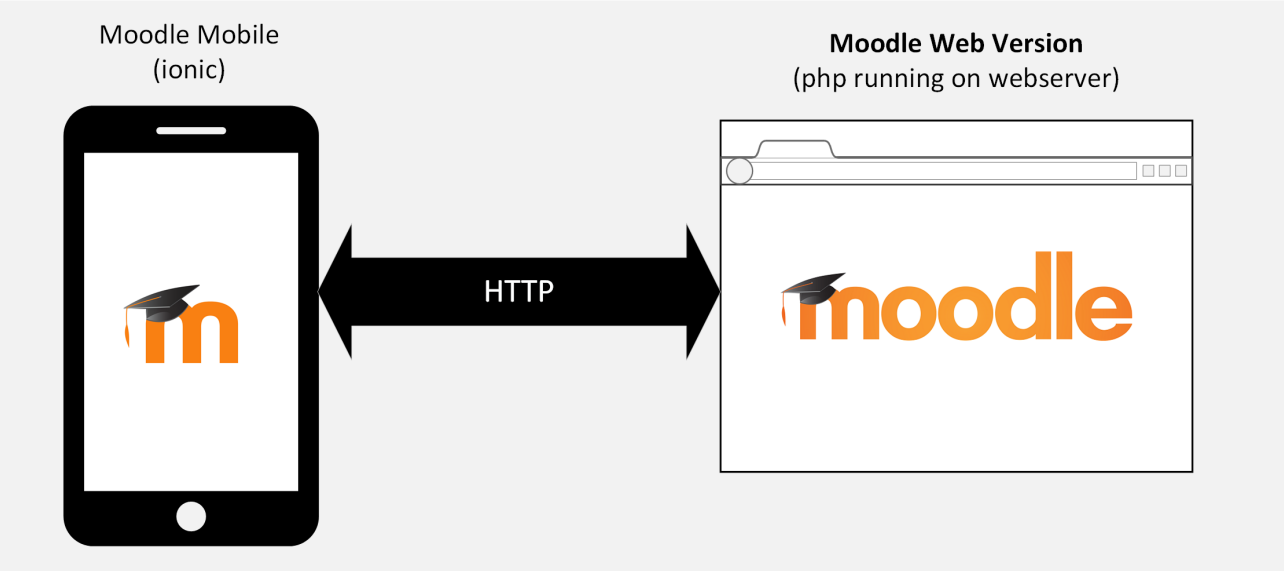


Abbildung 6: Moodle Mobile und die Schnittstelle zu Moodle (Web)

## 6.2 Anforderungen

In einer ersten Version von StudentQuiz Mobile sollen folgende Basisanforderungen erfüllt werden:

### 6.2.1 Funktionalität

- Auswahl von Fragen mithilfe eines vereinfachten Filterverfahrens
- Beantwortung von Fragen (Alle Fragetypen)
- Bewertung von Fragen
- Kommentieren von Fragen

### 6.2.2 Implementation

- Anlehnung an Moodle Mobile Quiz (Siehe: Abbildung 7)
- Für die Darstellung der verschiedenen Fragetypen soll die vorhandene Implementation innerhalb von Moodle Mobile verwenden werden.
- Möglichkeit Fragen Offline zu beantworten

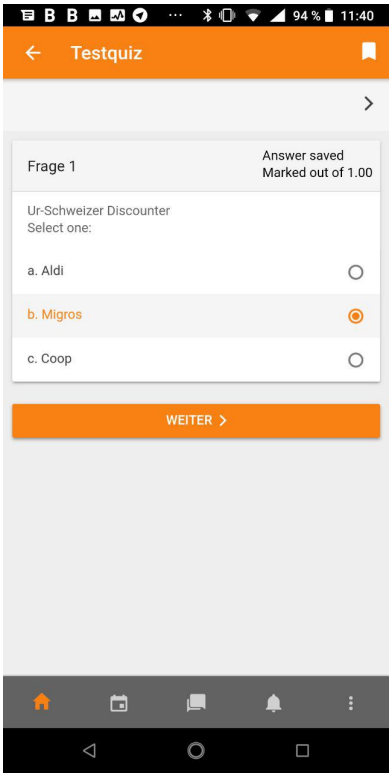


Abbildung 7: Moodle Mobile - Quiz

## 6.3 Design

### 6.3.1 Benutzerführung

Abbildung 8 stellt den Programmablauf im StudentQuiz Mobile dar. Er ist dem Ablauf der Webversion sehr ähnlich. Nach jeder Frage wird wie auch in der Web Variante in der Mobile Version umgehend das Resultat und die Erklärung angezeigt. Auch sehr wichtig ist die anschliessende Bewertung, die zentral für die Qualitätssicherung der Fragen ist.

In Abbildung 9 ist zusätzlich ein Markup für die Beantwortung einer Frage dargestellt.

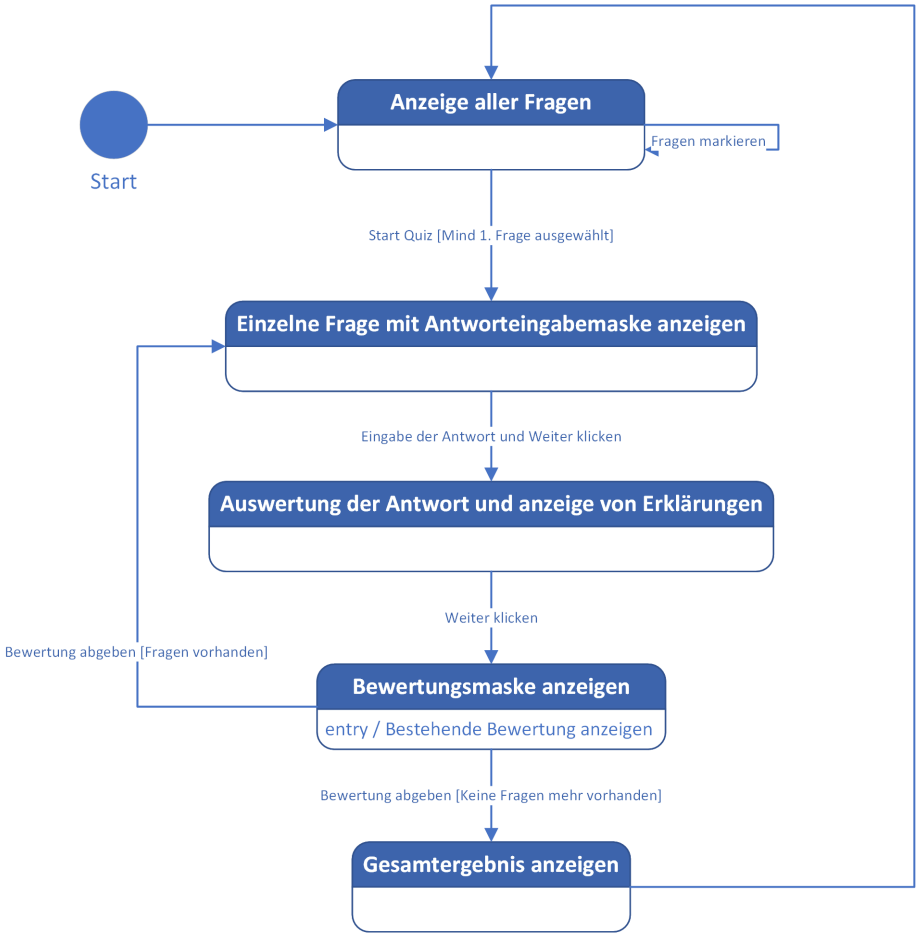


Abbildung 8: StudentQuiz Mobile Programmablauf aus Sicht des Benutzers

### 6.3.2 Datenübermittlung

Die Datenübermittlung soll auf ein Minimum reduziert sein. Das übertragen von vorgerenderten Ansichten soll vermieden werden. Genauso die laufende Synchronisierung während des Durchspielens des Quizzes. Alle Fragen mit ihren Antworten sollen beim Aufruf des StudentQuiz auf dem Smartphone geladen werden. Nach Abschluss oder Abbruch des Quizzes soll die Auswertung (beinhaltet die Antworten, die Wertung (richtig oder falsch), die Bewertungen und die Kommentare des Users) an den Server übermittelt werden.


Abbildung 9: Markup - Ablauf Fragebeantwortung und Abgabe der Bewertung

Wird ein StudentQuiz in der Mobile App geöffnet, sollen die Fragen nur dann neu übertragen werden, wenn eine Änderung vorliegt. Die Kodierung der Daten soll per JSON erfolgen. Eine entsprechende Schnittstelle ist bereits vorhanden.

### 6.3.3 Use Cases Fully Dressed

In den Tabellen 2,3 und 4 werden alle *Use Cases Fully Dressed* für StudentQuiz in Moodle mobile aufgeführt.

| | |
|---|---|
| Name | UC01: Anzeigen von StudentQuiz in Moodle mobile |
| Primary Actor | Student (User) |
| Beschreibung | In der Moodle mobile app: In einem Kurs wird eine StudentQuiz Activity geöffnet. |
| Stakeholders | ▪ Student |
| Preconditions | ▪ StudentQuiz Plugin ist installiert<br>▪ StudentQuiz Activity ist vorhanden |
| Postconditions | ▪ Der Student sieht alle Fragen<br>▪ Der Student kann einen Quiz Durchlauf starten.<br>▪ Der Student kann seinen Fortschritt und sein Ranking ansehen. |
| Main Success Story | 1. StudentQuiz Activity wird im Kursmodul angewählt.<br>2. StudentQuiz Activity wird angezeigt. |

Tabelle 2: UC01 Fully Dressed

| | |
|---|---|
| Name | UC02: Durchspielen eines Fragesets |
| Primary Actor | Student (User) |
| Beschreibung | In der Moodle mobile App wird ein Quiz gestartet und die ausgewählten Fragen durchgespielt. |
| Stakeholders | ▪ Student |
| Preconditions | ▪ StudentQuiz Plugin ist installiert<br>▪ StudentQuiz Activity ist vorhanden<br>▪ In der StudentQuiz Activity sind Fragen vorhanden. |
| Postconditions | ▪ Der Student hat eine oder mehrere Fragen beantwortet.<br>▪ Der Student hat nach jeder Frage direkt ein Feedback erhalten.<br>▪ Für jede durchgespielte Frage hat der Student eine Bewertung abgegeben.<br>▪ Der Student konnte nach jeder Frage einen Kommentar abgeben. |
| Main Success Story | 1. StudentQuiz Activity wird im Kursmodul ausgewählt.<br>2. StudentQuiz Activity wird angezeigt.<br>3. Fragen für den nächsten Quiz Durchlauf werden ausgewählt.<br>4. Quiz wird mit *Start Quiz* gestartet.<br>5. Bis alle ausgewählten Fragen beantwortet sind, wird folgender Prozess wiederholt:<br>(a) Frage wird angezeigt<br>(b) Student wählt Antwort aus und klickt auf *Check*<br>(c) Auswertung über die Richtigkeit der Antwort und zusätzliche Hinweise werden angezeigt.<br>(d) Der Student klickt auf *Next*<br>(e) Falls die Frage noch nicht Bewertet ist, gibt der Student eine Bewertung ab.<br>(f) Der Student kann freiwillig einen Kommentar schreiben.<br>(g) Der Student klickt auf *Next*<br>6. Alle Fragen sind beantwortet. |
| Alternative Flows | Der Quiz Durchlauf wird frühzeitig abgebrochen. Alle gegebenen Antworten werden trotzdem in die Punkteberechnung miteinbezogen. |

