HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Crowdsourced Quizzes

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2014

Autor(en): Oussama Zgheb, Tobias Zahner
Betreuer: Prof. Frank Koch

Quizzenger - Technischer Bericht

Oussama Zgheb & Tobias Zahner

19. Dezember 2014

Datum	Version	Änderung	Autor
23.09.14	1.0	Initial	Oussama Zgheb
23.09.14	1.1	Ergänzungen	Tobias Zahner
25.09.14	1.2	Managment Summary	Oussama Zgheb
01.10.14	1.3	Ergänzungen	Oussama Zgheb
04.11.14	1.4	Implementationsdetails	Tobias Zahner
17.12.14	1.5	Statistiken	Oussama Zgheb

Inhaltsverzeichnis

1 Abstract **5**

2 Management Summary **5**
2.1 Ausgangslage . . . 5
2.1.1 Zielgruppe . . . 5
2.1.2 Bestehende Lösungen . . . 5
2.2 Ergebnisse . . . 7
2.3 Statistiken . . . 7
2.3.1 Zeitauswertung . . . 7
2.3.2 Versionsverwaltung . . . 8
2.3.3 Lines of Code . . . 9
2.4 Ausblick . . . 10

3 Aufgabenstellung **11**
3.1 Motivation . . . 11
3.2 Funktionale Anforderungen . . . 11
3.2.1 UC - Quiz durchführen . . . 11
3.2.2 UC - Quiz erstellen . . . 11
3.2.3 UC - Lernen . . . 12
3.2.4 UC - Frage erstellen . . . 12
3.2.5 UC - Frage bewerten . . . 12
3.2.6 UC - Frage suchen . . . 12
3.2.7 UC - Moderation einer Kategorie (CRUD) . . . 12
3.3 Nicht funktionale Anforderungen . . . 12
3.3.1 Sicherheit . . . 12
3.3.2 Leistung . . . 13
3.3.3 Datenintegrität . . . 13
3.3.4 Wiederherstellung . . . 13
3.3.5 Wartbarkeit . . . 13
3.3.6 Benutzerfreundlichkeit . . . 13
3.4 Machbarkeitsnachweis . . . 13
3.4.1 Fragetypen . . . 14
3.4.2 Performance . . . 14
3.4.3 Test . . . 14

4 Lösungskonzept **16**
4.1 Deployment . . . 16
4.2 Architektur . . . 16
4.2.1 Logische Architektur . . . 16
4.3 Datenbank . . . 18
4.3.1 Schema . . . 19
4.4 Qualitätsmassnahmen . . . 21
4.4.1 Integrationstests . . . 21
4.4.2 Browsertests . . . 21
4.4.3 Usabilitytests . . . 21
4.4.4 Performance . . . 22

5 Umsetzung **23**
5.1 Quiz durchführen . . . 23

5.2 Bewertungen . . . 24
5.3 Schwierigkeit . . . 24
5.4 Suche . . . 24
5.5 Lernmodus . . . 25
5.6 Ajax . . . 25
5.7 Mobiletauglichkeit . . . 26
5.8 Sessionmangement . . . 26
5.8.1 Login . . . 26
5.8.2 Logout . . . 26
5.8.3 Überprüfung . . . 27
5.9 SQL Helper . . . 27
5.10 Frameworks & Libraries . . . 27
5.10.1 Fremd Code - Deklarierung . . . 28
5.10.2 DataTables . . . 28
5.10.3 Bootstrap . . . 28
5.10.4 Bootstrap Validator . . . 28
5.11 Logging . . . 29
5.11.1 Aufteilung . . . 29
5.11.2 Aufbau . . . 29
5.11.3 Loglevel . . . 29
5.12 Konfiguration . . . 29
5.13 Sicherheit . . . 31
5.13.1 Authentifizierung . . . 31
5.13.2 Sicherheitsmassnahmen . . . 31

6 Quellverzeichnis **33**

7 Anhänge **33**

1 Abstract

Fragesammlungen und daraus resultierende Test- und Quiz-Systeme stehen und fallen mit der Anzahl der vorhandenen Fragen. Da ein einzelner Administrator oder ein Moderatorenteam keinen allzu grossen Fragepool zusammenbringen können, ohne einen riesigen Aufwand zu betreiben, werden bei Quizzenger die Fragen aus der Crowd gewonnen. Dies hat neben der reinen Anzahl der Fragen auch den Vorteil, dass das Fachwissen von vielen verschiedenen Leuten genutzt wird und die Fragen damit qualitativ hochwertig und vielfältig werden. Als Vorbild dient dabei Wikipedia[1], mit dem Unterschied, dass bei Quizzenger Fragen statt Beiträge gesammelt werden.

Natürlich ist ein Crowdbasiertes System nicht ohne Tücken und es sind zwei Herausforderungen zu bewältigen:

1. Wie bringe ich die User dazu nicht nur Fragen und Quizzes zu lösen, sondern auch selber Fragen einzureichen?
2. Wie stelle ich die Qualität der eingereichten Inhalte sicher und wie vermeide ich Vandalismus?

Punkt 1 wurde mit der Implementation einer Community und einem dazugehörigen Punktesystem gelöst. Dieses soll Anreiz schaffen, selber Fragen einzureichen.

Das Punktesystem löst indirekt auch den zweiten Punkt: Durch die Punktzahl wird geregelt wer welche Rechte erhält. Kurz gesagt heisst dies, dass die vorbildlichen User mit Ansehen in der Community oder gar Moderationsrechten belohnt werden. Zusätzlich können Inhalte von den Benutzern bewertet werden, was einen Einfluss auf die Relevanz der Inhalte in der Anwendung hat. Quizzenger wurde als Webapplikation implementiert wobei Serverseitig PHP und MySQL und Clientseitig HTML 5, CSS 3, Javascript, AJAX und jQuery verwendet wurden. Um Quizzenger auch Mobiletauglich zu machen wurde zusätzlich Bootstrap [2] eingesetzt.

2 Management Summary

2.1 Ausgangslage

2.1.1 Zielgruppe

Unsere Installation von Quizzenger zielt in erster Linie auf den Einsatz an Schulen und Universitäten ab. Es ist sowohl für Dozenten als auch für die Studenten ein nützliches Hilfsmittel. Gerade wenn viele Studenten an Quizzenger teilnehmen, kommt schnell eine grosse Anzahl von Fragen zusammen. Diese können dann z.B. als Prüfungsvorbereitung genutzt werden. Die Dozenten hingegen, haben Zugriff auf einen grossen Fragepool und können sich von diesen zu Prüfungsfragen inspirieren lassen. Natürlich können sie zusätzlich auch Probetests erstellen oder selber Fragen einreichen um die Studenten zu kontrollieren. Bei allfälligen Probetests sehen sie aufgrund der Durchführungsstatistiken, welche Schüler noch Probleme haben. Da der Anzahl Kategorien keine Grenzen gesetzt sind, kann durchaus eine Installation für die gesamte Schule oder sogar für mehrere Schulen gemeinsam genutzt werden.

Jedoch ist Quizzenger nicht auf Schulen beschränkt. Da Quizzenger GPLv3 lizenziert ist, steht die Software für jeden kostenlos zur Verfügung und kann nach belieben angepasst und erweitert werden. So könnten auf einer eigenen Installation die Kategorien beliebig gewählt werden, z.B. innerhalb eines Berufsfeldes.

2.1.2 Bestehende Lösungen

Im Rahmen der Studienarbeit haben wir drei bestehende Lösungen angeschaut und verglichen. Es handelt sich dabei um die Produkte Wikiquiz, Innovedum und Academe.

Wikiquiz bietet zwar viele Features, es mangelt aber an qualitativ hochwertigen Inhalten. Dazu wirkt das Design überladen und veraltet, was nicht wirklich dazu einlädt, dort aktiv zu werden. Beim Test funktionierte ausserdem die Registrierung nicht, so dass eine Teilnahme verunmöglicht wurde. Wikiquiz hat also durchaus die richtige Idee, jedoch ist es sehr schlecht und viel zu umständlich umgesetzt worden.

Innovedum wurde ziemlich schön umgesetzt, jedoch ist es auf ein Thema beschränkt und die Themen sind vorgegeben. Es kann dadurch nicht wirklich für eigene Zwecke gebraucht werden. Auch ist es nur für Schweizer Studenten möglich daran teilzunehmen (Switch-Login). Dazu fehlt eine Suchfunktion um Inhalte schneller zu finden.

[1] http://www.wikipedia.org
[2] http://getbootstrap.com

Academe ist sehr umständlich und ebenfalls nur für Schweizer Studenten möglich. Es scheint nur möglich zu sein, eigene Fragensets anzulegen und diese dann durchzuführen. Ein gemeinsamer Fragepool war nicht auffindbar, ebenso wenig wie andere Inhalte.

Die drei getesteten Lösungen im Vergleich:

	Wikiquiz	**Question Bank Projekt - Innovedum**	**Academe**
Wer kann Autor werden?	Jeder, Quiz muss jedoch durch einen Administrator freigeschaltet werden.	Nur autorisierte Personen dürfen Fragen erstellen. Es wird ein Authentisierungspasswort benötigt.	Es gibt verschiedene Usergruppen, unter anderem die Gruppe Autoren. Allerdings bleibt unklar, wie man in die Benutzergruppen kommt.
Wer kann an einem Quiz teilnehmen?	Jeder, Registrierung benötigt.	Alle Schweizer Studenten	Alle Schweizer Studenten
Wie kann man sich einloggen?	Mit seinem angelegten Login.	Switch-AAI	Switch-AAI
Welche Fragetypen sind verfügbar?	- Single- & Multiple-Choice - Bilder ordnen (Drag & Drop) - Bilder anklicken - Antworten eintippen (Lückentext)	- Single- & Multiple Choice	- Keine Angaben / Keine Quizzes verfügar
Wie wird bewertet?	Pro Frage kann eine Bewertung mit Sternen und Text abgegeben werden, ohne dass man dazu aufgefordert wird. Man muss manuell auf „bewerten“ klicken.	Fehler und Anmerkungen können in einem Forum gemeldet werden.	Pro Quiz ist ein Feedbackformular vorhanden,dieses kann jederzeit ausgefüllt werden.
Wie ist die Navigation gestaltet?	Treeview mit vorgegebenen Hauptthemen. Auf Anfrage können neue Unterthemen erstellt werden.	Treeview der Alphabetisch geordnet ist. Fixe Tiefe, pro Thema gibt es sowohl Karteikarten als auch Testkarten.	Keine Struktur erkennbar.
Was für eine Suchfunktion ist vorhanden?	Suche nach: - Themenbereich - Autor - Textsuche	Keine	Suche nach Modul, Kurs oder Frage. Extended Search über verschiedene Attribute.

Gegenüber diesen drei Lösungen, bietet Quizzenger folgende Vorteile:

- Einfache Struktur, es braucht keine Einarbeitung
- Quiz können manuell oder automatisch erstellt werden
- Registrierung für jeden möglich
- Kategoriestruktur ist nicht vorgegeben und dennoch kontrollierbar
- Sourcecode ist öffentlich einsehbar (GPLv3-Lizenz)
- Software kann beliebig für eigene Zwecke angepasst werden
- Auch für mobile Geräte optimiert
- Motivierendes, leicht verspieltes Design führt zu einem höheren Spassfaktor
- Community-Features integriert

2.2 Ergebnisse

Das Ergebnis der Studienarbeit ist eine voll funktionale, Cross-Platform taugliche und getestete Webapplikation welche alle in der Aufgabenstellung gegeben Punkte erfüllt. Die anfänglich definierten Use Cases konnten in ihren Grundzügen übernommen und umgesetzt werden.

Die Performance sowie das Zusammenspiel der einzelnen Komponenten überzeugen in allen Aspekten.

Dank der frühen Entscheidung, das MVC Pattern zu verwenden und einen versierten Prototypen zu erstellen wurde ein einfach wartbarer Code erstellt.