Tabelle 3: UC02 Fully Dressed

| Name | UC03: Anzeigen der Statistik |
|---|---|
| Primary Actor | Student (User) |
| Beschreibung | In der Moodle Mobile App: In einem Kurs wird eine StudentQuiz Activity geöffnet und die Statistik angezeigt. |
| Stakeholders | ▪ Student |
| Preconditions | ▪ StudentQuiz Plugin ist installiert<br>▪ StudentQuiz Activity ist vorhanden<br>▪ StudentQuiz Quiz Durchläufe sind vorhanden. |
| Postconditions | ▪ Der Student sieht alle Fragen<br>▪ Der Student kann seinen Fortschritt und sein Ranking ansehen. |
| Main Success Story | 1. StudentQuiz Activity wird im Kursmodul angewählt.<br>2. StudentQuiz Activity wird angezeigt.<br>3. Der Student wählt *Statistik*<br>4. Der Fortschritt und das Ranking werden angezeigt. |
| Alternative Flows | N/A |

Tabelle 4: UC03 Fully Dressed

## 6.4 Analyse

Für den Mobile Support von Plugins gibt es von Moodle eine Dokumentation.[5] In dieser Dokumentation werden drei verschiedene Implementationsmöglichkeiten vorgestellt:

| | Rendering | JS |
|---|---|---|
| Templates generated and downloaded when the user opens the plugins | Server | Nein |
| Templates downloaded on login and rendered using JS data | Client | Ja |
| Pure Javascript plugins | Client | Ja |

Tabelle 5: Implementationsmethoden für Moodle Plugins in Moodle mobile[5]

Der grösste Aufwand ist die Implementation der Fragetypen. Da gemäss den Anforderungen diese nicht selber Implementiert werden sollte, wurde zuerst diese Funktionalität im Code gesucht. Dazu wurde der Code von Moodle Web durchsucht. Jedes Plugin mit *mobile support* besitzt ein db/mobile.php. Beim Core Plugin Quiz wurde ein solche Datei jedoch nicht gefunden. Stattdessen wurden folgende zwei Dateien mit relevantem Code gefunden, der auf eine Schnittstelle hinweist (Siehe Abbildung 10).

```
1  ?php
2  / ...
3  functions = array(
4  'mod_quiz_get_quizzes_by_courses' => array(
5  'classname'     => 'mod_quiz_external',
6  'methodname'    => 'get_quizzes_by_courses',
7  'description'   => 'Returns a list of quizzes in a provided list of courses,
8  if no list is provided all quizzes that the user can view will be returned.',
9  'type'          => 'read',
10 'capabilities'  => 'mod/quiz:view',
11 'services'      => array(MOODLE_OFFICIAL_MOBILE_SERVICE)
12 ),
13
14 / ...
```

Abbildung 10: Ausschnitt von mod/quiz/db/services.php

Abbildung 11 zeigt einer der Rückgabefunktionen innerhalb des Moodle Core Quiz. Es werden *Arrays* zurückgegeben, die in einem anderen Teil des Codes in *JSON* umgewandelt und an die App übermittelt werden.

Der Abschnitt im Code, der mit der Darstellung auf dem Mobile App etwas zu tun hat, wurde in Moodle Web nicht gefunden. Stattdessen wurde dieser im *Source Code* der Mobile App gefunden.

Abbildung 12 zeigt wie die Darstellung einer einzelner Frage in einem Template der Mobile App programmiert ist.

Nach dieser Analyse war klar:

- Die Fragetypen werden mit dem html tag `<core-question` eingebunden.
- Die serverseitige Implementierung von Core Quiz weicht von der Dokumentation ab.
- Die Programmierung ohne den Einsatz von JavaScript bzw. TypeScript ist ausgeschlossen.

```php
?php
// ...

/**
* Quiz external functions
*
* @package    mod_quiz
* @category   external
* @copyright  2016 Juan Leyva <juan@moodle.com>
* @license    http://www.gnu.org/copyleft/gpl.html GNU GPL v3 or later
* @since      Moodle 3.1
*/
class mod_quiz_external extends external_api {

// ...
/**
* Returns a list of quizzes in a provided list of courses,
* if no list is provided all quizzes that the user can view will be returned.
*
* @param array $courseids Array of course ids
* @return array of quizzes details
* @since Moodle 3.1
*/
public static function get_quizzes_by_courses($courseids = array()) {
// ...
$result = array();
$result['quizzes'] = $returnedquizzes;
$result['warnings'] = $warnings;
return $result;
}
```

Abbildung 11: Ausschnitt von mod/quiz/classes/external.php

```
47 <!-- Questions -->
48 <form name="addon-mod_quiz-player-form" *ngIf="questions && questions.length && !quizAborted &&
   ↪  !showSummary">
49 <div *ngFor="let question of questions">
50 <ion-card id="addon-mod_quiz-question-{{question.slot}}">
51 <!-- "Header" of the question. -->
52 <ion-item-divider color="light">
53 <h2 *ngIf="question.number" class="inline">{{ 'core.question.questionno' | translate:{$a:
   ↪  question.number} }}</h2>
54 <h2 *ngIf="!question.number" class="inline">{{ 'core.question.information' | translate }}</h2>
55 <ion-note text-wrap item-end *ngIf="question.status || question.readableMark">
56 <p *ngIf="question.status" class="block">{{question.status}}</p>
57 <p *ngIf="question.readableMark"><core-format-text
   ↪  [text]="question.readableMark"></core-format-text></p>
58 </ion-note>
59 </ion-item-divider>
60 <!-- Body of the question. -->
61 <core-question text-wrap [question]="question" [component]="component"
   ↪  [componentId]="quiz.coursemodule" [attemptId]="attempt.id" [offlineEnabled]="offline"
   ↪  (onAbort)="abortQuiz()" (buttonClicked)="behaviourButtonClicked($event)"></core-question>
62 </ion-card>
63 </div>
64 </form>
```

Abbildung 12: Ausschnitt von src/addon/mod/quiz/player/player.html

## 6.5 Resultat

Die Analyse hatte bis zu diesem Zeitpunkt folgende Fragen aufgeworfen:

- Welche Implementationsmethoden (Siehe: Tabelle 5) können verwendet werden?
- Ist diese Implementierung im Rahmen der Bachelorarbeit umsetzbar?

Alle gesammelten Informationen wurden in einem kurzen Informationsdokument zusammengefasst und an den Betreuer Frank Koch übergeben (siehe Unterabschnitt C.1).

Das Fazit der persönlichen Gespräche von Frank Koch mit Moodle Entwicklern war, dass eine Implementation mit der Methode *Pure Javascript plugins* erfolgen müsste. Das erfordert jedoch grosse Kenntnisse über Ionic[12] und TypeScript.[17] Zudem würde die Einarbeitung in den Moodle Mobile Code mehrere Monate dauern. Das sprengt den Rahmen einer Bachelorarbeit, deshalb wurde auf das Implementationskonzept und die Implementation zugunsten anderer Features verzichtet.

# 7 Technischer Bericht - Teil 3 - Performance

## 7.1 Ausgangslage

In der Bachelorarbeit »Moodle Plugin StudentQuiz« von Denis Manente und Simon Schaefer[1] und in Feedback von Benutzern wurde auf verschiedene Peformanceprobleme aufmerksam gemacht.

Dieser Abschnitt konzentriert sich auf die Suche dieser Performance-Probleme und das Aufzeigen von möglichen Lösungen.

In StudentQuiz gibt es folgende Ansichten (Webseiten):

- **view.php** - Hauptansicht von StudentQuiz (siehe Abbildung 16)
- **attempt.php** - Quiz Ansicht mit einer einzelnen Frage
- **ranking.php** - Anzeige der eigenen Statistik und der Community Statistik
- **reportrank.php** - Anzeige des Rankings von allen Studenten
- **renderer.php** - Alle Klassen, die für das Rendering verantwortlich sind, sind in dieser Datei.

Die Ansicht `view.php` wird dabei immer als erstes angezeigt, wenn eine StudentQuiz Activity geöffnet wird. Es ist die meistbenutzte Ansicht.

Die Ansicht `attempt.php` wird immer aufgerufen, wenn eine Frage innerhalb eines Quiz Durchlaufes angezeigt wird.

## 7.2 Zielsetzung

Das Ziel ist es die Ladezeit von einzelnen Ansichten zu verringern. Dabei soll eine Antwortzeit des Servers von unter 2 Sekunden erreicht werden. Es wird nur die Zeit gemessen, die benötigt wird, um die Ansicht von Server zum Webbrowser zu übermitteln. Das Rendering im Webbrowser und das Laden von zusätzlichen Webseite Elementen (z.B. Bilder) werden in der Messung nicht berücksichtigt.

## 7.3 Lokalisierung von Problemstellen

Um Performance-Probleme effizient zu finden, verwende ich einen mehrteiligen Ansatz. Das Vorgehen ist je nach Programmiersprache und Applikation anders. Im vorliegenden Fall mit Moodle konzentrieren wir uns auf PHP und MySQL.

### 7.3.1 Messmethoden

| Schritt | Bestand-teil | Methode | Fragestellung |
|---|---|---|---|
| 1 | Testdaten | Manuelles Generieren von Testdaten Mithilfe eines PHP Scriptes. | Wie kann ich möglichst realitätsnahe Testdaten generieren? |
| 2 | Benutze-rerfahrung | Die Applikation »durchklicken« und sich nach Häufigkeit des Aufrufes langsame Ansichten notieren. | Welche Ansichten sind langsam? Welche Ansichten werden häufig aufgerufen? |
| 3a | xDebug Profiler Snapshot | Ein xDebug Profiler Snapshot zeigt, welche Funktionen in der Applikationen viel aufgerufen werden und wie viel Zeit diese in Anspruch nehmen. | Welche Funktionen sind langsam? |
| 4a | Manuelle Zeitmes-sung | Debug Zeilen für die Zeitmessung direkt im Code einfügen. Werte über Standardausgabe anzeigen. | Was soll gemessen werden? Wie kann die Änderung am besten gemessen werden? Wie können die Werte miteinander verglichen werden? |
| 3b | MySQL Query Log | Mithilfe des Querylogs langsame Queries auffindig machen. | Welche Queries brauchen lange für die Ausführung? |
| 4b | MySQL Execution Plan | Betroffene Query mit EXPLAIN (Executation Plan) analysieren und optimieren. | Welche Teile der Query sind inperformant? Indexe einführen oder Query umschreiben? |

Tabelle 6: Vorgehen zum Auffinden von Performance-Problemen



## 7.4 Der »Start Quiz«-Vorgang ist langsam

### 7.4.1 Problembeschreibung

Wenn im StudentQuiz ein neuer Fragedurchlauf mit einem Klick auf *Start Quiz* gestartet wird, dauert dies je nach der Anzahl der ausgewählten Fragen so lange, dass der Benutzer diese Wartezeit als Störend empfindet.

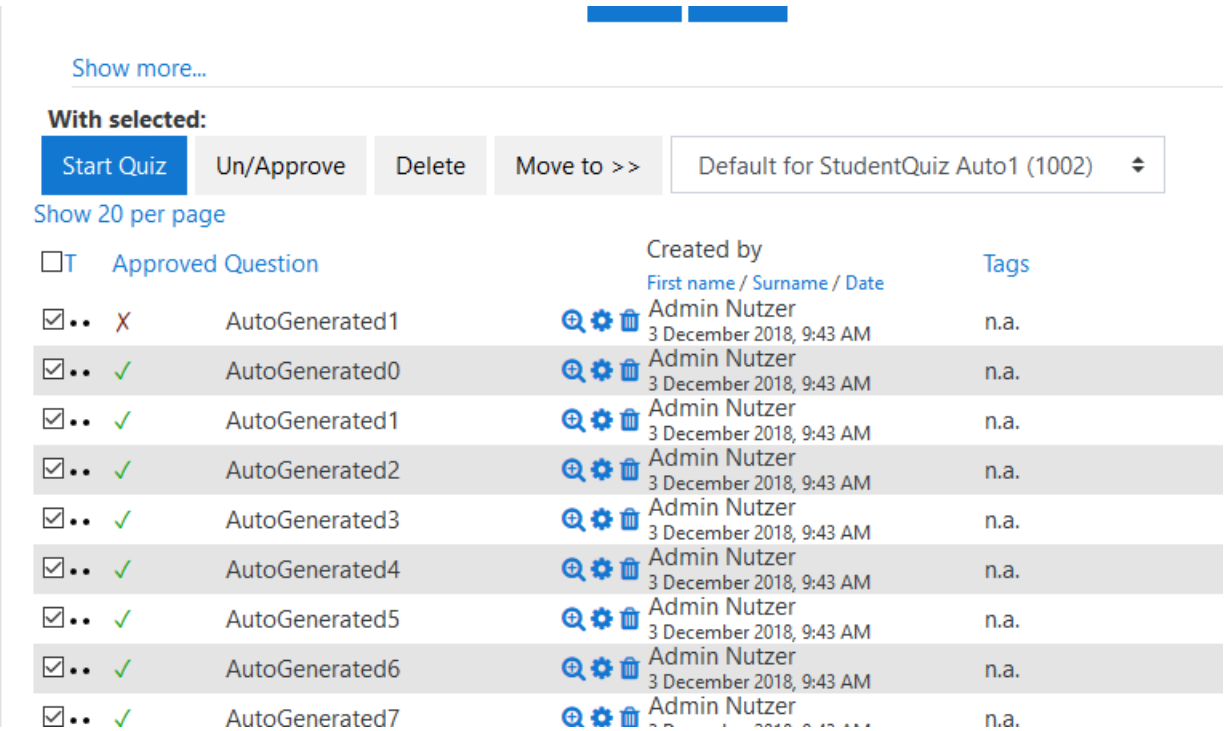


Abbildung 13: Ansicht von view.php mit den ausgewählten Fragen und der *Start Quiz* Schaltfläche

### 7.4.2 Analyse

Performance-Probleme bei Applikationen wie Moodle mit einer separaten Datenbank können aufgrund von ineffizienten Abfragen entstehen. Die Ursache findet sich in der relativ hohen Round-Trip-Time zwischen Datenbank und Applikation (im Vergleich zu der verstrichenen Zeit innerhalb einer Applikation) und den komplexen Abfragen mit vielen *Joins* und *Subqueries*. Um die Ursache von langsamen Seitenaufrufen zu finden, lohnt es sich, den SQL Query Log genauer anzuschauen.

Der Aufruf von *Start Quiz* durch einen Klick auf den Button schickt eine HTTP POST Anfrage an `moodle/mod/studentquiz/view.php`. Dort werden folgende Vorbereitungen getroffen:

1. Aus der HTTP POST Anfrage werden die IDs der vom User ausgewählten *Questions* für den Fragedurchlauf entnommen.
2. Die *Questions* werden aus der Datenbank geladen.
3. Es wird eine *Question Usage* erstellt. Darunter kann man sich einen »Fragedurchlauf« vorstellen.

4. Zusätzlich wird für jede Frage ein Eintrag in den folgenden zwei Tabellen erzeugt:
   - `question_attemps` → Hier wird die Richtige Antwort, die Skala der Punkteverteilung, sowie die Antwort des Benutzers gespeichert.
   - `question_attempt_steps` → Hier wird gespeichert, ob die Frage richtig oder falsch beantwortet und wie viele Punkte dafür vergeben wurden.
5. Ein Eintrag in `studentquiz_attemp` wird erstellt. Dieser dient der Verknüpfung mit dem *question Table*.

**Probleme**

- Je mehr Fragen im Durchlauf beantwortet werden sollen, desto mehr Datenbankeinträge werden generiert. Konkret: Pro Frage zwei Einträge. Das führt zu unnötiger Speicherbelegung.
- Falls eine Frage nicht beantwortet wird, weil der Durchlauf vorher abgebrochen wird, bleiben die `question_attemps` und `question_attempt_steps` in der Datenbank und haben keine weitere Verwendung.
- Der `INSERT INTO` Vorgang wird in einzelnen SQL Abfragen durchgeführt. Dieser Vorgang ist sehr zeitintensiv.

### 7.4.3 Lösung

Neu wird beim Start des *Attempts* nur noch die erste Frage geladen und die dazugehörigen Einträge in `question_attemps` und `question_attemps_steps` angelegt. Damit die oben aufgezählten Datenbankeinträge nicht für alle Fragen umgehend erstellt werden, war es notwendig, ein zusätzliches Feld in der Tabelle `studentquiz_attempt` anzulegen, um die ausgewählten IDs der *Questions* abzulegen. Diese sind notwendig, damit die beiden Einträge in `question_attemps` und `question_attemps_steps` erstellt werden können. Diese Änderung bringt folgende Vorteile mit sich:

- Wenn ein Quiz Durchlauf vorzeitig abgebrochen wird, verbleiben keine unnütze Datenbankeinträge in den Tabellen `question_attemps` und `question_attemps_steps`.
- Die »Start Quiz« Funktion lädt wesentlich schneller.
- Die totale Anzahl der *Inserts* bei einem vollständigen Quiz Durchlauf verringert sich nicht. Dafür verteilen sich die Datenbankanfragen auf einen grösseren Zeitraum und verhindern somit einen plötzlichen Ausschlag der Belastung des Servers.

In Tabelle 7 sind die Messwerte vor und nach der Optimierung aufgeführt. Man sieht eine deutliche Beschleunigung des Vorganges. Abbildung 14 und 15 zeigen die `INSERT INTO` vor beziehungsweise nach der Optimierung.

| AnzahlFragen | Vor der Optimierung | Nach der Optimierung | Unterschied Absolut | Unterschied Relativ |
|---|---|---|---|---|
| 1002 Fragen (Standard) | 2043 ms | 231 ms | 1812 ms | 89% |

Tabelle 7: JMeter Messung (4 Messpunkte) vor und nach der Optimierung

```
1  INSERT INTO mdl_question_usages (contextid,component,preferredbehaviour)
   ↪  VALUES('67','mod_studentquiz','immediatefeedback')
2  -- LOOP for questions
3  INSERT INTO mdl_question_attempts
   ↪  (questionusageid,slot,behaviour,questionid,variant,maxmark,minfraction
4  ,maxfraction,flagged,questionsummary,rightanswer,responsesummary,timemodified)
   ↪  VALUES('3838','1','immediatefeedback','2036','1',1,'0','1','0','Response is
   ↪  False\n','False',NULL,'1544577413')
5  INSERT INTO mdl_question_attempt_steps
   ↪  (questionattemptid,sequencenumber,state,fraction,timecreated,userid)
   ↪  VALUES('50706','0','todo',NULL,'1544577413','2')
6  ...
7  -- END LOOP
8  INSERT INTO mdl_studentquiz_attempt (categoryid,userid,studentquizid,questionusageid)
   ↪  VALUES('24','2','7','3838')
```

Abbildung 14: Beispiel eines Query Logs von *Start Quiz* vor der Optimierung

### 7.4.4 Alternative Lösung

Alternativ hätte das `INSERT INTO` Statement optimiert werden können. Anstatt jeden Wert für `question_attemps` und `question_attemps_steps` in einem einzelnen Statement zu übertragen, hätten alle Einträge in einem Statement übertragen werden können. Diese Lösung konnte jedoch nicht umgesetzt werden, da dafür eine Modifikation des Moodle Core nötig gewesen wäre. Zudem hätte dies das Problem der unnötigen Datenbankeinträgen nicht gelöst.

### 7.4.5 Implementierung

Die Performance-Optimierung wurde umgesetzt und innerhalb des *Github Issues* (`https://github.com/frankkoch/moodle-mod_studentquiz/issues/72`) im *develop4 Branche* veröffentlicht.

```
1 INSERT INTO mdl_question_usages (contextid,component,preferredbehaviour)
  ↪ VALUES('67','mod_studentquiz','immediatefeedback')
2 2018-12-12T01:36:12.215170Z    12003 Query    INSERT INTO mdl_question_attempts
  ↪ (questionusageid,slot,behaviour,questionid,variant,maxmark,minfraction,maxfraction
3 ,flagged,questionsummary,rightanswer,responsesummary,timemodified)
  ↪ VALUES('3869','1','immediatefeedback','2036','1',1,'0','1','0','Response is
  ↪ False\n','False',NULL,'1544578572')
4 2018-12-12T01:36:12.218083Z    12003 Query    INSERT INTO mdl_question_attempt_steps
  ↪ (questionattemptid,sequencenumber,state,fraction,timecreated,userid)
  ↪ VALUES('54196','0','todo',NULL,'1544578572','2')
5 2018-12-12T01:36:12.220945Z    12003 Query    INSERT INTO mdl_studentquiz_attempt
  ↪ (categoryid,userid,studentquizid,questionusageid,ids)
  ↪ VALUES('24','2','7','3869','2036,3337,3338,3339,3340,3341,3342,3343,3344,3345
6 ,3346,3347,3348,3349,3350,3351,3352,3353,3354,3355')
```

Abbildung 15: Beispiel eines Query Logs von *Start Quiz* nach der Optimierung

## 7.5 view.php ist langsam

### 7.5.1 Problembeschreibung

Die Hauptansicht view.php ist die wichtigste Ansicht in der Applikation. Sie ist die erste Ansicht, die beim Öffnen einer StudentQuiz Activity angezeigt wird. Von ihr aus können alle Funktionen erreicht werden. Es ist zentral, dass diese Ansicht möglichst performant ist.