2.3 Statistiken

2.3.1 Zeitauswertung

Die Studienarbeit wird mit 8 ECTS Credits verrechnet. Dies entspricht einem Aufwand von 240 Stunden pro Student. Insgesamt soll also 480 Stunden in das Projekt investiert werden.

Soll / Ist Stunden Wie in der untenstehenden Tabelle zu sehen ist, konnte der gegeben Zeitrahmen sehr gut eingehalten werden.

Soll	Ist
480 h	482.60 h

Studentotal pro Teammitglied Die Projektzeit ist mit einer minimalen Abweichung von 3% sehr fair verteilt worden.

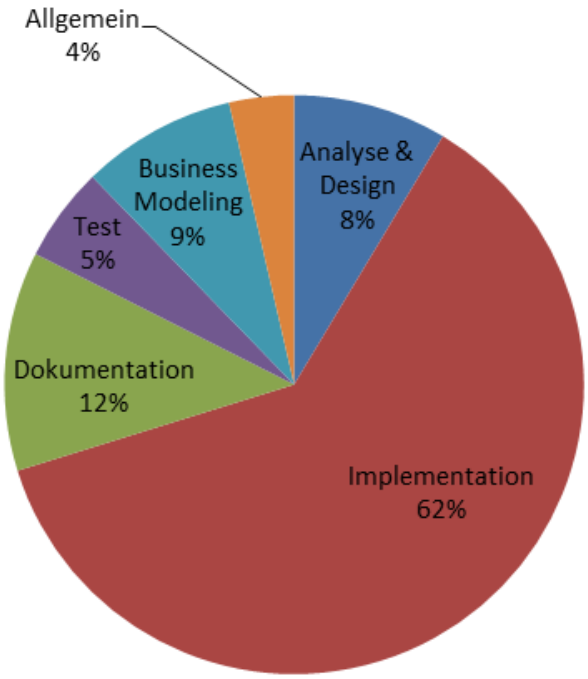
Teammitglied	Studentotal
Tobias Zahner	237.55 h
Oussama Zgheb	245,05 h

Aktivitäten Bei der Zeiterfassung in Redmine kann jede abgebuchte Zeit einer Aktivität zugeordnet werden.

Auf der untenstehenden Grafik sieht man, dass die Aufwände in den einzelnen Aktivitäten den RUP Prozess widerspiegeln, z.B. dass die Implementation in C1 beginnt (Oktober) und gegen Ende gegen null geht.

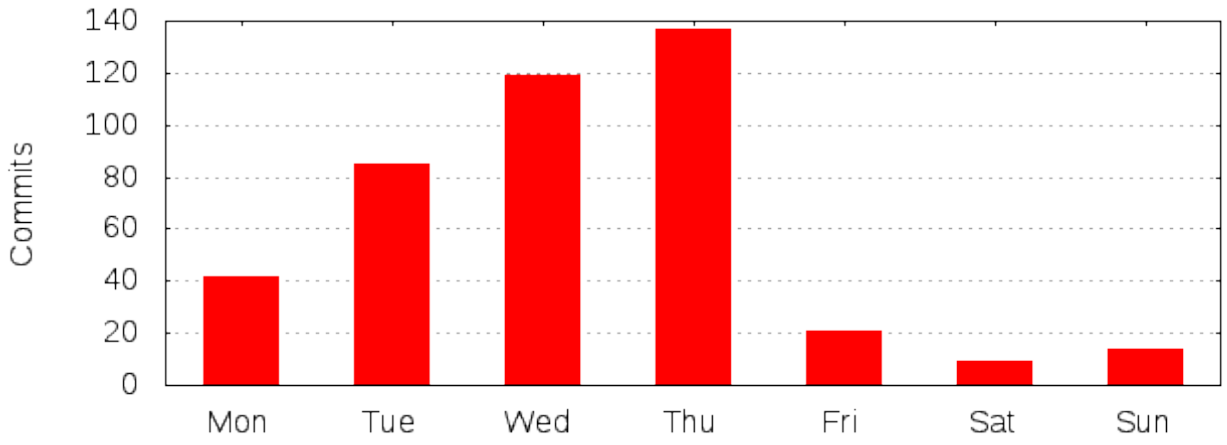
Aktivität	2014-9	2014-10	2014-11	2014-12	Gesamtzeit
Analyse & Design	19.20	13.50	4.25	4.00	40.95
Implementation	9.25	102.50	147.05	35.25	294.05
Dokumentation	3.25	9.75	1.50	44.00	58.50
Test			3.25	22.00	25.25
Requirements	1.60	1.75			3.35
Business Modeling	21.00	18.25	2.00		41.25
Deployment				2.00	2.00
Allgemein	2.50	12.00		2.75	17.25
Gesamtzeit	**56.80**	**157.75**	**158.05**	**110.00**	**482.60**

Hier sieht man noch die Aktivitäten im Verhältnis zueinander.

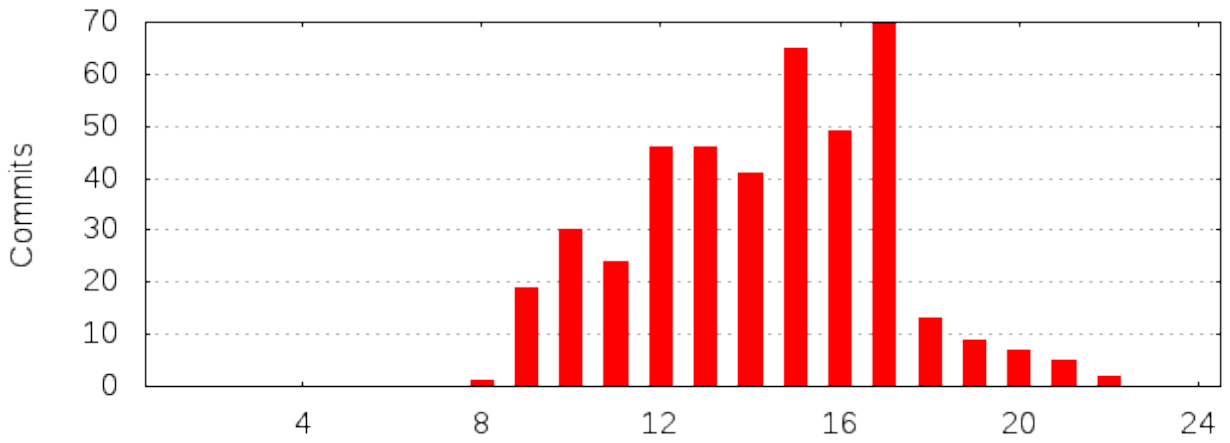


2.3.2 Versionsverwaltung

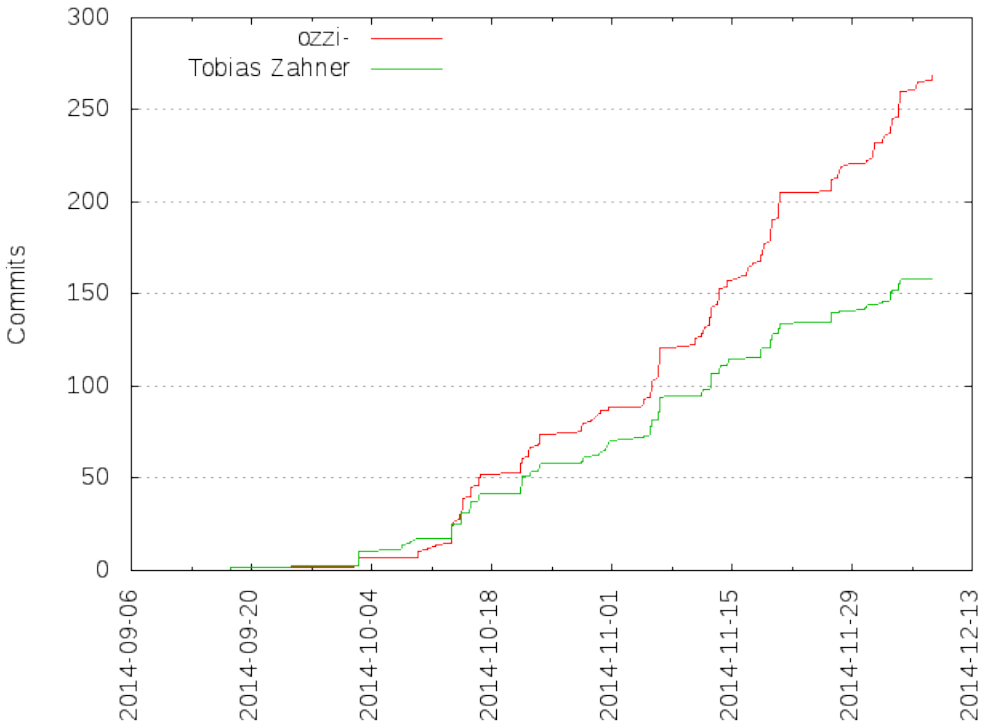
Commits - Tage Zu sehen ist, dass Mittwoch sowie Donnerstag die aktivsten Tage waren. Dies überschneidet sich auch mit den freigehaltenen Slots für die SA welche beide Teammitglieder festgelegt haben.



Commits - Tageszeit Hier zu sehen ist , dass die beliebteste Uhrzeit für Commits 15:30-16:30 ist. Dies deckt sich mit den Erwartung, nach einem Arbeitstag seine Änderungen zu commiten.



Commits - Total Wie hier zu sehen ist, wurde kontinuierlich gepusht, wobei dies das Teammitglied Oussama Zgheb (ozzi-) in kürzeren Intervallen getan hat.



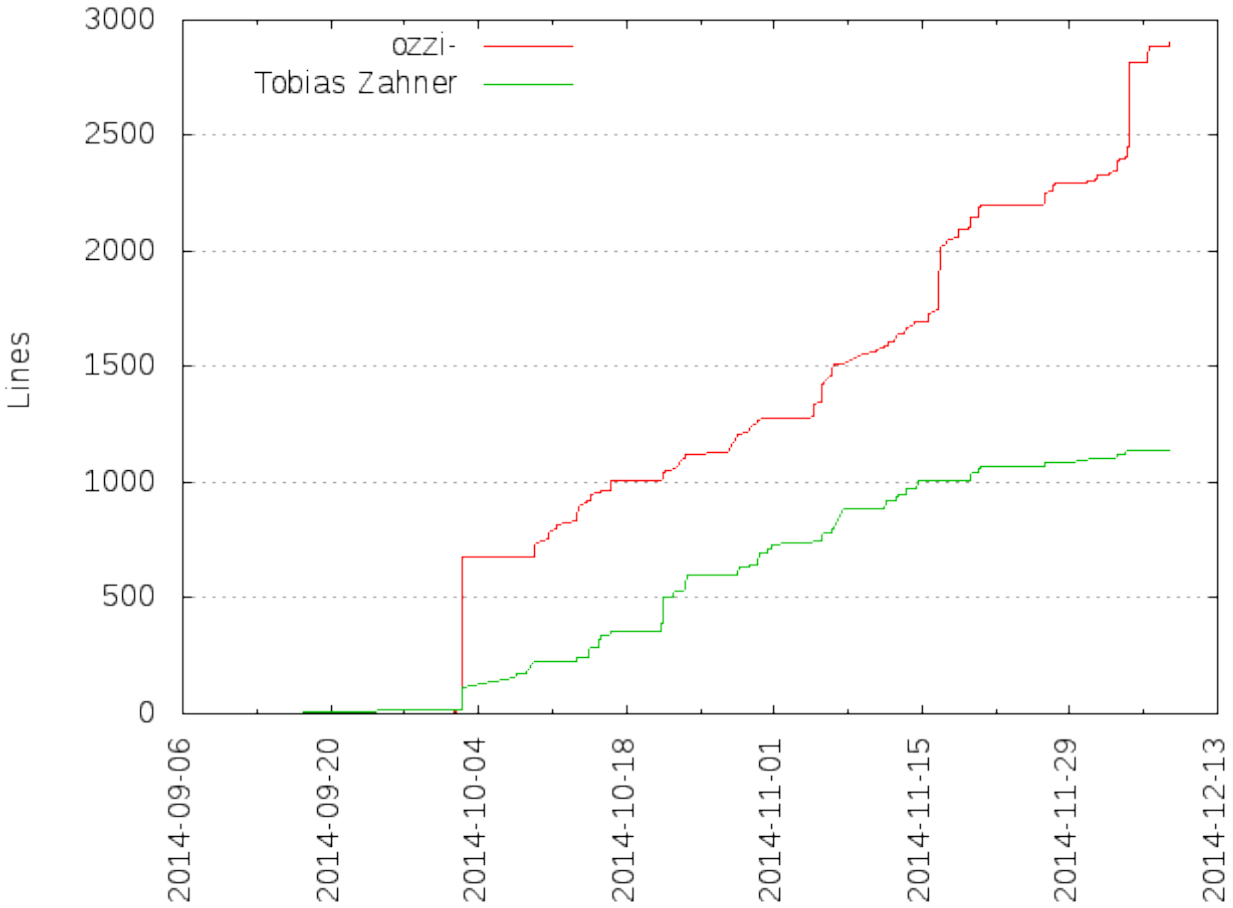
2.3.3 Lines of Code

»Lines of Code« sind eine von vielen brauchbaren Indikatoren einer Projektgrösse. Um eine Übersicht zu bieten wurde mit CLOC [3] eine detaillierte Auswertung erstellt.