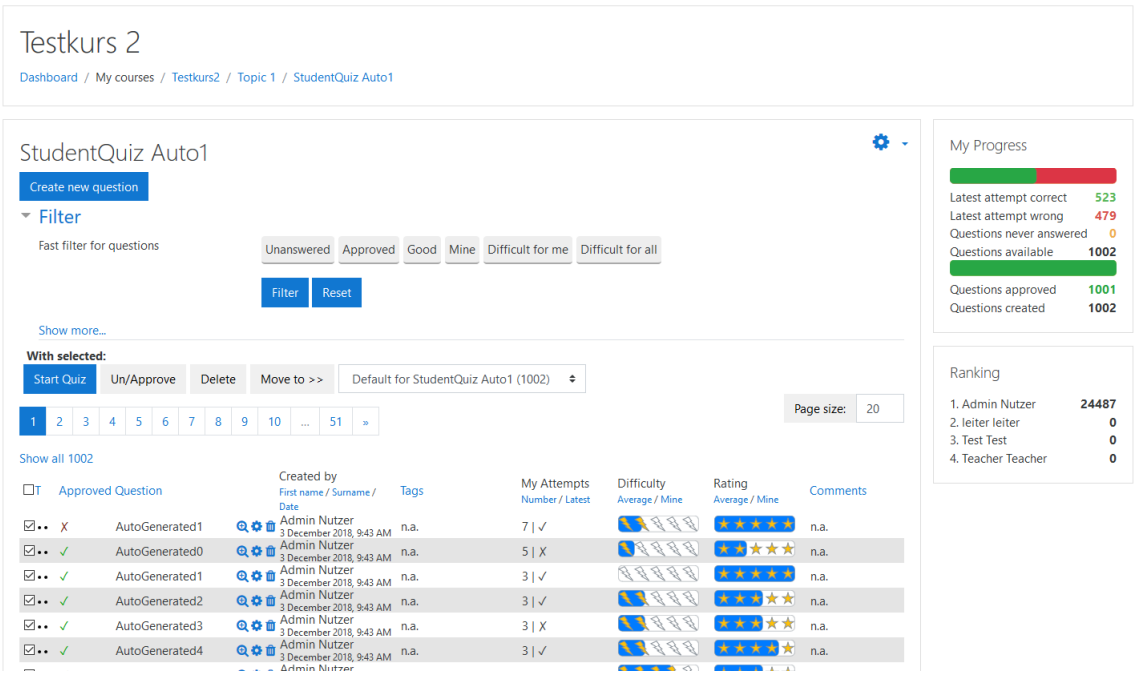


Abbildung 16: Die Ansicht view.php

### 7.5.2 Analyse

Für die Analyse wurde wie in Unterabschnitt 7.3 beschrieben vorgegangen.

**PHP Xdebug Profiler**
In Abbildung 17 ist ersichtlich, welche Funktionen am meisten Zeit benötigen. Interpretiert man diese Ausgabe, stellt man fest, dass zwei Operationen überdurchschnittlich viel Zeit beanspruchen: Einerseits das Rendering der Tabelle, andererseits eine SQL Abfrage.

**Manuelle Analyse: Question Table**
Zuerst wurde der Code eingehend Schritt für Schritt mit dem Debugger analysiert und Auffälligkeiten notiert. Dabei wurde festgestellt:

- Die Question Table im StudentQuiz Modul ist ähnlich aufgebaut wie das Moodle Core Quiz Modul.
- Die Question Table nutzt Moodle Core Code.

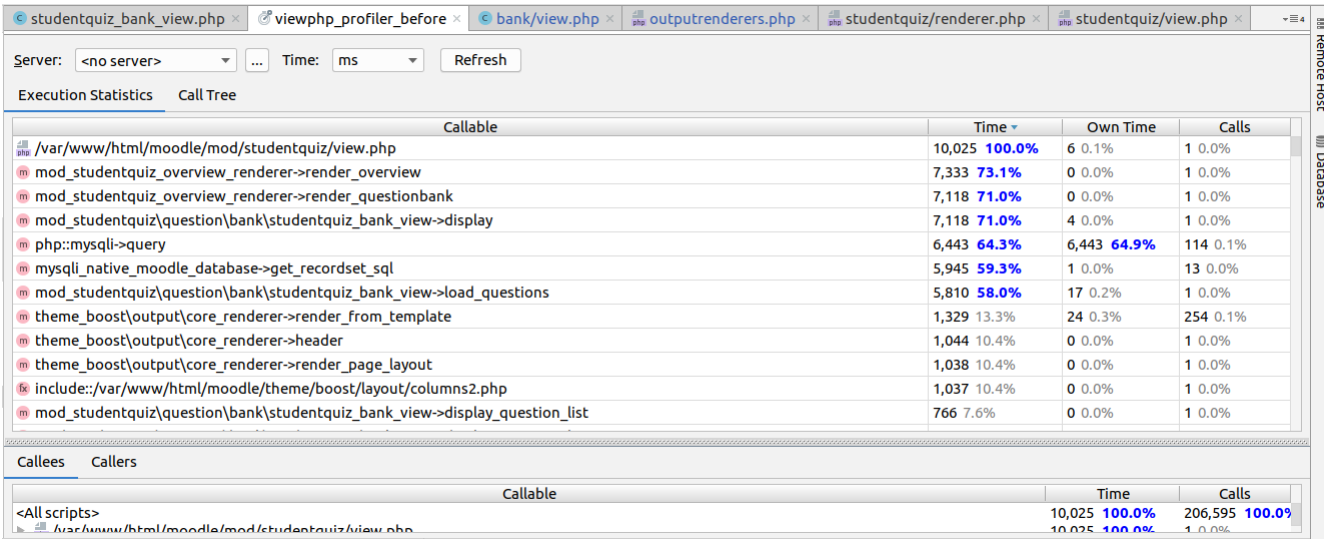


Abbildung 17: Xdebug profiler Ausgabe vor der Optimierung

- Für jede Spalte gibt es eine eigene Klasse fürs Rendering.
- Die *Difficulty Bar* und *Star Bar* sind in HTML definierte SVG Bilder und werden im PHP Code gerendert.

Aus dem Sequenzdiagramm in Abbildung 18 sind folgende Dinge zu entnehmen:

- `bank/view.php` und `column_base` sind Bestandteile von Moodle Core
- Pro Spalte in der Tabelle gibt es im StudentQuiz eine Klasse.
- `*_column` ist je nach Spalte eine andere Klasse (siehe Tabelle 9)
- `render_*_column(...)` ist jeweils eine andere Funktion. Diese befindet sich jedoch in der Datei renderer.php

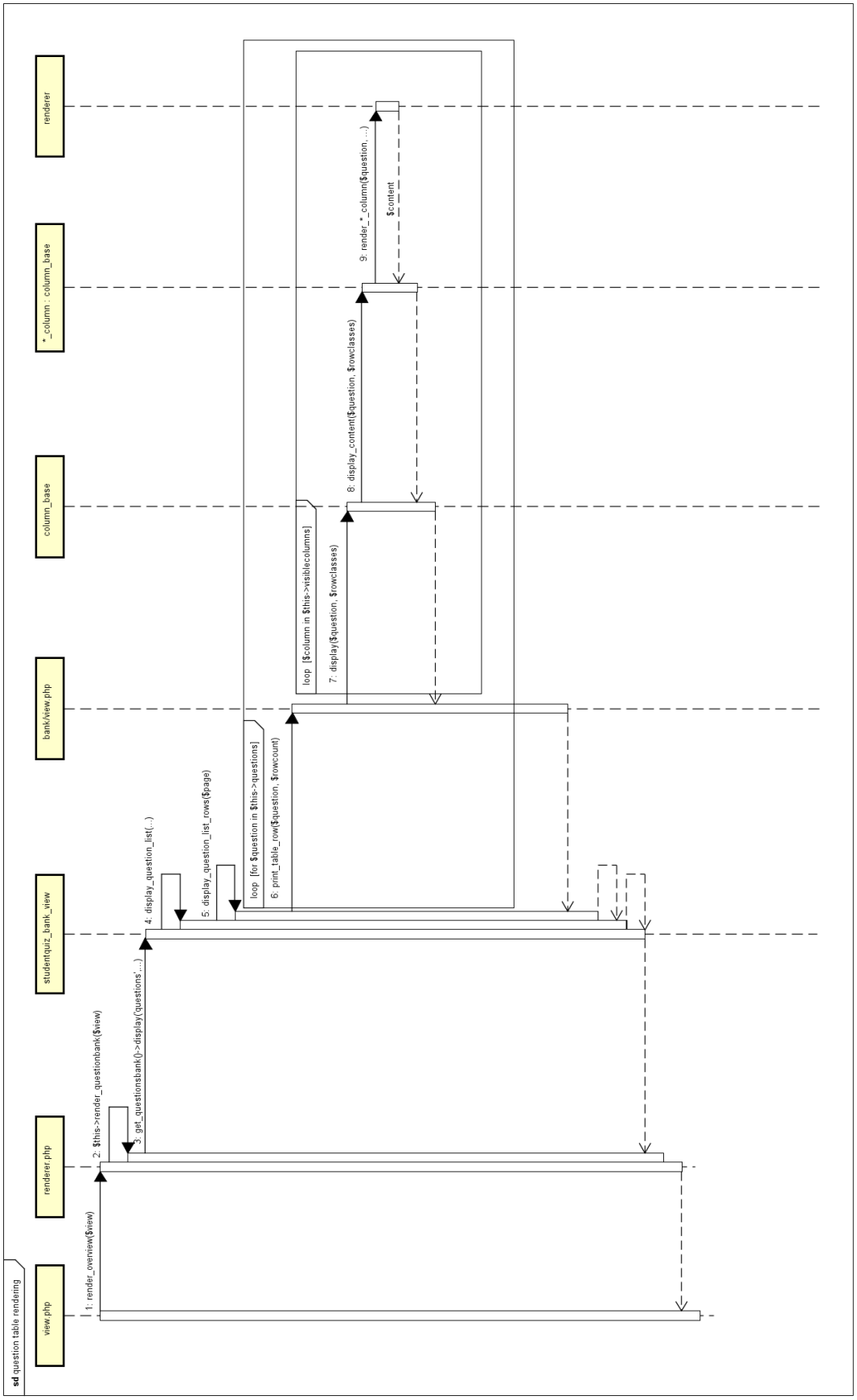


Abbildung 18: Sequenzdiagramm Rendering der Tabelle

Analysiert wurden zuerst die einzelnen Spalten und wie viel Zeit sie im Vergleich zu den anderen brauchen, um dargestellt zu werden (Siehe dazu Tabelle 9). Die ungleichmässige Verteilung der prozentualen Anteile an der Zeit, die für die ganze Spalte verwendet wird, zeigt, welche verhältmässig viel Zeit in Anspruch nehmen. Optimal ist dabei eine möglichst gleichmässige Verteilung zu erreichen. Dafür sollen die Spalten mit den höchsten Prozentanteilen optimiert werden.

Alle Änderungen sollen im Sinne der Moodle Struktur erfolgen und nicht grundlegend davon abweichen. Die Grundstruktur für das Question Table Rendering der verschiedenen Klassen pro Spalte soll deshalb nicht verändert werden. Entschieden wurde Folgendes:

- Codestellen suchen, bei denen manuelles Caching eingefügt werden kann (z.B. wenn Bilder aus dem Template geladen werden).
- Das Rendering von der *Difficulty* und *Star* Vektorgrafiken aus dem Serverseitigen PHP Code entfernen und mithilfe von Javascript in den Browser auslagern.

**Manuelle Analyse: Langsame SQL Queries** Die betroffene SQL Abfrage konnte direkt aus dem SQL Log entnommen werden. Die Analysemethode ist in Unterabschnitt 9.6 erklärt.

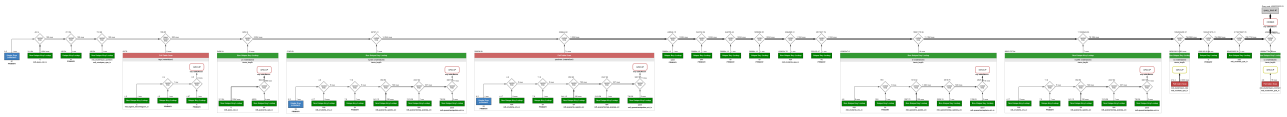

Abbildung 19: MySQL Workbench Query Analyse mit EXPLAIN

### 7.5.3 Umsetzung und Lösung

Der problematische Full Table Scan, der für die langsame Abfrage hauptverantwortlich ist, ist in Abbildung 20 mit einem Kommentar markiert. Dieser konnte durch eine Änderung der Query optimiert werden, ohne einen Index anlegen zu müssen.

**Tabellenrendering**
Der Code des SVG Rendering konnte fast 1:1 in JavaScript übernommen werden.

### 7.5.4 Ergebnis

Nach der Optimierung konnte eine deutliche Verbesserung der Performance festgestellt werden. Dafür wurden mehrere Tests mit JMeter durchgeführt, um die Ergebnisse in Zahlen zu belegen. Dafür wurde ein Frageset mit 1002 Fragen erstellt und damit jeweils 10 Messung durchgeführt. Als Resultat wurde der Durchschnitt genommen.

In den Resultatetabellen Tabelle 10, 11 und 12 ist deutlich ersichtlich, dass die Performance wesentlich verbessert wurde. Die absoluten Werte sind dabei mit Vorsicht zu geniessen, da Sie auf einer Entwicklermaschine (Leistung siehe Unterabschnitt 4.8) ausgeführt wurden. Die damit erreichten Resultate sind

```sql
SELECT
# ...some columns...
FROM
mdl_question q

LEFT JOIN
# Some left joins...

LEFT JOIN
(SELECT
qa.questionid, qas.state mylastattempt
FROM
mdl_studentquiz sq
JOIN mdl_studentquiz_attempt sqa ON sqa.studentquizid = sq.id
JOIN mdl_question_usages qu ON qu.id = sqa.questionusageid
JOIN mdl_question_attempts qa ON qa.questionusageid = qu.id
LEFT JOIN mdl_question_attempt_steps qas ON qas.questionattemptid = qa.id
LEFT JOIN mdl_question_attempt_step_data qasd ON qasd.attemptstepid = qas.id
INNER JOIN (SELECT MAX(qasd.id) maxqasdid FROM mdl_studentquiz sq
JOIN mdl_studentquiz_attempt sqa ON sqa.studentquizid = sq.id
JOIN mdl_question_usages qu ON qu.id = sqa.questionusageid
JOIN mdl_question_attempts qa ON qa.questionusageid = qu.id
LEFT JOIN mdl_question_attempt_steps qas ON qas.questionattemptid = qa.id
LEFT JOIN mdl_question_attempt_step_data qasd ON qasd.attemptstepid = qas.id
WHERE qasd.name = '-submit' AND sq.id = 7
AND sqa.userid = 2
AND (qas.state = 'gradedright'  # Grund für sehr langsamen Full Table Scan
OR qas.state = 'gradedwrong' OR qas.state = 'gradedpartial')
GROUP BY qa.questionid) qasdmax ON qasd.id = qasdmax.maxqasdid
WHERE qasd.name = '-submit') mylatts ON mylatts.questionid = q.id

LEFT JOIN
# more left joins...

WHERE q.parent = 0 AND q.hidden = 0    AND q.category = '24' ORDER BY q.timecreated DESC
```

Abbildung 20: Diese SQL Query ladet die Fragen für die Tabelle aus der Datenbank.

durchaus zufriedenstellend, weitere Optimierungen sind aber möglich und werden im nachfolgenden Kapitel beschrieben.

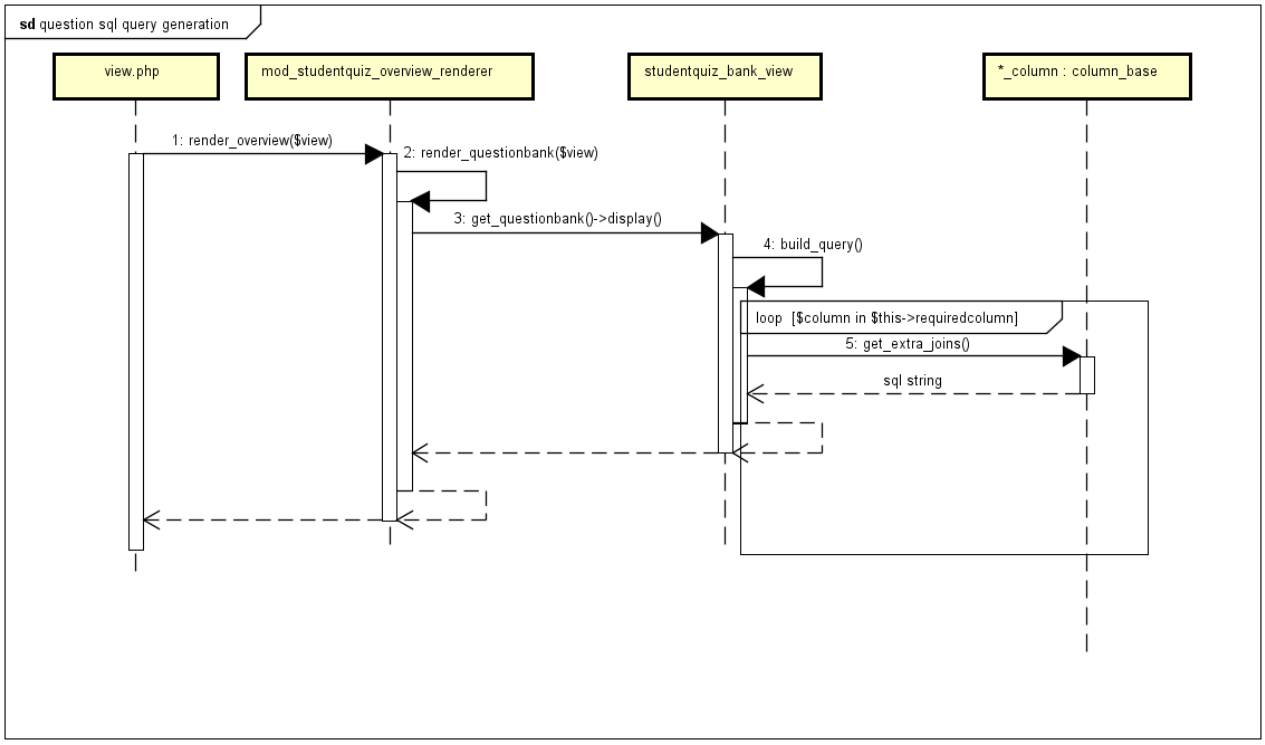


Tabelle 8: Sequenzdiagramm zeigt die generische Generierung der SQL Query für die Abfrage der Daten für die Question Table

```
AND (qas.state = 'gradedright'  # Grund für sehr langsamen Full Table Scan
OR qas.state = 'gradedwrong'
OR qas.state = 'gradedpartial')
# ersetzt durch
AND qas.fraction is not null
```

Abbildung 21: Vor und nach der Optimierung der SQL Query

| Id | Name | Class | Vor Optimierung Abs (ns) | Vor Optimierung Rel | Nach Optimierung Abs (ns) | Nach Optimierung Rel |
|---|---|---|---|---|---|---|
| 0 | Checkbox | \core_question\bank\checkbox_column | 0.058 ns | 1% | 0.024 ns | 1% |
| 1 | Question Type | \core_question\bank\question_type_column | 1.016 ns | 20% | 0.730 ns | 24% |
| 2 | Approved | \mod_studentquiz\bank\approved_column | 0.314 ns | 6% | 0.269 ns | 9% |
| 3 | Name | \mod_studentquiz\bank\question_name_column | 0.093 ns | 2% | 0.079 ns | 3% |
| 4 | View | \mod_studentquiz\bank\preview_column | 1.124 ns | 21% | 0.513 ns | 17% |
| 5 | Edit | \core_question\bank\edit_action_column | 0.386 ns | 7% | 0.432 ns | 14% |
| 6 | Delete | \core_question\bank\delete_action_column | 0.347 ns | 7% | 0.359 ns | 12% |
| 7 | Name + Date | \mod_studentquiz\bank\anonym_creator_name_column | 0.436 ns | 8% | 0.305 ns | 10% |
| 8 | Tags | \mod_studentquiz\bank\tag_column | 0.087 ns | 2% | 0.076 ns | 2% |
| 9 | My Attempts | \mod_studentquiz\bank\practice_column | 0.074 ns | 1% | 0.074 ns | 2% |
| 10 | Difficulty | \mod_studentquiz\bank\difficulty_level_column | 0.620 ns | 12% | 0.046 ns | 2% |
| 11 | Rating | \mod_studentquiz\bank\rate_column | 0.561 ns | 11% | 0.044 ns | 1% |
| 12 | Comments | \mod_studentquiz\bank\comment_column | 0.072 ns | 1% | 0.070 ns | 2% |
| | | **Total** | **5.188 ns** | | **3.020 ns** | |
| | | Overall Improvement (Pro Zeile) | 2.168 ns | 42% | | |

Tabelle 9: Zeitoptimierung des Tabellenrenderings der Ansicht view.php

| AnzahlangezeigterFragen | Vor der Optimierung | Nach der Optimierung | Unterschied Absolut | Unterschied Relativ |
|---|---|---|---|---|
| 20 Fragen (Standard) | 6439 ms | 1350 ms | 5089 ms | 79% |
| 1002 Fragen | 11001 ms | 4490 ms | 6511 ms | 59% |
| Differenz (982 Fragen) | 4562 ms | 3140 ms | 1422 ms | 31% |

Tabelle 10: JMeter Messung (10 Messpunkte) vor und nach der Optimierung mit drei Attempts

| AnzahlangezeigterFragen | Vor der Optimierung | Nach der Optimierung | Unterschied Absolut | Unterschied Relativ |
|---|---|---|---|---|
| 20 Fragen (Standard) | 10941 ms | 5369 ms | 5572 ms | 49% |
| 1002 Fragen | 22619 ms | 8317 ms | 14302 ms | 63% |
| Differenz (982 Fragen) | 11687 ms | 2948 ms | 8933 ms | 76% |

Tabelle 11: JMeter Messung (10 Messpunkte) vor und nach der Optimierung mit 31 Attempts

| AnzahlangezeigterFragen | Vor der Optimierung | Nach der Optimierung | Unterschied Absolut | Unterschied Relativ |
|---|---|---|---|---|
| 20 Fragen (Standard) | 60069 ms | 28841 ms | 31228 ms | 51% |

Tabelle 12: JMeter Messung (10 Messpunkte) vor und nach der Optimierung mit 31 Attempts und 10 Threads Parallel