Sprache	LOC
PHP	2313
HTML	1911
SQL	196
CSS	101

Hier zu sehen ist die aktuelle Zahl LOC im Projekt über die gesamte Projektzeit pro Teammitglied.

[3] http://cloc.sourceforge.net/



Der Unterschied der LOC kann durch die etlichen sichtbaren Sprünge erklärt werden. Diese sind externe Libraries / Tools die importiert wurden durch Oussama Zgheb. Wie z.B. am 04.10. wurde Bootstrap importiert, um den 15.11. Datatables. Der letzte Sprung entstand schliesslich durch ein grösseres Refactoring.

Wenn man die Imports weglässt, sieht man, dass ziemlich genau gleich viel Code eingereicht wurde.

Diese Diagramme und weitere Git Statistiken befinden sich in den Anhängen und wurden mit GitStatistics[4] erstellt.

2.4 Ausblick

In naher Zukunft sehen wir in den folgenden drei Bereichen das grösste Ausbaupotential :

Weitere Fragetypen Da während der ganzen Entwicklung von Quizzenger auf weitere Fragetypen geachtet wurde, sollte es gut möglich sein die Anwendung um weitere Fragetypen zu erweitern. Weitere Informationen dazu können dem Kapitel 3.4.1. entnommen werden.

Auswertung Da alle nutzbaren Daten der Benutzer gespeichert werden, können durchaus ausgeklügelte Bewertungsmechanismen implementiert werden. Als Beispiel ein Indikator für die »Sharpness« einer Frage, also ob Sie präzise formuliert ist. Auch die Schwierigkeit der Frage kann durch Einbezug der Leistungen der beantwortenden Usern genauer gewichtet werden.

Medientypen Es könnten verschiedene Medien einer Frage angegeben werden, also ein Bild welches die Fragestellung erweitert etc.

[4] `http://gitstats.sourceforge.net/`

3 Aufgabenstellung

Die durch den Betreuer Prof. Frank Koch gegebene Aufgabenstellung kann in den Anhängen gefunden werden.

3.1 Motivation

Unsere Motivation ist es als erste deutschsprachige Fragesammlungs- und Lernplatform folgende Punkte zu erfüllen:

- Frei zugänglich
- Quantität und Qualität der Inhalte sind akzeptabel
- Einfacher Aufbau
- Mobiletauglichkeit

Mit dieser Plattform wollen wir allen eine Möglichkeit bieten, von Ihrem Wissen untereinander zu profitieren.

3.2 Funktionale Anforderungen

3.2.1 UC - Quiz durchführen

1. Ein Teilnehmer erhält einen Link zu einem Quiz.
2. Der Teilnehmer öffnet den Link in seinem Browser auf einem beliebigen tauglichen Gerät (PC/Tablet/Smartphone etc.).
3. Der Teilnehmer startet das Quiz.
4. Eine Frage erscheint und der Teilnehmer wählt die Antwort aus, die er für richtig hält.
5. Die korrekte Lösung wird angezeigt sowie allfällige Erklärungen.
6. Schritte 4 & 5 werden so oft wiederholt, bis das Quiz beendet ist.
7. Der Teilnehmer sieht nun wie viele Punkte er erreicht hat und wie viele Punkte maximal möglich gewesen wären.

Extensions:

5. a) Der Teilnehmer ist mit der Frage / Lösung nicht einverstanden oder möchte einen Kommentar hinterlassen.

   1. Mittels der Diskussionsfunktion werden ihm alle bisherigen Kommentare und Bewertungen angezeigt.
   2. Der Teilnehmer kann nun selbst einen Kommentar und / oder eine Bewertung abgeben.

7.a) Der Teilnehmer will das Quiz wiederholen.

   1. Zu Schritt 3.

7.b) Der Teilnehmer will das Quiz bei sich speichern

   1. Falls der Teilnehmer angemeldet ist, hat er die Möglichkeit eine Kopie des durchgeführten Quizzes zu speichern.

3.2.2 UC - Quiz erstellen

1. Der Benutzer markiert eine oder mehrere Fragen.
2. Der Benutzer fügt die markierten Fragen einem bestehenden oder neuen Quiz hinzu.
3. Schritt 1. - 2. wiederholt sich beliebig.

3.2.3 UC - Lernen

1. Der Benutzer navigiert zur "Lernen"-Seite.
2. Er wählt die Kategorie(n), Fragenanzahl, Suchmodus und Schwierigkeitsgrad aus, die er gerne hätte.
3. Das System generiert ihm ein Quiz, das möglichst nahe an seine Kriterien herankommt.
4. Der User führt das Quiz durch.
5. Nach Beendigung des Quizzes hat der User die Wahl, ob er das Quiz erneut lösen, beenden, speichern oder teilen will.

3.2.4 UC - Frage erstellen

1. Der Benutzer navigiert zur "Frage erstellen"-Seite.
2. Er wählt Themenbereich, Kategorie und Subkategorie aus, in der er die Frage erstellen will.
3. Der Benutzer gibt die Frage und die Antwortmöglichkeiten ein. Die richtige Antwort markiert er mit einem Haken.
4. Der Benutzer speichert die Frage und erstellt bei Bedarf eine weitere Frage in der gleichen Kategorie.

Extensions:

2.a) Die gewünschte Subkategorie existiert noch nicht / Die Frage passt in keine bestehende Subkategorie

1. Der Benutzer erstellt die gewünschte Subkategorie.

3.2.5 UC - Frage bewerten

1. Der Benutzer startet ein Quiz und beantwortet die erste Frage.
2. Nach Beantwortung der Frage, hat der User die Möglichkeit die Frage zu bewerten. Zusätzlich kann er auch einen Kommentar zu seiner Bewertung abgeben.
3. Nach der (optionalen) Bewertung geht der User weiter zur nächsten Frage.

3.2.6 UC - Frage suchen

1. Der Benutzer wechselt zur Suchmaske und gibt einen Suchbegriff ein.
2. Es wird nach Fragen gesucht, die die gewünschten Keywords enthalten und diese werden aufgelistet. Durchsucht werden die Felder Fragetext und Tags.
3. Der Benutzer kann aus den Suchresultaten nun die Fragen des Suchresultates beantworten oder diese zu einem Quiz hinzufügen.

3.2.7 UC - Moderation einer Kategorie (CRUD)

1. Der Moderator sieht auf der Moderationsseite eine Liste aller Fragen seiner Kategorie(n), sortiert nach "Handlungsbedarf".
2. Auf jede Frage hat er Vollzugriff und kann diese bearbeiten, verschieben oder löschen.

3.3 Nicht funktionale Anforderungen

3.3.1 Sicherheit

Injections Injections sollen durch Prepared Statements nicht möglich sein.

Anmeldedaten in Datenbank Die Passwörter sollen mit einem zufälligen und eigenem Salt mit möglichst hoher Entropie als Hash gespeichert werden.

Passwortrichtlinien Die verwendeten Passwörter müssen mindestens 6 Zeichen lang sein und nicht gleich wie der Benutzername oder die Email Adresse.

CUD Operationen Sämtliche Create / Update / Delete Operationen sollen bei fehlenden Berechtigungen Clientseitig nicht sichtbar sowie Serverseitig überprüft werden. Gewisse Read Operationen auf Quizzes sind auch betroffen.

3.3.2 Leistung

Antwortzeiten Die Zeit die ein Benutzer nach einer Aktion warten muss bis die neue Seite vollständig auf seinem Gerät angezeigt ist, soll die branchenüblichen Antwortzeiten [5] einhalten.

Anzahl Benutzer Die Anzahl der gleichzeitigen Benutzer soll lediglich durch die Leistung der Hardware limitiert sein.

3.3.3 Datenintegrität

Die Datenbank soll durch Constraints immer in einem konsistenten Zustand sein.

Falls möglich sollen korrupte Daten eine Fehlermeldung auslösen und nicht weiter verarbeitet werden.

3.3.4 Wiederherstellung

Prozess Ein Einspielen des letzten SQL-Dump genügt um die Wiederherstellung von Quizzenger durchzuführen. Es wird empfohlen die Website in dieser Zeit vom Netz zu nehmen.

Automatisches Backup Ein automatisches Backup wird nicht durchgeführt. Wird dieses benötigt, kann der Administrator einen Cron Job einrichten, welcher einen SQL-Dump erstellt.

3.3.5 Wartbarkeit

Das System soll einfach Wartbar sein. Wichtige Eigenschaften müssen über ein Config-File definiert werden können.

3.3.6 Benutzerfreundlichkeit

- Sämtliche Formulare sollen validiert werden und dem Benutzer mitgeteilt werden was noch nicht stimmt.
- Das Look and Feel und die Bedienung soll auf allen Seiten einheitlich sein.
- Die Seite soll auf den meistgenutzten Geräten und Browsern korrekt dargestellt werden.

3.4 Machbarkeitsnachweis

Dieses Kapitel soll zeigen, dass die Ansprüche an weitere Funktionen und Performance in der Zukunft gegeben sind.

Die Aussagen beziehen sich auf die reine Datenbankebene, nicht auf die Softwareebene. Dies, da die Implementationsdetails je nach Umsetzung in der Zukunft variieren können und laut den Regeln

des Software Engineering kein Code produziert werden soll, bevor er gebraucht wird.

[5] http://www.nngroup.com/articles/website-response-times/

3.4.1 Fragetypen

Zusätzliche Fragetypen wurden als mögliche zukünftige Features gewünscht. Nachfolgend wird gezeigt, dass diese in unserer Architektur möglich sind oder wieso nicht.

Multiple Correct Answer Unsere Datenbank kann mittels dem Attribut »corectness« mehrere richtige Antworten festlegen.

Der Fragetyp muss dementsprechend gesetzt werden (Attribute Type).

BSP:

Frage	correctness
1	100
2	0
3	100

Complex Multiple Choice Die Complex Multiple Choice ist von den Anforderungen gleich wie die Multiple Correct Answer.

One best Answer Unsere Datenbank kann mittels dem Attribute »corectness« eine richtige sowie mehrere 'halbrichtige' Antworten festlegen.

Der Fragetyp muss dementsprechend gesetzt werden (Attribute Type).

BSP:

Frage	correctness
1	100
2	0
3	50

Fill-In (Lückentext) Es wird eine oder mehrere Answers gegeben welche die möglichen Texte festlegen.

Der Fragetyp muss dementsprechend gesetzt werden (Attribute Type).

True/False Es werden zwei Answers erstellt, eine »Richtig« eine »Falsch« und mittels dem Attribut »correctness« dementsprechend festgelegt.

Der Fragetyp muss dementsprechend gesetzt werden (Attribute type).