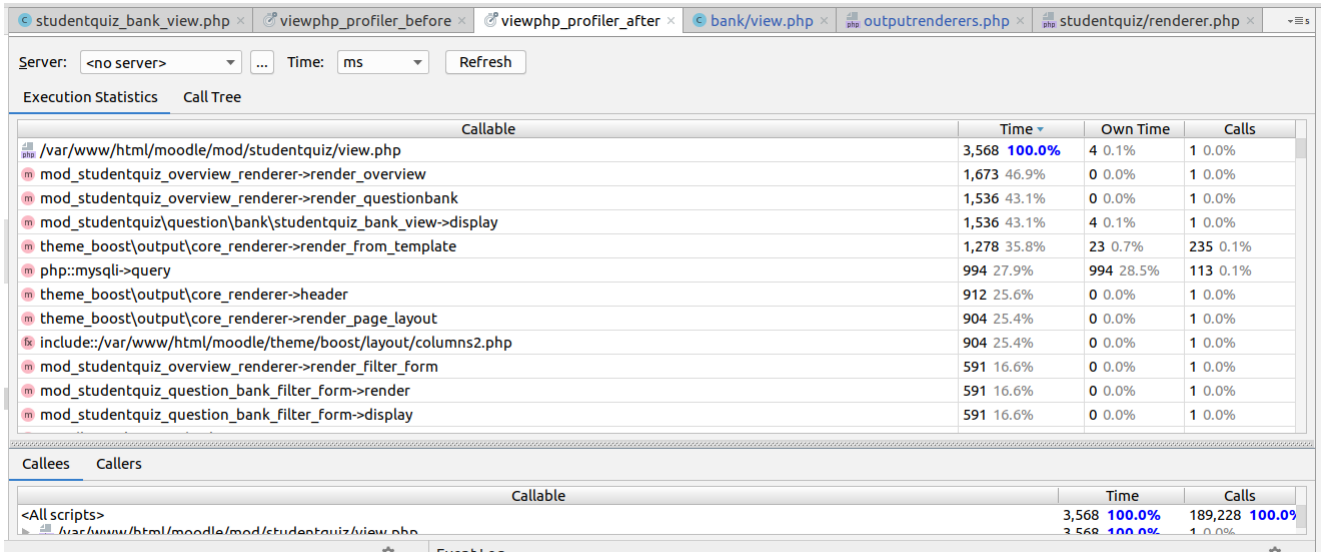


Abbildung 22: XDebug Profiling nach den Optimierungen

## 7.6 Einführung von Aggregatswerten

### 7.6.1 Problemstellung

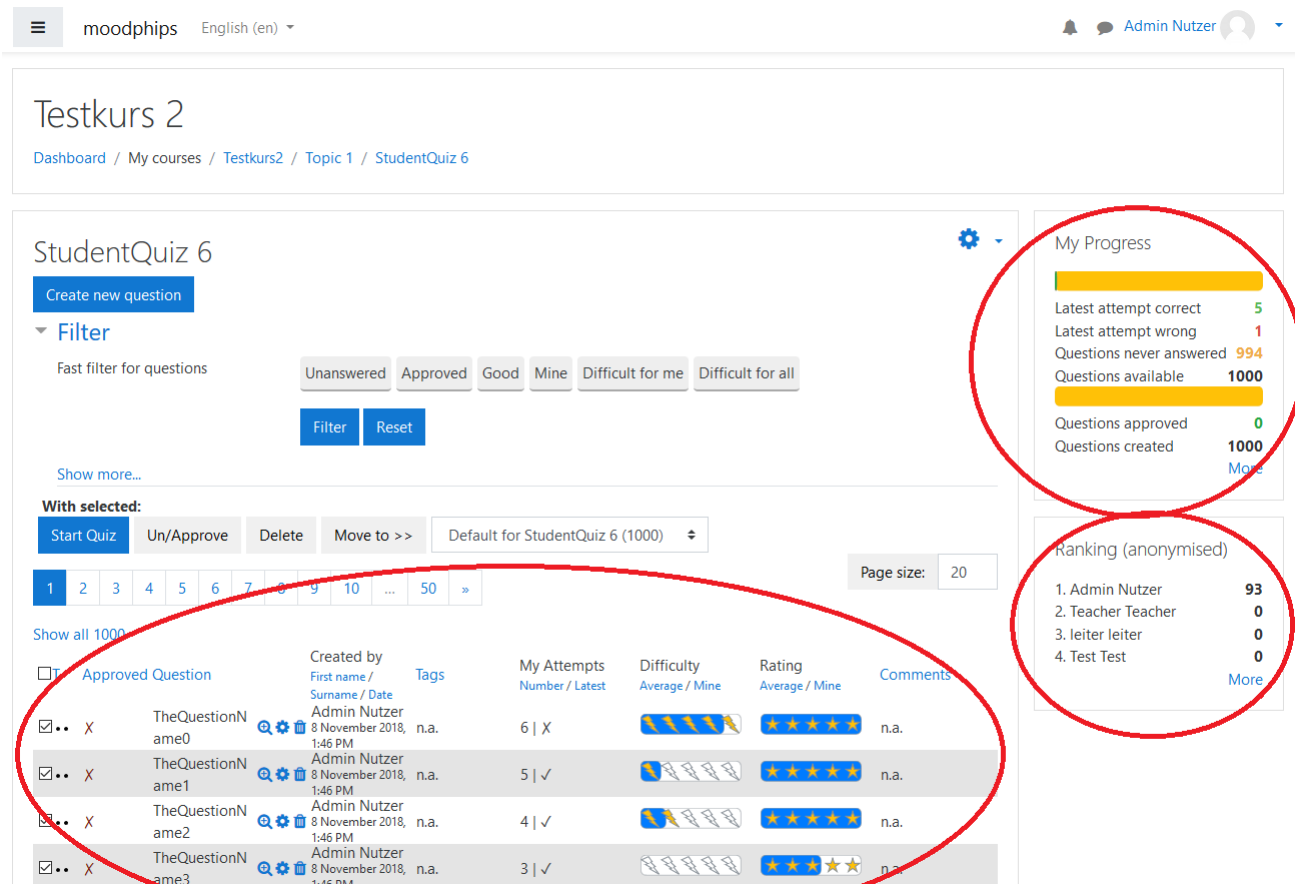


Abbildung 23: Die Ansicht view.php mit den drei zentralen Elementen

In Abbildung 23 sind die drei wichtigsten Elemente der Hauptansicht view.php dargestellt: Die Fragetabelle, »My Progress« und »Ranking«. Hinter jedem dieser Elemente steckt eine komplexe SQL Query. Alle diese SQL Queries bedienen sich der Datenbanktabellen `question_attempts` und `question_attempt_steps`. In Abbildung 24 ist eine beispielhafte Berechnung für die grosse Anzahl von generierten Datenbankeinträgen zu sehen.

$$1 * StudentQuizInstanz * 100 Fragen * 100 Versuche * 10 User * 2 Tabellen = 200'000 Datenbankeinträge$$

Abbildung 24: Berechnung der Anzahl Datenbankeinträge pro StudentQuiz Activity in den Tabellen `question_attempts` und `question_attempt_steps`.

In `question_attempts` und `question_attempt_steps` sind Informationen zu jeden einzelnen Versuchen, pro StudentQuiz Instanz, pro Frage und pro Benutzer abgespeichert. Dieser Detailgrad an Informationen wird von den Elementen in Abbildung 23 nicht benötigt. Gebraucht werden nur die folgenden Informationen pro Benutzer und StudentQuiz:

- Anzahl der Versuche
- Anzahl der erfolgreichen Versuche
- Anzahl der gescheiterten Versuche
- Information ob der letzte Versuch erfolgreich war

Die Bachelorarbeit »Moodle Plugin StudentQuiz« von Denis Manente und Simon Schaefer[1] beschreibt im Abschnitt 4.9.2 eine neue Tabelle, die bereits in das Datenbankmodell von Studentquiz integriert wurde, aber bisher noch keinen Einsatz fand.

### 7.6.2 Analyse

Die Analyse gestaltet sich insofern aufwendig, dass hinter den in Abbildung 23 aufgeführten Elementen komplexe SQL Abfragen stehen.

Um die Tabelle `studentquiz_progress` abzufüllen, welche die aggregierten Werte halten soll, müssen während der Durchführung des Quizzes die Informationen laufend eingetragen werden. Die Gründe, weshalb dies genau zu diesem Zeitpunkt im Prozess geschehen muss, sind die Folgenden:

- Ein Quizdurchgang wird nur durch einen einzelnen Benutzer ausgeführt. Der Prozess der Frageprüfung mit dem Klick auf »Check« erfolgt nur ein einziges Mal. Zu diesem Zeitpunkt erfolgt keine gleichzeitige Aktualisierung der Werte an einer anderen Stelle.
- Die Erstellung der aggregierten Werte am Ende des Quizdurchganges ist ausgeschlossen, weil der Benutzer das Quiz nicht zwingend beendet.
- Die nachträgliche Berechnung der aggregierten Werte aus den Tabellen `question_attempts` und `question_attempt_steps` gestaltet sich sehr aufwändig. Zudem müssten diese Tabellen dann weiterhin geladen werden.

**Datenmigration**
Es gilt zu beachten, dass die bestehenden Daten in `question_attempts` und `question_attempt_steps` in die neue Tabelle migriert werden müssen.

**Risiko der Datenmigration**
Wie in der Berechnung in Abbildung 24 gezeigt, muss bei einer bestehenden StudentQuiz Installation, welche auf die neue Version mit den aggregierten Werten aktualisiert wird, beachtet werden, dass es sich um eine sehr grosse Datenmenge handelt. Wenn dieser Prozess schief läuft, kann es zu einem sehr grossen und schwierig zu behebenden Datenchaos kommen.

### 7.6.3 Lösungsvorschlag

Mit der Berücksichtigung der oben aufgeführten Punkte wird der folgende Integrationsplan vorgeschlagen.

1. Die Aggregatswerte werden vorerst nur für neue StudentQuiz Instanzen implementiert. Dafür wird in der Tabelle mit den StudentQuiz Instanzen `studentquiz` eine zusätzliche Spalte namens `aggregated` eingefügt, die markiert, ob eine Instanz die Aggregierten Werte verwendet. Ist dies nicht der Fall, werden die bestehenden SQL Abfragen genutzt und die `studentquiz_progress` Tabelle während des Durchlaufs nicht abgefüllt. Der default Value wird bei der Anpassung der Datenbank während dem Upgradevorgang auf die neue StudentQuiz Version auf `false` gesetzt. Neu erstellte StudentQuiz Instanzen verwenden hier jedoch den Wert `true` und nutzen die neuen aggregierten Werte.
2. Implementation der Migrationsfunktionen innerhalb einer StudentQuiz Instanz. Mit einer zusätzlichen Schaltfläche im Administrationsmenü können einzelne StudentQuiz Instanzen auf die aggregierte Variante migriert werden.
3. Automatische Migration innerhalb von upgrade.php oder mit einer Schaltfläche für moodle Administratoren (manuelle Ausführung der Migration).
4. Löschen von alten attempt und attempt_steps Daten.

Da bei diesem Migrationsvorgang Programmierfehler zu erwarten sind und er deshalb sehr heikel ist, sollen diese Schritte nacheinander ausgeführt werden. Zwischen den einzelnen Releases soll eine Veröffentlichung stattfinden. Damit kann die Community testen, ob die neuen Aggregatswerte auch wirklich funktionieren. Der letzte Schritt im Integrationsplan darf erst nach genauer Prüfung durchgeführt werden, da er der einzige Schritt ist, der nicht mehr Rückgängig gemacht werden kann.

### 7.6.4 Umsetzung

Die Schritte 1 und 2 sind innerhalb dieser Bachelorarbeit umgesetzt worden. Alle Commits sind mit Github Issue (`https://github.com/frankkoch/moodle-mod_studentquiz/issues/73`) verknüpft.

### 7.6.5 Ergebnis

Um die Performanceverbesserung zu messen wurden zwei Performance-Tests durchgeführt. Der Erste wurde mit nur einem einzelnen gleichzeitigen Zugriff aufgeführt (siehe Tabelle 13). Der Zweite wurde mit vielen Parallelzugriffen ausgeführt (siehe Tabelle 14). Zudem wurde ein Testzyklus verwendet, der die Threads über einen definierten Zeitraum zuschaltet, die Last vorübergehend beibehält und danach kontinuierlich herunterfährt (Abbildung 25).

| | |
|---|---|
| Anzahl Fragen | 1000 |
| Anzahl Durchläufe | 10 |
| Anzahl User | 4 |
| Anzahl parallele Threads | 1 |
| Anzahl angezeigter Fragen | 20 |
| Anzahl Messungen (Samples) | 10 |
| Durchschnittliche Antwortzeit vor Optimierung | 20945 ms |
| Durchschnittliche Antwortzeit nach Optimierung | 1669 ms |
| Absoluter Unterschied | 19276 ms |
| Relativer Unterschied | 92 % |

Tabelle 13: JMeter Messung mit aggregierten Daten

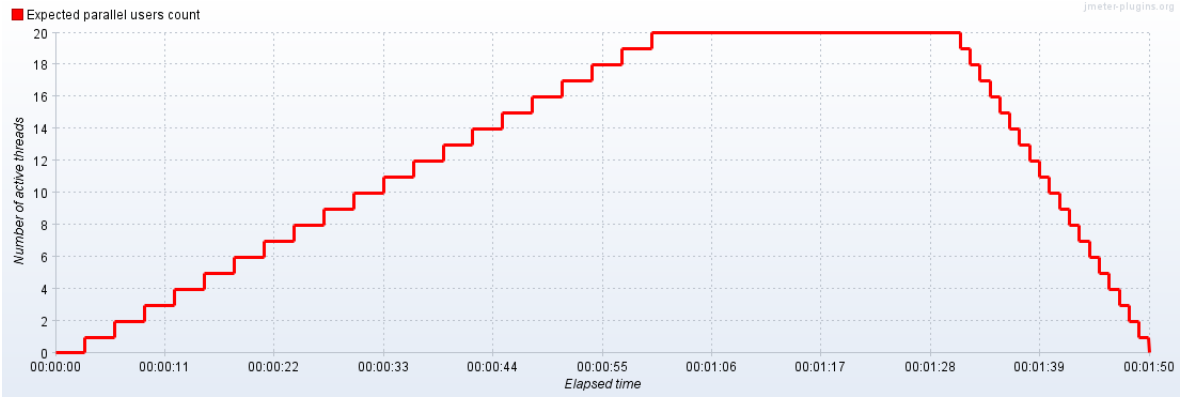


Abbildung 25: JMeter Messung Testzyklus für die Parallelzugriffe

| | |
|---|---|
| Anzahl Fragen | 1000 |
| Anzahl Durchläufe | 10 |
| Anzahl User | 4 |
| Anzahl parallele Threads | 20 |
| Startdauer | 60s |
| Last halte Dauer | 30s |
| Herunterfahrdauer | 20s |
| Anzahl angezeigter Fragen | 20 |
| Anzahl Messungen (Samples) vor Optimierung | 20 |
| Durchschnittliche Antwortzeit vor Optimierung | 95821 ms |
| Anzahl Messungen (Samples) nach Optimierung | 110 |
| Durchschnittliche Antwortzeit nach Optimierung | 1871 ms |
| Absoluter Unterschied | 93950 ms |
| Relativer Unterschied | 98 % |

Tabelle 14: JMeter Messung mit aggregierten Daten mit Parallelzugriffen

# 8 Technischer Bericht - Ergebnisdiskussion

Die Ergebnisse wurden bereits in den jeweiligen Kapiteln aufgeführt. Abschliessend lässt sich sagen, dass in verschiedenen Bereichen Beiträge zur Weiterentwicklung geleistet werden konnten. So konnte ein Anforderungskonzept zu Moodle Mobile erstellt werden, dass für die Implementierung genutzt werden kann. Weiterhin wurden viele Verbesserungen am Code vorgenommen, durch die die Aufnahme in den Moodle Core verbessert wurde.

Am meisten fallen die Verbesserungen der Performance ins Gewicht. Das zweite Konzept, dass die SQL Abfrage von `view.php` verbesserte, hat zwar die Antwortzeit verringert, jedoch nicht unter die anfangs geforderten 2 Sekunden gebracht. Erst die zusätzliche Einführung der Aggregatwerte brachte die geforderte Verbesserung mit sich.

Für den Benutzer ist somit die Wartezeit (entspricht einer Wartezeit von über 2 Sekunden) wesentlich verringert worden. Bei der Hauptansicht, sowie beim Starten seines Quizes erhält der Benutzer innerhalb von kürzester Zeit eine Serverantwort und wird in seinem Workflow bzw. »Lernflow« nicht unterbrochen.

Trotz dem Refactoring wurde die Grundstruktur der Applikation nicht wesentlich verändert. Das Tabellen Rendering verwendet weiterhin die Codebasis der *Questions Library* von Moodle Core.

## 8.1 Messresultate view.php insgesamt

Um die schrittweise Verbesserung der Performance der Ansicht `view.php` aufzuzeigen, wurde nochmals eine einfache Messung über alle zwei Optimierungsphasen hinweg durchgeführt. In Tabelle 15 sind alle Testdetails, sowie Resultate aufgeführt und in Abbildung 26 werden diese nochmals grafisch dargestellt.

| | |
|---|---:|
| Anzahl Fragen | 1000 |
| Anzahl Durchläufe | 10 |
| Anzahl User | 4 |
| Anzahl parallele Threads | 1 |
| Anzahl angezeigter Fragen | 20 |
| Durchschnittliche Antwortzeit vor Optimierung | 14798 ms |
| Durchschnittliche Antwortzeit Erste Optimierung | 10285 ms |
| Durchschnittliche Antwortzeit Zweite Optimierung | 885 ms |

Tabelle 15: JMeter Messung von view.php über alle Optimierungen hinweg

## 8.2 Ausblick

Die Umsetzung der neuen *Question States*, der *Question Visibility* und der Integration in Moodle Mobile ist ein wichtiger nächster Schritt. Die Gründe dafür wurden bereits in dieser Arbeit aufgeführt.

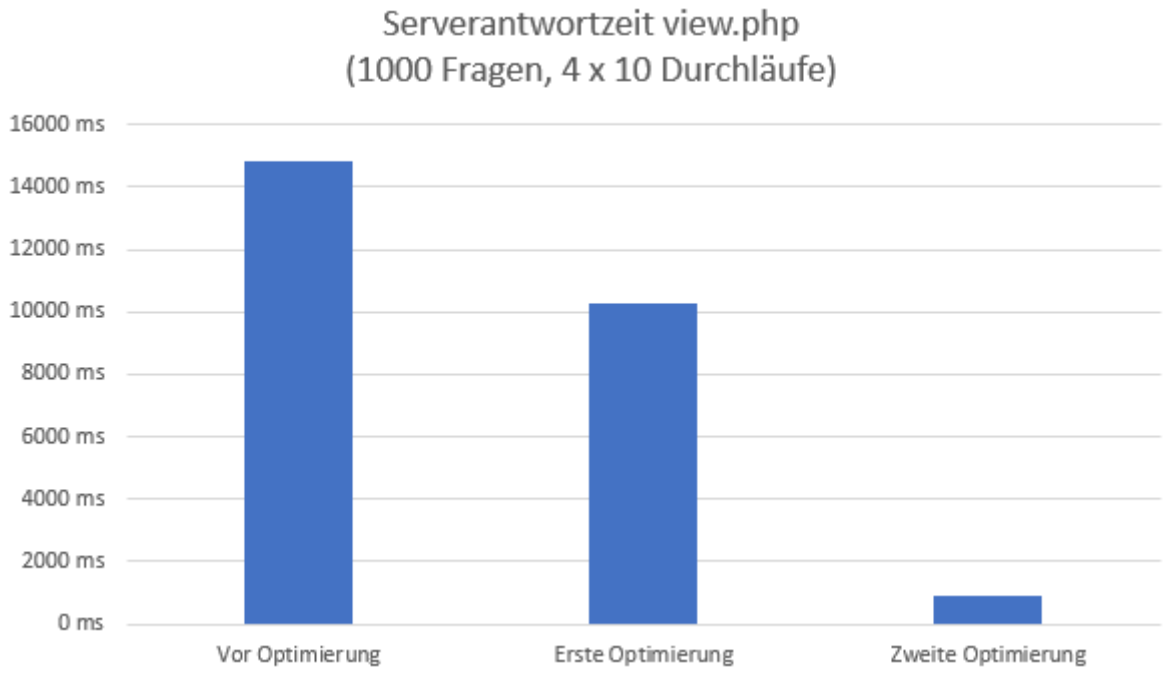


Abbildung 26: JMeter Messung von view.php über alle Optimierungen hinweg

# 9 Softwaredokumentation

## 9.1 Installation von moodle

Die Anleitung auf `https://docs.moodle.org/36/de/Installation_von_Moodle` erklärt die Installation von moodle sehr detailliert.

## 9.2 Plugin Installation

Die Plugin-Installation setzt eine funktionierende moodle Instanz mit der Version 3.4, 3.5 oder 3.6 voraus (Siehe vorhergehenden Abschnitt).

### 9.2.1 Standardinstallation über moodle plugin directory

Der offizielle Release von StudentQuiz kann auf `https://moodle.org/plugins/mod_studentquiz` heruntergeladen und gemäss der offiziellen Anleitung von moodle `https://docs.moodle.org/36/de/Plugins_installieren` werden.

### 9.2.2 Manuelle Installation von develop4

Um die Entwicklungsversion dieser Bachelorarbeit zu verwenden, muss StudentQuiz manuell installiert werden.

- Ins moodle Verzeichnis `mod/` wechseln.
- `git clone https://github.com/frankkoch/moodle-mod_studentquiz.git -b develop4 -o studentquiz`
- `http://host/moodle/admin/index.php` aufrufen und Plugin Installation starten.

## 9.3 Benutzeranleitung

Eine ausführliche Benutzeranleitung kann auf `https://docs.moodle.org/36/en/StudentQuiz_module` gefunden werden.

## 9.4 Datenbanktabellen

Die Abbildung 27 zeigt das Datenmodell zum Zeitpunkt der StudentQuiz Version 3.2.1. Es sind alle relevanten Tabellen von StudentQuiz und moodle core aufgeführt. `mdl_` ist dabei ein von der moodle Instanz Abhängiger *Table prefix*.