Zuordnung Dieser Aufgabentyp erfordert eine gänzlich andere Datenstruktur und kann nicht ohne zusätzliche Tabellen erstellt werden.

3.4.2 Performance

Diverse Vorkehrungen werden getroffen um die Performance auch bei schnell wachsender Userzahl zu unterstützen, diese sind unter dem Kapitel »Architektur« zu finden.

Um eine realistische Aussage über die Performance auch bei schnell wachsenden Datenmengen geben zu können wurden diverse Tests durchgeführt.

3.4.3 Test

Um die Performance zu testen wurden einige tausende Einträge wie folgt erstellt:

```
INSERT INTO questionperformance
(question_id, user_id, questionCorrect, quiz_id, session)
VALUES (3,5,1,qzID,i)
```

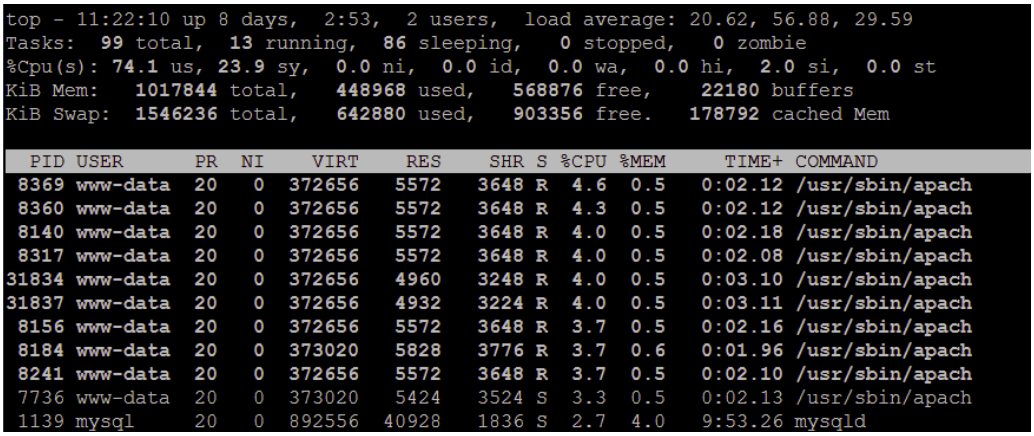
Danach wurde das Einfügen abgebrochen und gewartet bis der MySQL Dienst alles verarbeitet hat. Nun wurde der Load Generator gestartet und mehrere Performance Werte gemessen. Aus den Werten wurde der Mittelwert berechnet und festgehalten.

Load Generator Um möglichst realistische Bedingungen zu simulieren wurde ein Bashscript erstellt, welches gegen die 150 Abfragen pro Sekunde ausführt.

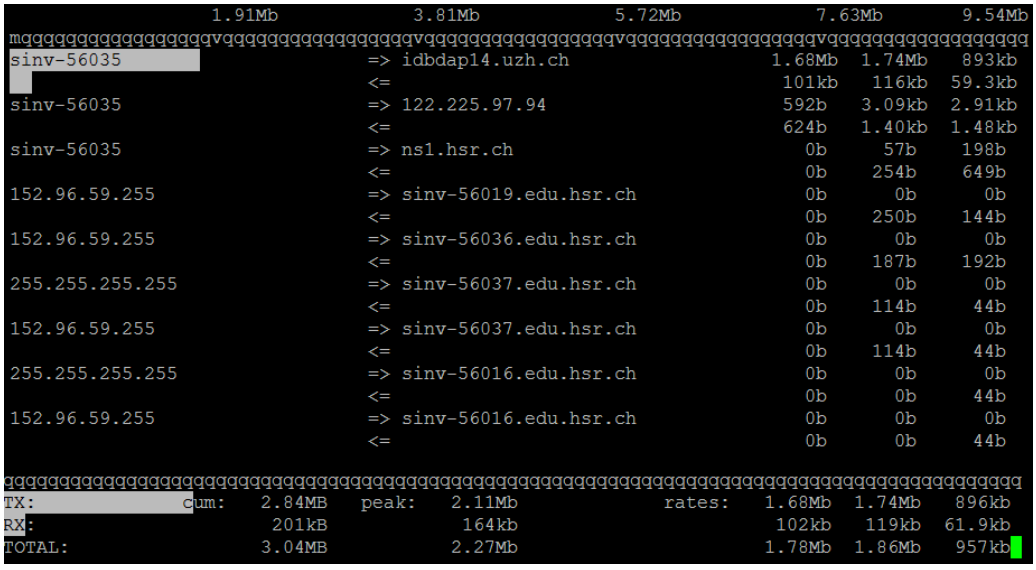
```
while :
do
wget -qO- https://sinv..../benchmark.php -no-check-certificate &> /dev/null
done
```

Auslastung des Servers Die Auslastung des Servers während dem Load Generator war ziemlich gross, bei etlichen Verbindungen kein Wunder. Die Reaktionszeit war trotzdem immer noch sehr gut und das System antwortete ohne Verzögerung in der Konsole.

Hier ein Screenshot des Servers, überwacht wurde mit dem »top« Befehl.



Hier ein Screenshot des Servers, das Tool »iftop« zeigt die aktuellen Verbindungen



Ergebnisse Die Ergebnisse wurden während der Ausführung des Load Generators gemessen, sobald sich nach dem Einfügen MySQL um die Index Generierung gekümmert hat.

Einträge	+	Speicher	+	Mittlere Antwortzeit	+	RAM	+
~ 520'000	-	24.5 MiB	-	5.8E-5	-	39.72 MB	-
~1'913'841	268%	87.6 MiB	257%	7.2E-5	24%	50.2 MB	25.8%
~6'452'523	237%	316.8 MiB	261%	7.5E-5	4.16%	55.6 MB	10.7%
~12'739'167	97.4%	606.1 MiB	91.3%	8.1E-5	8%	64.8 MB	16.5%

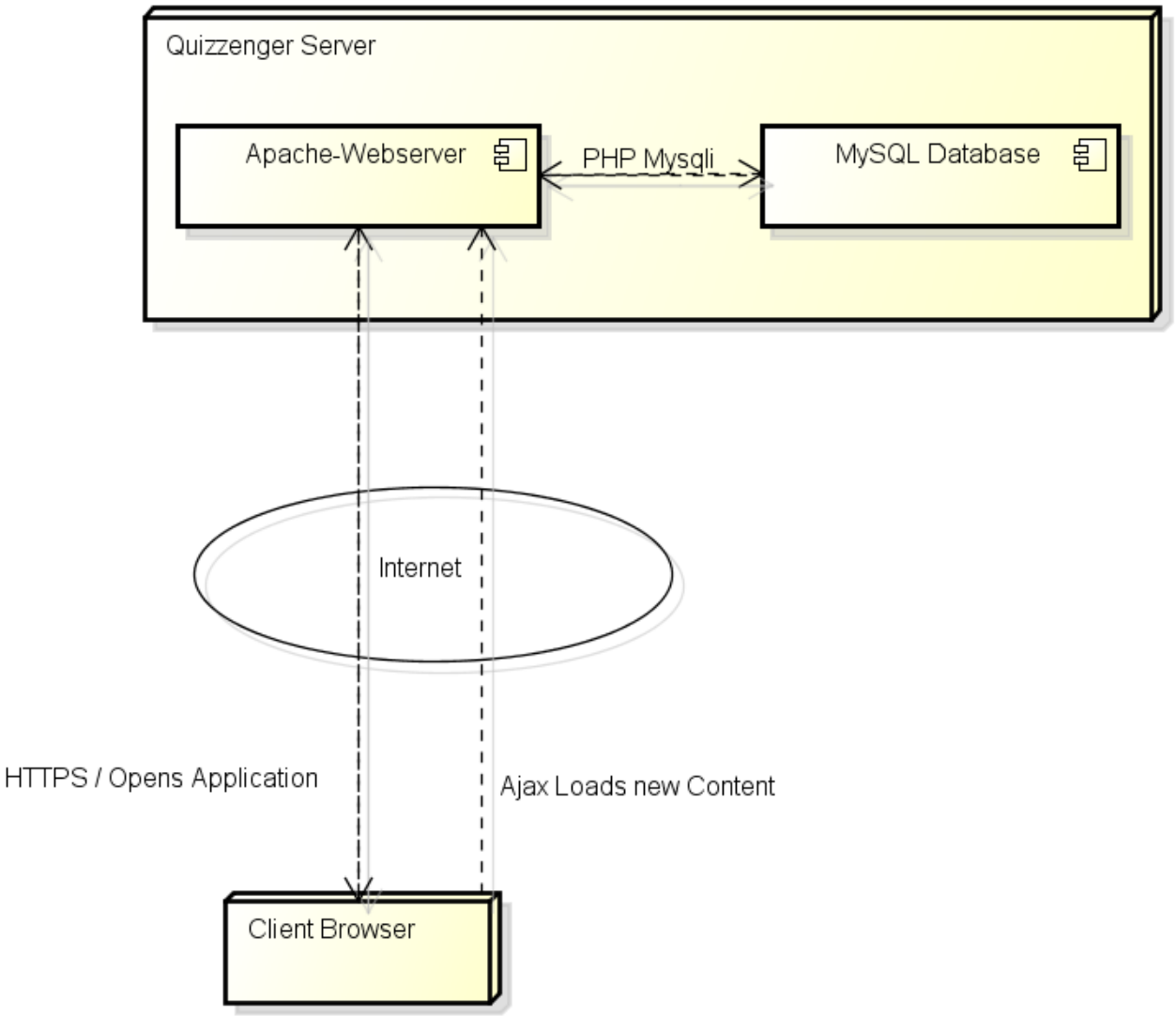
Schlussfolgerungen

- Die Grösse der Tabelle wächst ziemlich proportional zu der Anzahl Einträge
- Die Antwortzeit wächst nur gering und ist vollkommen unbedenklich
- Die Auslastung des Arbeitsspeichers wächst geringfügig, dies Problem lässt sich einfach mit stärkerer Hardware lösen

4 Lösungskonzept

4.1 Deployment

Quizzenger ist eine typische Webapplikation. Die Software läuft auf einem Webserver und verwendet die MySQL Datenbank. Die Datenbank kann sich dabei auch auf einem anderen Server befinden. Der Benutzer von Quizzenger kommuniziert via HTTPS mit dem Webserver und lädt bei Bedarf per Ajax Inhalte nach oder führt so Aktionen durch.