Alle Analysemethoden basieren auf einem x64 basierten lubuntu mit der Version 18.04. Dabei sind folgende Packete in der neusten Version installiert:

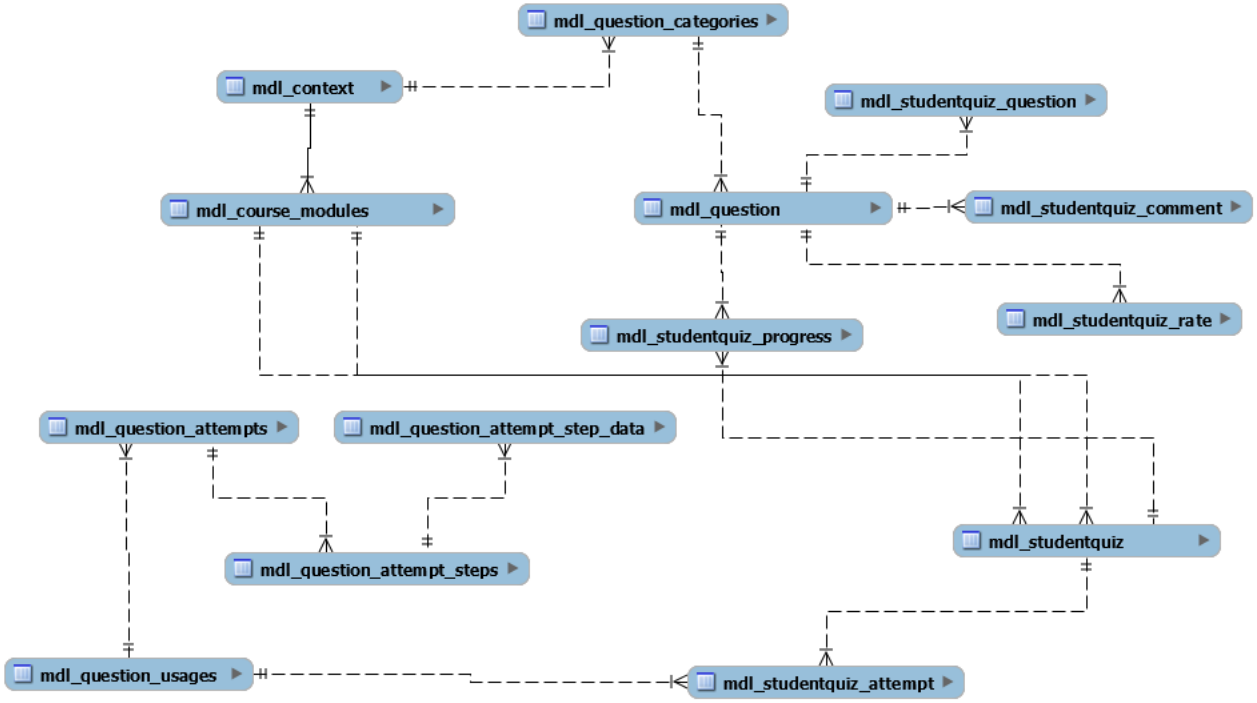


Abbildung 27: Das Datenmodell von StudentQuiz in der Version 3.2.1

- apache2
- code
- composer
- curl
- git
- google-chrome-stable
- libapache2-mod-php7.2
- libgnome-keyring-common
- libgnome-keyring-dev
- mysql-client
- mysql-server
- net-tools
- nodejs
- openssh-server
- oracle-java8-installer
- php7.2
- php7.2-curl
- php7.2-gd
- php7.2-intl

- php7.2-ldap
- php7.2-mbstring
- php7.2-mysql
- php7.2-pspell
- php7.2-soap
- php7.2-xml
- php7.2-xmlrpc
- php7.2-zip
- phpcpd
- phpunit
- php-xdebug
- vim

## 9.5 Anforderungen und Design im Detail

- Es hat keine Änderungen der Funktionalität im Rahmen dieser Arbeit gegeben - Alle neue Funktionaliltäten sich nur Lösungsvorschläge. Anleitungen sind deshalb nicht vorhanden. -

## 9.6 SQL Query Log

Um die SQL Queries, die an die MySQL Datenbank gesendet werden live mitzuverfolgen und zu protokollieren eignet sich die folgende Methode. Zuerst muss die Protokollierung aktiviert werden. Dies erfolgt direkt durch die Eingabe der SQL Befehle (Siehe: Abbildung 28).

```
1 SET global general_log = 1;
2 # Abfrage des Pfades der Logdatei
3 SHOW Variables like "%general_log%"
```

Abbildung 28: SQL Commands um die Protokollierung zu aktivieren

Es gilt jedoch zu beachten, dass sich diese Methode nur dann eignet, wenn gleichzeitig keine anderen Zugriffe auf die Datenbank erfolgen, denn es werden alle SQL Queries gelogget. Bei der MySQL Datenbank heisst diese Funktionalität `General Query Log`.[8]

Unter Linux kann eine Konsole geöffnet werden und die Befehle mitverfolgt werden (siehe Abbildung 29).

```
1 tail -f /var/lib/mysql/philipp-VirtualBox.log
```

Abbildung 29: Echtzeitmitverfolgung der ausgeführten SQL queries

Wenn man die Queries in Echtzeit mitverfolgt fallen besonders langsame SQL Queries auf, da bei diesen eine längere Wartezeit erfolgt bis zur nächsten Query. So lassen sich manuell ohne grossen Aufwand langsame SQL Queries erkennen.

## 9.7 xDebug Profiler Snapshot

Um die Performance des PHP Codes zu beurteilen eignet sich eine xDebug Profiler Snapshot.

Mit folgenden Schritten lässt sich ein solcher durchführen:

- Mithilfe von `phpinfo()` prüfen ob xdebug installiert ist und `xdebug.profiler_enable` vorhanden ist.
- In der Datei /etc/php/7.2/apache2/conf.d/20-xdebug.ini werden verschiedene Zeilen eignefügt oder angepasst. (siehe Abbildung 30)
- Prüfen ob der Pfad `profiler_output_dir` vorhanden ist, bei Bedarf erstellen und die entsprechende Dateisystemrechte setzen.
- `apache2` neustarten.
- Die gewünschte PHP Webseite im webbrowser aufrufen. **Wichtig:** Das Laden der Webseite dauert länger als üblich.
- Die entsprechende Datei aus dem in `profiler_output_dir` definierten Pfad entnehmen. **Es ist meistens die grösste Datei**.
- Die Datei mit phpstorm öffnen (*Tools* → *Analyze Xdebug Profiler Snapshot*).

```
1 xdebug.profiler_enable = 1
2 xdebug.profiler_output_dir = /var/lib/apache2/profiling/
3 xdebug.profiler_output_name = cachegrind.out.%p%r
```

Abbildung 30: Benötigte Zeilen in der Datei 20-xdebug.ini für das Profiling

## 9.8 xDebug mit phpStorm

Die Anleitung von JetBrains[13] erklärt detailiert, wie sich phpstorm mit xdebug nutzen lässt. Dazu relevant ist insbesondere die Konfiguration in 20-xdebug.ini (Siehe Abbildung 31). Danach muss apache2 neugestartet werden. Alle weiteren Schritte können von der Anleitung entnommen werden.

## 9.9 Generieren von Testdaten

Zuerst wird mithilfe SQL Query Log Methode (siehe Unterabschnitt 9.6) alle `INSERT INTO` und `UPDATE` Befehle gesammelt. Dazu spielt man den Vorgang mit dem man die Testdaten sonst manuell Anlagen würde manuell direkt in der Applikation durch Abbildung 32.

```
xdebug.show_error_trace = 1
xdebug.idekey = PHPSTORM
xdebug.show_error_trace = 1
xdebug.remote_autostart = 1
xdebug.file_link_format = phpstorm://open?%f:%l
xdebug.remote_enable = 1
xdebug.remote_port = 9000
xdebug.remote_host = "localhost"
```

Abbildung 31: Benötigte Zeilen in der Datei 20-xdebug.ini für das Debugging mit PhpStorm

- Erstellen von Fragen (Typ True/False)
- Erstellen Bewertungen
- Erstellen von `Question attempts` und den zugehörigen Daten

```
tail -f /var/lib/mysql/philipp-VirtualBox.log | grep "INSERT INTO"
tail -f /var/lib/mysql/philipp-VirtualBox.log | grep "UPDATE"
```

Abbildung 32: Benötigte Zeilen in der Datei 20-xdebug.ini für das Debugging mit PhpStorm

Im beilligenden PHP Projekt `phpscripts` ist ein Beispiel vorhanden, dass folgende Funktionen abdeckt:

- Hinzufügen von True/False Fragen zu einer bestehenden StudentQuiz Instanz.
- Hinzufügen von Quiz Durchgängen in einer bestehenden StudentQuiz Instanz für einen bestehenden User
- Hinzufügen von Bewertung zu Fragen zu einer bestehenden StudentQuiz Instanz für einen bestehenden User

## 9.10 JMeter

Apache JMeter `https://jmeter.apache.org/` ist eine Open Source Applikation zur Durchführung von Belastungstests und zur Messung der Performance.

### 9.10.1 Installation

JMeter kann von der oben aufgeführten Webseite heruntergeladen, entpackt und danach direkt gestartet werden. Um die beiliegenden Testpläne (Dateien mit der Endung `.jmx`) zu öffnen, werden zusätzliche Plugins benötigt. Installieren Sie dazu den Plugin Manager gemäss `https://jmeter-plugins.org/wiki/PluginsManager/`. Danach können Sie mithilfe des Plugin Managers das Plugin mit dem Namen *Custom Thread Groups* installieren (Siehe `https://jmeter-plugins.org/wiki/UltimateThreadGroup/`).

### 9.10.2 Ausführung

Die beiliegenden `.jmx` Dateien können mit JMeter geöffnet und den Bedürfnissen nach konfiguriert werden.

Details zu der Konfiguration wurden aus der Bachelorarbeit »Moodle Plugin StudentQuiz« von Denis Manente und Simon Schaefer[1] entnommen.

## Literatur

[1] Manente, Denis and Schaefer, Simon (2017) Moodle Plugin StudentQuiz. Bachelor thesis, HSR Hochschule für Technik Rapperswil.
*Vorhergehende Bachelorarbeit von Denis Manente und Simon Schaefer über das Moodle Plugin StudentQuiz.*
URL: https://eprints.hsr.ch/620/
(besucht am 01.11.2018).

[2] Moodle Mobile release notes
URL: https://docs.moodle.org/dev/Moodle_Mobile_release_notes
(besucht am 16.11.2018)

[3] Bundesamt für Statistik - Mobile Internetnutzung
URL: https://www.bfs.admin.ch/bfs/de/home/statistiken/kultur-medien-informationsgesellschaf
informationsgesellschaft/gesamtindikatoren/haushalte-bevoelkerung/
mobile-internetnutzung.assetdetail.5306975.html
(besucht am 16.11.2018)

[4] Moodle Mobile
URL: https://docs.moodle.org/dev/Moodle_Mobile
(besucht am 22.11.2018)

[5] Mobile support for plugins
URL: https://docs.moodle.org/dev/Mobile_support_for_plugins
(besucht am 22.11.2018)

[6] GDPR - General Data Protection Regulation URL: https://eugdpr.org/
(besucht am 10.12.2018)

[7] Moodle Coding Style URL: https://docs.moodle.org/dev/Coding_style
(besucht am 10.12.2018)

[8] MySQL General Query Log URL: https://dev.mysql.com/doc/refman/8.0/en/query-log.html
(besucht am 10.12.2018)

[9] moodle plugins - Activities: StudentQuiz URL: https://moodle.org/plugins/mod_studentquiz
(besucht am 10.12.2018)

[10] Moodle Sourcecode auf github.com URL: https://github.com/moodle/moodle
(besucht am 10.12.2018)

[11] Moodle Mobile URL: https://github.com/moodlehq/moodlemobile2
(besucht am 10.12.2018)

[12] Ionic Framework URL: https://ionicframework.com/
(besucht am 11.12.2018)

[13] PhpStorm - Configuring Xdebug URL: https://www.jetbrains.com/help/phpstorm/
configuring-xdebug.html
(besucht am 11.12.2018)

[14] MySQLTutoral - MySQL INSERT Statement URL: http://www.mysqltutorial.org/mysql-insert-statement.aspx
(besucht am 12.12.2018)

[15] Travis CI - Continuous Integration Software URL: https://travis-ci.org/
(besucht am 13.12.2018)

[16] Moodle Privacy Api Development Guide URL: https://docs.moodle.org/dev/Privacy_API
(besucht am 18.12.2018)

[17] Typescript URL: https://www.typescriptlang.org/
(besucht am 18.12.2018)