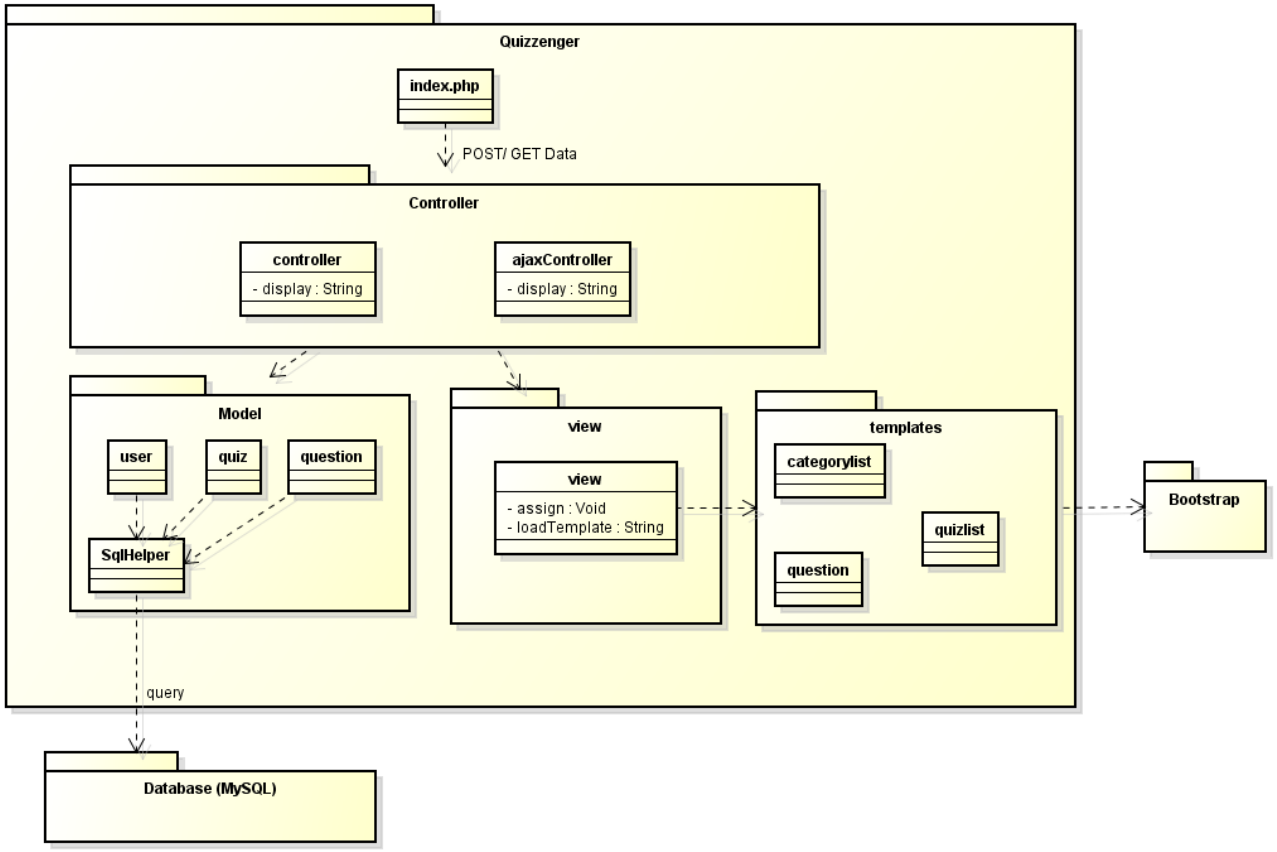
4.2 Architektur

4.2.1 Logische Architektur

Quizzenger setzt sich grob aus folgenden Komponenten zusammen:

- Index
- Controllers
- Models
- View
- Templates
- SQL-Helper
- Bootstrap
- DataTables

Anschliessend werden alle Komponenten im Detail betrachtet. Die untenstehende Grafik zeigt die Systemübersicht in UML.



Index ist der Einstiegspunkt in die gesamte Applikation. Dieser startet je nach Bedarf einen Controller oder einen Ajax Controller und übergibt diesem einen Array von POST und GET Parametern.

Controllers haben die Aufgabe, die passenden Templates der View zuzuordnen und diese dann mit Domain Objekten aus der Datenbank zu befüllen. Der Controller hat Zugriff auf alle geladenen Domain Models, welche die Schnittstelle zur Datenbank bilden. Die Controller wurden bewusst getrennt um die Leserlichkeit und Organisation des Codes zu verbessern. Bei Seiten die nur erreichbar sind, wenn der Benutzer angemeldet ist, führt der Controller eine Weiterleitung zur Loginseite durch.

Der Controller erstellt anhand den empfangenen Requests das gewünschte Template, lädt das Skelet der Seite und füllt den Inhalt ein.

Der ajaxController dient zur logischen Trennung der Ajax Aufrufe und den normalen kompletten Seiten. Der ajaxController lädt dabei nicht noch den ganzen HTML Header dazu und ist so schlank wie möglich gehalten um schnelle Antwortzeiten zu ermöglichen.

Models Domain Models enthalten diverse Funktionen welche spezifische Objekte oder Sets von Objekten aus der Datenbank zurückliefern oder verändern. Die meisten Models enthalten also Funktionen wie getObjectByID, getAllObjectsFromCategory und so weiter. Ausnahmen sind loginModel sowie registrationModel, welche mehr als nur Datenbank Querys enthalten.

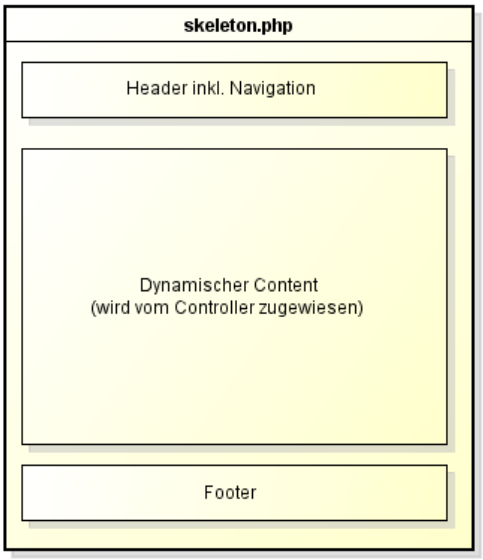
Die Models dürfen den User auf eine neue Seite führen, falls dies angebracht ist.

SQL Helper Da sämtliche Models viele verschiedene Datenbank Queries tätigen, wurde diese Funktionalität in eine Helper Klasse ausgelagert. In dieser wird ein Datenbankobjekt initialisiert und es können Queries darauf durchgeführt werden. Der SQL Helper bietet dazu auch die Funktionalität zur Aufbereitung der gewonnenen Daten für den Gebrauch in PHP-Datenstrukturen. Dadurch konnten die Model-Methoden stark vereinfacht und vereinheitlicht werden und duplizierter Code vermindert werden.

View Die View wurde erstellt um komfortabel ein Template laden zu können, sowie als Behälter für die anzuzeigenden Domain Models zu dienen. Beim laden eines Templates wird der generierte Output in einen Buffer gespeichert und nicht direkt ausgegeben, so kann der Controller bestimmen, wann die Seite angezeigt werden soll.

Templates Die Templates enthalten PHP Code der grösstenteils über die empfangenen Domain Objekte iteriert und HTML Code generiert, welcher diese Objekte nun visualisiert. Wichtig ist, dass keine »Programmlogik« in den Templates vorhanden ist, sodass diese bei Bedarf wiederverwendet oder ausgetauscht werden können. Da abgesehen vom eigentlichen Inhalt der Seite jede Seite gleich aufgebaut ist, wird ein Haupttemplate (skeleton.php) verwendet, welches die Inhalte enthält, die auf jeder Seite angezeigt werden. Das sind zum Beispiel die Navigation und der Footer sowie das gesamte HTML-Skelett (<html /><head /><body />). Dieses Haupt-Template enthält einen Container »content«, worin jeweils der Inhalt der Seite geladen wird. Dieses Modell hat mehrere Vorteile:

- Jede Seite sieht gleich aus, sauberer Aufbau
- Die Templates der einzelnen Seiten sind übersichtlicher, da nur der Content beschrieben wird
- Änderungen am Aufbau / Design der Seite müssen nur in einem File vorgenommen werden.



Je nach dem ob ein Benutzer angemeldet ist, werden gewisse Inhalte zusätzlich angezeigt. Diese sind zum Beispiel private Daten des Benutzers oder Funktionen welche nur angemeldeten Benutzern zur Verfügung stehen.

4.3 Datenbank

Die Persistenz wird mittels einer MySQL Datenbank realisiert und durch MySQLi von PHP aus angesprochen.

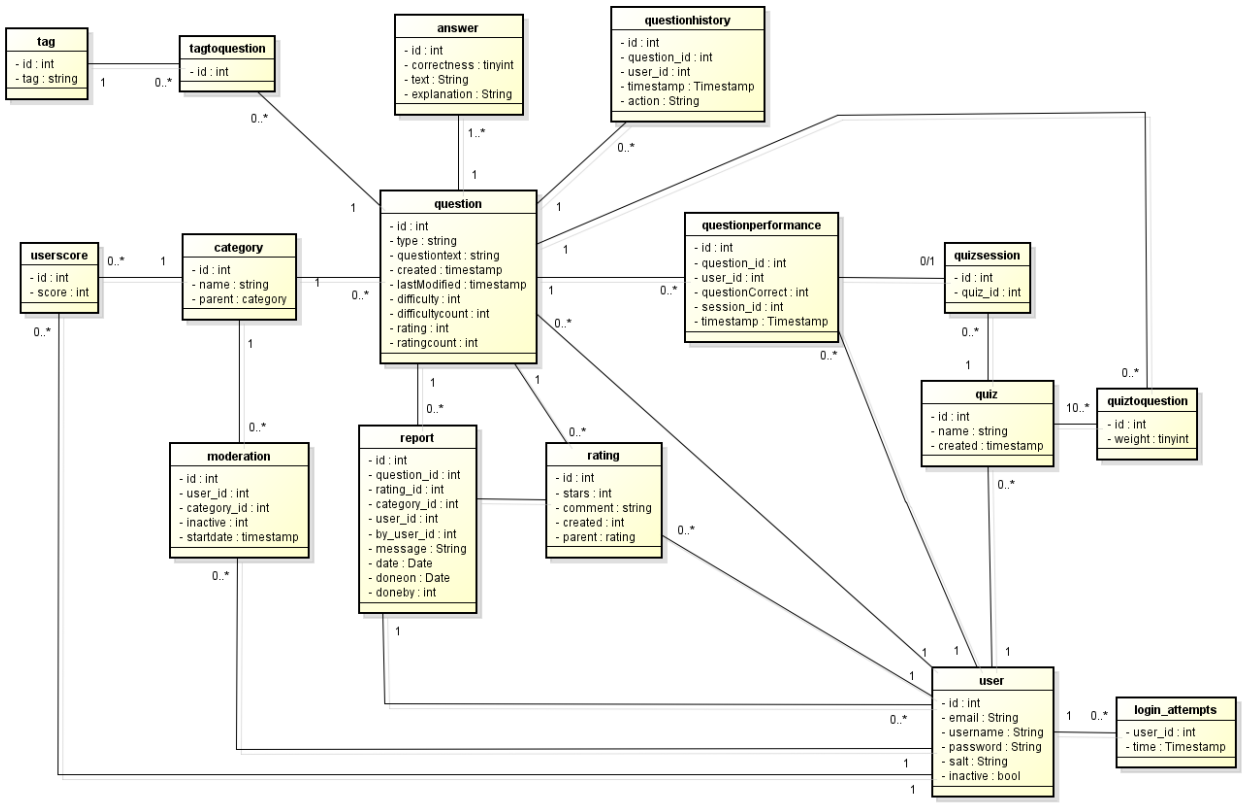
Wir haben uns für MySQL entschieden, da beide Teammitglieder bereits einiges an Erfahrung mit MySQL haben und PHP die beste Unterstützung dafür hat.

MySQLi ist eine PHP Erweiterung zum komfortablen Zugriff auf MySQL Datenbanken, wobei das i für improved steht. [6]

Die Klasse SqlHelper übernimmt die direkten Querys und bietet diverse Hilfsmethoden.

In den Querys wird bei Problemen ein Logeintrag hinzugefügt, sowie mittels Reflection der Aufrufer ermittelt.

4.3.1 Schema



Datenbank Schema in UML

Zu jeder Tabelle gibt es eine Erklärung ihrer Funktion sowie eine der einzelnen Feldern wo dies nicht selbsterklärend ist.

answer enthält mehrere Antworten für eine »question«. Wie viele Antworten es für eine Question gibt ist dabei variabel und wird durch die Anwendung und nicht durch die Datenbank überprüft. Dies ist so gelöst, da solche Constraints in der Datenbank selbst nur sehr mühsam zu ändern sind und nicht auch noch zusätzlich in der GUI geändert werden muss.

Correctness [tinyint] : Eine Zahl zwischen 0 - 100 welche die »Richtigkeit« der Antwort beschreibt, wobei 100 vollkommen richtig und 0 falsch bedeutet.

Im Rahmen der SA wurden keine Werte zwischendurch verwendet, diese finden lediglich bei anderen Fragetypen Verwendung.

category enthält die einzelnen Kategorien in einer Baumstruktur, dies wird durch speichern des Parentnode gespeichert. Falls eine Kategorie keinen Parentnode eingetragen hat (0) ist dies eine Oberkategorie. Die Leafnodes werden dadurch erkannt, da keine anderen Nodes diese als Parentnode haben.

login_attempts enthält Informationen zu fehlgeschlagenen Logininformationen. Dadurch können Bruteforce Attacken vermindert werden.

[6] `http://php.net/manual/de/book.mysqli.php`

moderation enthält Informationen welche Benutzer Moderationsrechte haben. Die Moderationsrechte werden Kategorienweise gegeben.

inactive [tinyint] : Entweder 0 oder 1, 1 bedeutet die Moderationsrechte wurden entzogen und sind demzufolge auch nicht in der Anwendung aktiv.

question enthält die vom Benutzer erstellten Fragen. Um die Abfragen auf die Datenbank zu minimieren werden die Bewertungen sowie die ermittelten Schwierigkeitsgrade in der Question selbst gespeichert, dies in einer aggregierten Form.

type [string] : Beschreibt den Fragetyp, im Rahmen unserer SA wurde der Fragetyp Singlechoice implementiert.

difficulty [double] : Eine Zahl zwischen 0-100 welche die Schwierigkeit einer Frage beschreibt, wobei 100 einfach und 0 schwer bedeutet.

difficultycount [int] : Enthält die Anzahl der in »difficulty« aggregierten einzelnen Werten.

rating [double]: Eine Zahl zwischen 0-5 welche die Bewertung einer Frage beschreibt, wobei 0 schlecht und 5 sehr gut bedeutet.

ratingcount [int] : Enthält die Anzahl der in »rating« aggregierten einzelnen Werten.

questionhistory enthält Informationen zur Änderungsgeschichte von Questions. Dies, da Fragen durch den Autor sowie Moderatoren editiert werden können. Somit ist eine gewisse Transparenz für alle Benutzer gegeben.

questionperformance enthält Informationen zur Beantwortung einer Frage durch einen Benutzer. Es wird dabei der Wert der »correctness« der gewählten »answer« gespeichert. Anhand dieser Daten werden Statistiken und Bewertungen generiert.

session_id [int] : Die Frage kann im Kontext eines Quizzes beantwortet werden, entweder NULL oder die ID einer »quizsession«.

quiz enthält einen Namen sowie den Benutzer. Die Fragen welche auf dieses Quiz zeigen können von verschiedenen Benutzern und Kategorien stammen. Das Quiz mit der ID = -1 wird benötigt, um die im »Lernen« Modus temporär Quizzes zu markieren.

quizsession wird benötigt um mehrere und sogar parallele Durchführungen von Quizzes zu ermöglichen.

quiztoquestion wird benötigt um einem Quiz mehrere Fragen hinzuzufügen.

weight [tinyint] : Ermöglicht das Gewichten einer Frage im Quizkontext, so kann eine besonders schwere Frage z.B. 5 Punkte geben während alle anderen lediglich einen Punkt geben.

rating ermöglicht die Bewertung einer Frage sowie deren Diskussion.

parent [int] : Falls 0 bezieht es sich direkt auf die Frage und ist somit eine Bewertung, falls ungleich 0 ist es ein Kommentar zu einer Bewertung.

report enthält Informationen zu gemeldeten Inhalten durch Benutzer.

Dabei kann ein Report durch einen Benutzer als erledigt markiert werden und wird danach nicht mehr angezeigt.

tag hilft zur feineren Kategorisierung einzelner Fragen.

Es hätten auch die Tags als String in der Frage selbst gespeichert werden können, dies würde einfacher sein aber die Normalform verletzen sowie die Suche eventuell verlangsamen.

tagtoquestion enthält Informationen zur Verlinkung von einem oder mehreren Tags für eine Frage.

user enthält die Benutzerinformationen.

Weitere Informationen unter dem Thema »Sicherheit« zu finden.

inactive[tinyint] - Entweder 0 oder 1, falls 1 kann sich dieser Benutzer nicht anmelden und enthält eine entsprechende Meldung.

superuser[tinyint] - Entweder 0 oder 1, falls 1 hat der Benutzer Zugriff auf diverse mächtige Verwaltungs- und Moderierungs-tools und wird dementsprechend in der Anwendung durch ein Bild gekennzeichnet.

userscore enthält die gesammelten Punkten eines Benutzers pro Kategorie.

4.4 Qualitätsmassnahmen

Um die Qualität zu gewährleisten wurden die folgenden Massnahmen getroffen.

4.4.1 Integrationstests

Nach jedem neu implementierten Feature wurde vor dem Push ins Repository, das neue Feature getestet und allfällig andere betroffene Module überprüft.

4.4.2 Browsertests

Mit Desktop sowie Mobile Browsern wurden eine spezifische Auswahl an Seiten von Quizzenger analysiert und auf Ihre Funktion getestet. Somit ist gewährleistet, dass die meist verbreiteten Plattformen unterstützt werden. Auf den Desktop Browsern wurde die Seite zudem in drei verschiedenen Grössen geöffnet, um das Responsive Layout zu beobachten.

Getestet wurden:

- Windows - Firefox - 33.0.2
- Windows - Internet Explorer - 11.0
- Windows - Opera - 26
- Android - Chrome
- Sailfish - Firefox

Auf Grund dieser Erkenntnisse konnten nachträglich einige kleine Browser-spezifische Unschönheiten entfernt werden.

Die Screenshots der Tests sind in den Anhängen zu finden.

4.4.3 Usabilitytests

Usabilitytests wurden aus Zeitgründen informell durchgeführt. Die Aufgabenstellung umfasste folgende Punkte:

- Registrieren und Einloggen
- Erstellen einer Frage
- Erstellen eines Quizzes (durch Verwendung der Suchfunktion)
- Durchführen eines Quizzes via URL

Anhand dieser Tests wurden folgende Aktionen getroffen:

- Umbenennung von Kategorien
- Meine Fragen und Meine Quizzes wurden zusammengefasst unter Meine Inhalte
- Suchfunktion zusätzlich in der Navbar

4.4.4 Performance

Datenbank Um die Datenbank performant zu gestalten wurden folgende Punkte erfüllt.

- Wo sinnvoll entweder in der dritten Normalform (grosse Datenmengen sowie Operationen) oder Daten aggregiert & doppelt gehalten (Zeitkritische abfragen)

- Verwendung von passenden Datentypen wie z.B. tinyint

- Verwendung der innoDB Engine für MySQL

Datei Crunch Da die verwendeten Java Scripts relativ gross waren, wurden wo vorhanden die »min« Versionen verwendet. Bei Scripts bei denen »min« Versionen zur Verfügung standen wurde ein Online YUI Compressor [7] verwendet .

Aggregation Um den schnellen Seitenaufbau auf Seitens des Benutzers zu unterstützen wurde die Anzahl der zu übertragenden Dateien möglichst klein gehalten.

Zudem ist es sinnvoll mehreren JS und CSS Dateien zusammenzuführen um so den Overhead zu verkleinern. [8]

Hier ein Beispiel der zu ladenden Dateien,

Vorher:

200	GET	index.php	sinv-56035.edu.hsr.ch	html	27,18 KB	→ 11 ms
200	GET	bootstrap.min.css	sinv-56035.edu.hsr.ch	css	122,69 KB	→ 15 ms
200	GET	bootstrap-theme.min.css	sinv-56035.edu.hsr.ch	css	18,41 KB	→ 2 ms
200	GET	custom.css	sinv-56035.edu.hsr.ch	css	2,70 KB	→ 7 ms
200	GET	jquery.dataTables.min.css	sinv-56035.edu.hsr.ch	css	14,77 KB	→ 6 ms
200	GET	dataTables.responsive.css	sinv-56035.edu.hsr.ch	css	2,36 KB	→ 9 ms
200	GET	jquery-1.11.1.min.js	sinv-56035.edu.hsr.ch	js	93,54 KB	→ 29 ms
200	GET	bootstrap.min.js	sinv-56035.edu.hsr.ch	js	147,84 KB	→ 32 ms
200	GET	custom.js	sinv-56035.edu.hsr.ch	js	1,93 KB	→ 10 ms
200	GET	bootstrapValidator.min.js	sinv-56035.edu.hsr.ch	js	108,33 KB	→ 28 ms
200	GET	de_DE.js	sinv-56035.edu.hsr.ch	js	12,04 KB	→ 13 ms
200	GET	ajax.js	sinv-56035.edu.hsr.ch	js	5,71 KB	→ 11 ms
200	GET	jqueryfunctions.js	sinv-56035.edu.hsr.ch	js	10,61 KB	→ 11 ms
200	GET	bootbox.min.js	sinv-56035.edu.hsr.ch	js	8,43 KB	→ 20 ms
200	GET	jquery.dataTables.min.js	sinv-56035.edu.hsr.ch	js	76,64 KB	→ 27 ms
200	GET	dataTables.responsive.min.js	sinv-56035.edu.hsr.ch	js	6,41 KB	→ 18 ms
200	GET	bg.png	sinv-56035.edu.hsr.ch	png	37,75 KB	→ 11 ms
200	GET	header_50.png	sinv-56035.edu.hsr.ch	png	5,14 KB	→ 3 ms
200	GET	superuser.png	sinv-56035.edu.hsr.ch	png	0,83 KB	→ 6 ms
200	GET	moderator.png	sinv-56035.edu.hsr.ch	png	0,84 KB	→ 3 ms
200	GET	sort_both.png	sinv-56035.edu.hsr.ch	png	1,48 KB	→ 3 ms
200	GET	sort_desc.png	sinv-56035.edu.hsr.ch	png	1,46 KB	→ 5 ms

Nacher:

200	GET	index.php?view=default&info=mes_login_success	sinv-56035.edu.hsr.ch	html	26,40 KB	→ 28 ms
200	GET	bootstrap.min.css	sinv-56035.edu.hsr.ch	css	122,69 KB	→ 16 ms
200	GET	bootstrap-theme.min.css	sinv-56035.edu.hsr.ch	css	18,41 KB	→ 9 ms
200	GET	custom.css	sinv-56035.edu.hsr.ch	css	2,70 KB	→ 10 ms
200	GET	jquery.dataTables.min.css	sinv-56035.edu.hsr.ch	css	14,77 KB	→ 9 ms
200	GET	dataTables.responsive.css	sinv-56035.edu.hsr.ch	css	2,36 KB	→ 9 ms
200	GET	ajax.js	sinv-56035.edu.hsr.ch	js	5,71 KB	→ 7 ms
200	GET	jquery-1.11.1.min.js	sinv-56035.edu.hsr.ch	js	93,54 KB	→ 54 ms
200	GET	bootstrap.min.js	sinv-56035.edu.hsr.ch	js	154,64 KB	→ 64 ms
200	GET	custom.js	sinv-56035.edu.hsr.ch	js	1,93 KB	→ 15 ms
200	GET	jqueryfunctions.js	sinv-56035.edu.hsr.ch	js	10,70 KB	→ 16 ms
200	GET	jquery.dataTables.min.js	sinv-56035.edu.hsr.ch	js	76,64 KB	→ 47 ms
200	GET	dataTables.responsive.min.js	sinv-56035.edu.hsr.ch	js	6,41 KB	→ 30 ms
200	GET	bg.png	sinv-56035.edu.hsr.ch	png	37,75 KB	→ 17 ms
200	GET	header_50.png	sinv-56035.edu.hsr.ch	png	5,14 KB	→ 12 ms
200	GET	superuser.png	sinv-56035.edu.hsr.ch	png	0,83 KB	→ 13 ms
200	GET	moderator.png	sinv-56035.edu.hsr.ch	png	0,84 KB	→ 50 ms
200	GET	sort_both.png	sinv-56035.edu.hsr.ch	png	1,48 KB	→ 8 ms
200	GET	sort_desc.png	sinv-56035.edu.hsr.ch	png	1,46 KB	→ 9 ms

[7] http://refresh-sf.com/yui/
[8] http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html

Server Auf Anpassungen an der Webserver Software (Apache 2) sowie versierten MySQL Einstellungen wurde im Rahmen der Studienarbeit aus Zeit und Kapazitätsgründen verzichtet. Dies da riesige Datenmengen sowie viele aktive Benutzer & langfristig Beobachtung nötigt sind für ein Feintuning.

Grundsätzlich können bei den Einstellungen Best Practices der Software verwendet werden.

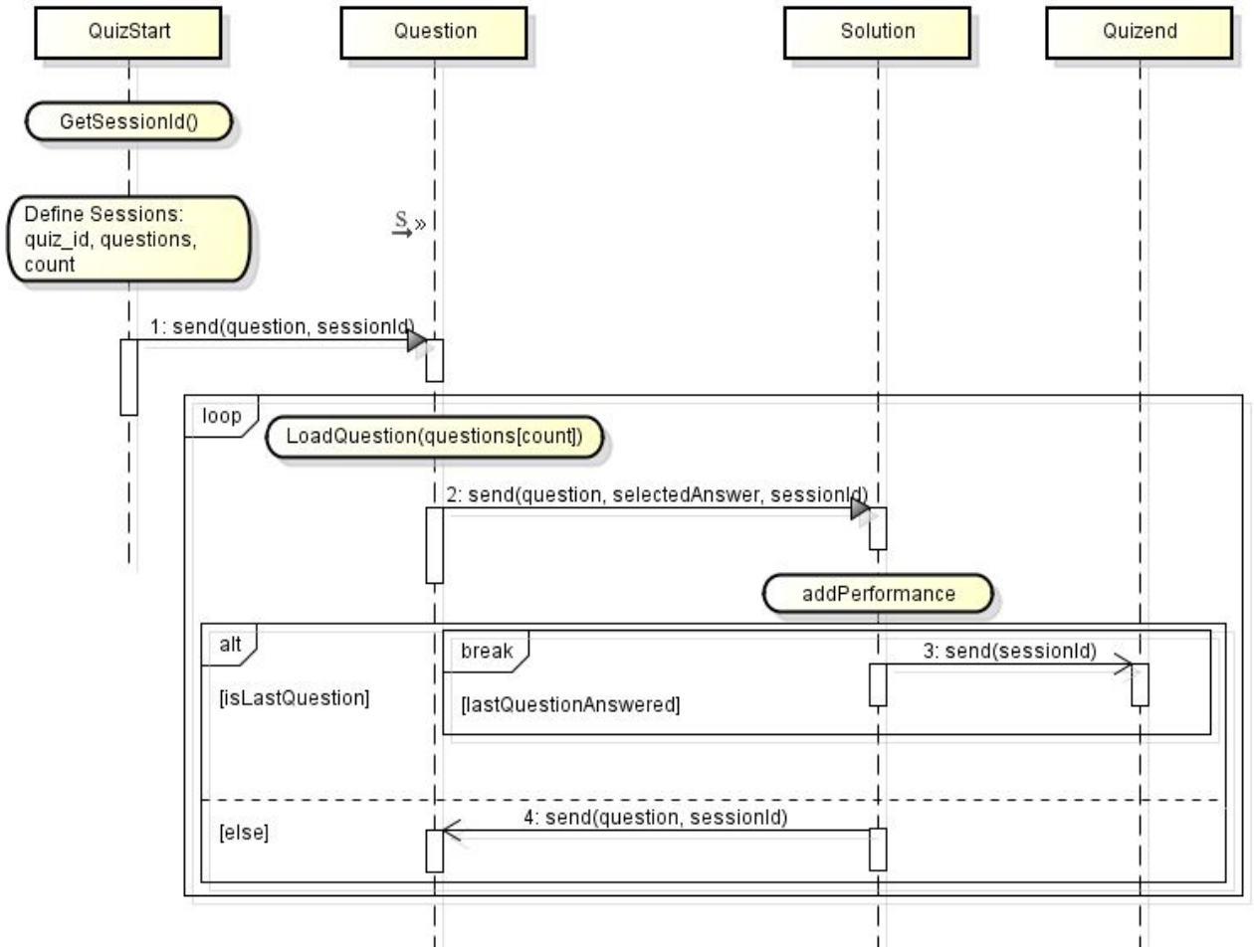
5 Umsetzung

5.1 Quiz durchführen

Die Quizdurchführung und Fragenbeantwortung sind keine triviale Angelegenheiten, dabei müssen viele Dinge beachtet werden. Dazu zählen folgende Herausforderungen:

- Es muss möglich sein, parallel mehrere Quizzes zu lösen (in verschiedenen Browser-Fenstern)
- Es soll dem User nicht möglich sein, während einem Quiz die Struktur sowie seine bereits getätigten Aktionen nachträglich zu beeinflussen.
- Das Aktualisieren einer Seite darf kein Fehlverhalten verursachen.

Implementiert wurde die Quizdurchführung wie in folgendem Sequenzdiagramm dargestellt:



Auf der Seite Quizstart wird eine neue SessionId generiert, diese SessionId definiert die aktuelle Quizdurchführung. Anschliessend werden die Session-Variabeln quiz_id (Quizidentifikation), questions (Fragen die zum Quiz gehören) und count (Anzahl beantwortete Fragen) definiert.

Mit einem Klick auf den Start-Quiz-Button wird der QuestionView die erste QuestionId sowie die SessionId mitgegeben bevor diese geladen wird. Dies geschieht über GET-Parameter. In der QuestionView wird die Frage nun geladen und angezeigt. Sobald der User diese beantwortet, wird er auf die solutionView umgeleitet. Dabei werden wieder via GET-Parameter die Argumente question, selected Answer und sessionId mitgegeben. Die SolutionView trägt anschliessend die Performance des Users für diese Frage in die Datenbank ein (siehe Datenbankmodell). Mit einem Klick auf nächste Frage wird wieder auf die QuestionView verlinkt unter Angabe von question und sessionId. Dieser

Vorgang wiederholt sich solange, bis die letzte Frage beantwortet wurde. Ist dies der Fall, wird auf die QuizEnd-View verwiesen, wo dem User anhand der eingetragenen Performances der Score dieser Durchführung angezeigt wird.

Etwas Kopfzerbrechen bereiteten die oben genannten Herausforderungen. Diese konnten aber allesamt bewältigt werden.

Die parallele Durchführung war zu Beginn nicht möglich, da die Session-Variabeln auch Browserfenster-, bzw. Tab-übergreifend definiert werden. Die Sessionvariabeln haben sich also gegenseitig überschrieben, was zu einem Durcheinander führte. Der erste Lösungsansatz war, die Sessionvariabeln durch POST-Daten zu ersetzen und jeweils von Seite zu Seite weiterzugeben. Dies brachte jedoch andere Probleme mit sich, so wird dazu z.B. immer ein Formular benötigt und bei einer Seitenaktualisierung könnten die POST-Daten erneut gesendet werden, oder eben nicht. Die implementierte Lösung greift nun wieder auf SESSION-Variabeln zurück, jedoch sind diese nun immer eindeutig. Dies wird erreicht, indem jede Variable zusätzlich durch die SessionId definiert wird. Statt $_SESSION[»questions«] heisst die Variable nun $_SESSION[»questions«. $session_id] also z.B. $_SESSION[»questions377«].

Die Implementation der eindeutigen SESSION-Variabeln löste gleichzeitig auch das Problem, der Benutzermanipulation. Alle Daten, die über GET übermittelt werden, können schliesslich vom User beliebig manipuliert werden. Eine Manipulation der SessionId oder der anderen Id's hätte aber keine Wirkung, da die dazugehörigen SESSION-Variabeln dann nicht existieren würde. Dies kann dann leicht abgefangen werden.

Auch ein Aktualisieren einer Seite, z.B. Solution war zu Beginn problematisch, da jedes mal eine neue Performance eingetragen wurde. Dies konnte abgefangen werden, indem geprüft wird, ob in der aktuellen Quizdurchführung schon eine Performance zur aktuellen Frage vorhanden ist. Dadurch passiert bei einer Seitenaktualisierung gar nichts.

5.2 Bewertungen

Die absolute Bewertung einer Frage ist die Aggregation aller abgegeben Bewertung. Dabei wird jede Bewertung als gleichwertig betrachtet.

Als Beispiel haben wir folgende Bewertungen:

3*
3*
4*
4*
5*

Dies ergibt demzufolge eine absolute Bewertung von 3.8. Die Sternanzeige wird dabei ab 0.5 aufgerundet auf 4 *.

Um nicht bei jeder abgegeben Bewertung alle zusammenzählen müssen, wird jede neue Bewertung zur absoluten Bewertung als Bruchteil der bereits abgegeben Bewertungen addiert. Dies ist schneller und benötigt weniger Speicherplatz in der Datenbank, zudem ist zu jederzeit ohne Rechnen bekannt was nun die absolute Bewertung ist.

Zu beachten ist, dass die Schwierigkeit erst ab einem frei konfigurierbaren Schwellenwert der Anzahl Bewertungen angezeigt wird. Dies wurde implementiert, da bei sehr wenigen Bewertungen kein wirklich repräsentativer Wert entsteht.

5.3 Schwierigkeit

Die Schwierigkeit wird durch die Beantwortungsstatistiken der User gewonnen. Bei jeder falsch Beantwortung steigt die Schwierigkeit und bei jeder korrekten Beantwortung sinkt diese. Dabei wird gleich wie bei den Bewertungen vorgegangen, mit den gleichen Beweggründen.

In der Zukunft könnten Beantwortungen von »guten« Usern schwerer gewichtet werden, als von »schlechten« Usern. Die dazu nötigen Daten sind vorhanden, die Implementierung konnte jedoch im Rahmen der SA aus Zeitgründen nicht mehr einbezogen werden.

5.4 Suche

Es existieren zweierlei Suchfunktionen in Quizzenger: Einerseits können alle Tabellen jeweils Clientseitig durchsucht werden. Diese Funktionalität wird von durch die DataTables Library ermöglicht.

Andererseits wurde auch eine serverseitige Suche integriert, welche direkt in die Navigationsleiste eingebunden wurde. Diese durchsucht aktuell alle Fragen nach dem Suchbegriff. Innerhalb der Frage werden der Fragetext sowie die Tags durchsucht. Da die tag-Tabelle in einer m:n-Beziehung zur question-Tabelle steht, ist die Suchquery alles andere als trivial. Die implementierte Query sieht folgendermassen aus ([PATTERN] ist der Suchbegriff):

```
SELECT q.*
FROM question q
LEFT JOIN (
        SELECT question_id,
        GROUP_CONCAT( tag.tag SEPARATOR ', ' ) AS tags
        FROM tagtoquestion ttq
        INNER JOIN tag ON tag.id = ttq.tag_id
        GROUP BY question_id
) t ON q.id = t.question_id
WHERE q.questiontext LIKE ("%[PATTERN]%")
OR t.tags LIKE ("%[PATTERN]%");
```

Zuerst werden alle Tags einer Frage gruppiert (GROUP BY question_id) und in einen kommagetrennten String zusammengefasst (GROUP_CONCAT). Dieser Tag-String wird schliesslich über einen JOIN mit der question-Tabelle verbunden. Das LIKE sorgt dafür, dass Gross-/Kleinschreibung ignoriert wird und die Modulozeichen (%), dass sowohl vor, als auch nach dem Pattern beliebige Zeichen folgen können. Rückgabewert der Query sind die vollständigen Data Rows der gefundenen Fragen aus der question-Tabelle.

5.5 Lernmodus

Beim Lernmodus wird das sogenannte temporäre Quiz als QuizID verwendet. Die ID ist -1, so kann eine Fallunterscheidung zu einem existierenden Quiz getroffen werden.

Im processGeneratequiz Controller werden die Formulardaten aufbereitet, die Filterdaten dem Quizmodel übergeben und erhält so einen Array mit den Questions. Nun wird direkt auf die erste Frage des Quizzes weitergeleitet. Ansonsten ist der Ablauf identisch zu der normalen Quiz durchführung.

5.6 Ajax

Auf einigen Seiten von Quizzenger müssen Teilinhalte dynamisch geladen werden oder Operationen auf dem Server durchgeführt werden, ohne dass die komplette Seite neu geladen wird. Ein Beispiel dafür ist z.B. wenn man beim Kategoriebaum auf eine Kategorie klickt, sollen die Subkategorien geladen werden. Vorab alle möglichen Kategorien zu laden macht keinen Sinn, da das sehr viele sein können und jedes Mal die Seite neu Laden wäre eine sehr schlechte Usability.

Zu diesem Zweck haben wir AJAX verwendet, welches einen HTTP-Request über Javascript ermöglicht. Das heisst es werden nur die Inhalte geladen, die man dynamisch nachladen muss, so entsteht kein überflüssiger Overhead.

Da der User schnelles Feedback auf seine Aktionen will, müssen AJAX-Requests möglichst Performant sein. Deshalb wird das blankContent-Template verwendet, welches leer ist.

Im ajax.js sind diverse Javascript-Funktionen definiert, welche einen HTTP-Request benötigen. Den Request an sich haben wir dabei in eine eigene Funktion ausgelagert, da dieser immer gleich funktioniert:

```
function ajaxGET(url){
        var xmlhttp;
        if (window.XMLHttpRequest) {
                xmlhttp = new XMLHttpRequest();
        }
        xmlhttp.open("GET", url, true);
        xmlhttp.send();
}
```

Die URL die aufgerufen wird beinhaltet dabei alle wichtigen Informationen für den HTTP-Request. Ein Aufruf der ajaxGET-Funktion sieht dann z.B. folgendermassen aus:

```
function deleteQuestion(question){
        ajaxGET("index.php?view=remove_question&type=ajax&question=" +
            question+"");
}
```

Durch das Argument type=ajax wird der AJAX-Controller aufgerufen. Analog zum normalen Controller, wird anhand des view-Arguments entschieden was zu tun ist. Das question-Argument ist schliesslich ein Funktionrelevantes Argument und dient zur Identifikation der zu löschenden Frage. Bei diesem Beispiel wird nur eine Operation ausgeführt (löschen), jedoch nichts zurückgegeben. Teilweise war es aber nötig, Inhalte wie z.B. Datenbankinhalte dynamisch zu laden. Dies ist z.B. der Fall wenn von einer Kategorie dynamisch die Unterkategorien geladen werden müssen. Der URL-Aufruf gibt dann den HTML-Code zurück, der vom AJAX-Controller erzeugt wird. Dieser zurückgegebene HTML-Code kann dann wiederum in der Javascript-Funktion verwendet werden:

```
xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
                document.getElementById(containerId).innerHTML = xmlhttp.
                    responseText;
        }
}
```

5.7 Mobiletauglichkeit

Bootstrap wird im Header eingebunden um Responsive Design elegant und effizient zu ermöglichen. Verwendet wurde die zur Zeit der Entwicklung aktuelle Version 3.2.0. Bootstrap wird im Template »skeleton.php« eingebunden.

5.8 Sessionmangement

Das Sessionmanagment geschieht naheliegenderweise über die PHP Session Variablen. Der Anmeldevorgang geschieht in »models/sessionmodel.php« in der »processLogin« Methode.

5.8.1 Login

Bei erfolgreicher Überprüfung des Users werden folgende Session Variablen gesetzt:

```
$user_browser = $_SERVER ['HTTP_USER_AGENT'];
$user_id = preg_replace ( "/[^0-9]+/", "", $user_id );
$_SESSION ['user_id'] = $user_id;
$username = preg_replace ( "/[^a-zA-Z0-9_\-]+/", "", $username );
$_SESSION ['username'] = $username;
$_SESSION ['login_string'] = hash ( 'sha512', $password . $user_browser );
$_SESSION ['email'] = $email;
$_SESSION['superuser']=($superuser==1?true:false);
```

5.8.2 Logout

```
//Clean up properly in orde to destroy session for good
$_SESSION = array (); // Unset all session values
$params = session_get_cookie_params (); // get session parameters so we an
    delete the cookie

// Renders it invalid / deleted
setcookie ( session_name (), '', time () - 42000, $params ["path"], $params
    ["domain"], $params ["secure"], $params ["httponly"] );

// Bye!
session_destroy ();
header('Location: index.php?info=mes_logout_success');
die();
```

5.8.3 Überprüfung

Bei jeder Seite muss überprüft werden,ob der Benutzer noch eingeloggt ist. Dies passiert bei jedem Seitenaufbau und wurde so implementiert, dass falls ein Benutzer nicht angemeldet ist, dieser automatisch die ID des Gast Users kriegt und so später im System identifiziert wird.

```
if ($this->login_check ( $this->mysqli )) {
    $GLOBALS ['loggedin'] = true;
} else {
    $GLOBALS ['loggedin'] = false;
    $_SESSION['user_id'] = 1; // ID=1 is Guest User
}
```

5.9 SQL Helper

Um Queries einheitlich und komfortabel auszuführen wurde im SQL Helper die Funktionen »s_query« sowie »s_insert« implementiert (das s_ steht dabei für Statement).

Da MySQLi die Werte sowie die Wertetypen als Varargs [9] entgegennimmt und diese Variabel in der Anzahl sind mussten wir uns etwas einfallen lassen. Dies da wir bei den Varargs nicht mehr richtig unterscheiden können, was nun ein Wert ist oder ein Wertetyp.

Beispiel Aufruf:

```
s_insert(INSERT INTO question (type,questiontext,user_id,category_id) VALUES
     (?,?,?,?) , $type,$questiontext,$user_id,$category_id,$string,$string,
    $int,$int)
```

Die entgegennehmende Funktion müsste dies nun sehr umständlich ermitteln.

Deshalb:

```
s_insert(INSERT INTO question (type,questiontext,user_id,category_id) VALUES
     (?,?,?,?) ,array($type,$questiontext,$user_id,$category_id),array(
    $string,$string,$int,$int))
```

Nun müssen wir uns nur noch mit zwei zusätzlichen Parametern herumschlagen, das Problem ist nun, wie die Funktion von MySQLi aufgerufen wird, die keine Arrays als Parameter will.

Gelöst mit:

```
call_user_func_array(array($stmt, 'bind_param'), $a_params);
```

Zudem haben wir ein optionales Flag »rowCheck« eingeführt, welches falls die Query keine Ergebnisse zurückliefert den Benutzer auf eine Fehlerseite weiterleitet. Dieses Flag wird benutzt, falls wir uns sicher sind, dass es mindestens ein Ergebnis gibt und keine Fehlerbehandlung anwenden können. Meist wird dies durch die Manipulation durch den User in der URL / Post Data herbeigeführt.

5.10 Frameworks & Libraries

Anstatt das Rad neu zu erfinden wurde auf gewisse Frameworks und Libraries zurückgegriffen. Wir haben diese dort verwendet, an Stellen wo die gleiche Funktionalität nicht in einem Semester erreichbar wäre und wo der Code nicht Produkte spezifisch ist.

Zudem steckt in diesen Libraries einiges an Erfahrung und eine Userbase.

[9] https://de.wikipedia.org/wiki/Variadische_Funktion

5.10.1 Fremd Code - Deklarierung

Jeglicher »Fremd Code« ist durch den Header erkennbar und ist in der nachfolgenden Liste aufgeführt:

- Ordner CSS
  /bootstrap-theme.min.css [BootStrap]
  /bootstrap.css [BootStrap]
  /bootstrap.min.css [BootStrap]
- Ordner DataTables:
  Ganzes Verzeichnis [DataTables]
- Ordner Fonts:
  Ganzes Verzeichnis [Bootstrap]
- Ordner Includes:
  /includes sqlhelper.php [Stackoverflow] , in der Funktion doStatement, markiert, Public Domain
- Ordner JS:
  /js bootstrap.min.js [Bootstrap]
  /jquery-1.11.1.min.js [Bootstrap]

5.10.2 DataTables

DataTables[10] ermöglicht es HTML Tables mit JQuery in interaktive Tabellen zu verwandeln.

Diese bieten folgende Features und könnten noch durch 'Extensions' erweitert werden:

- Volltext Filter
- Pagination
- Sortierung von Spalten

Der grosse Vorteil an DataTables gegenüber einer selbst programmierten Lösung ist, dass dies alles auf Seiten des Client passiert und die Einbindung sehr einfach ist. Die Initialisierung von DataTables pro Tabelle geschieht in »js/jqueryfunctions.js«.

Lizenz DataTables wurde unter der MIT [11] Lizenz veröffentlicht.

5.10.3 Bootstrap

Bootstrap ermöglicht es Responsive Design elegant und effizient zu verwirklichen.

Wir haben uns für Bootstrap entschieden, da die Lizenz uns gefällt und Vorkenntnisse von beiden Teammitgliedern bereits vorhanden war. Die Einbindung geschieht in »skeleton.php«.

Lizenz Bootstrap wurde unter der MIT [12] Lizenz veröffentlicht.

5.10.4 Bootstrap Validator

Bootstrap Validator ist ein mächtiges Werkzeug für die Überprüfung von Formularen und bietet Validatoren für alle möglichen Einsatzzwecke.

Die Verwendung von Bootstrap Validator war klar, da wir bereits Bootstrap und jQuery verwenden und da die Dokuentation sowie der Funktionsumfang uns überzeugt haben. Die Einbindung geschieht in »js/jqueryfunctions.js«.

[10] http://datatables.net/
[11] http://datatables.net/license/mit
[12] https://github.com/twbs/bootstrap/blob/master/LICENSE

Lizenz Bootstrap Validator wurde der Creative Commons BY-NC-ND 3.0 [13] Lizenz veröffentlicht.

Da Quizzenger kein kommerzielles Produkt ist und die Studienarbeit in keinem bezahlten Arbeitsverhältnis steht, stellt dies kein Problem dar.

5.11 Logging

Um eventuelle Fehlermeldungen sowie Informationen zum Systemstatus vor dem Benutzer zu verstecken und zu sammeln wurde ein Logging System implementiert.

Dies bring folgende Vorteile mit sich:

- Dem Benutzer werden nur spezifische zielführende Fehlermeldungen oder Informationen angezeigt ohne ihn zu verunsichern
- Ereignisse und Protokolle können durchaus sensitive Informationen beinhalten, diese können durch eine zentrale Ablage gesichert werden
- Bei Fehlern oder Tests kann das Log äusserst hilfreiche Hinweise geben

Das Log wird hauptsächlich in den Models verwendet, dort werden alle CRUD Operationen auf der Datenbank sowie allfällige Fehler oder unerlaubte Operationen gespeichert und bei Bedarf dem Benutzer mitgeteilt.

5.11.1 Aufteilung

Damit die Logdateien nicht zu gross werden und die Übersichtlichkeit gewahrt wird, gibt es pro Tag ein neues Logfile welches das besagte Datum trägt. Sobald ein Logeintrag angeforder wird und keine Datei besteht wird eine neue Angelegt. So wird bei Nichtbenutzung auch keine Logdatei angelegt.

5.11.2 Aufbau

```
[$d.m.Y - H:i:s$] . $messageLevel . " - " .
$message. " [IP: IP_Adresse - User: Username falls Angemeldet]
```

5.11.3 Loglevel

Um eine Kategorisierung oder Filterung von Logeinträgen zu ermöglichen wird bei jedem Eintrag ein Loglevel angefügt.

INFO - Meldungen wie Logins oder andere Events die bei Auswertungen interessante Informationen liefern (falls nichts angegeben wird, ist dies der Standard)

WARNING - Ein Fehler der mittels Fehlerbehandlung verbessert wurden konnte und problematische Ereignisse wie ungültige Logins

ERROR - Ein Fehler der nicht auftreten sollte, also zum Beispiel eine ungültige SQL Abfrage

FATAL - Ein schwerwiegender Fehler der den Betrieb der Anwendung verunmöglicht, zum Beispiel wenn die Datenbank nicht erreichbar ist

5.12 Konfiguration

Um den Betrieb und die Wartung von Quizzenger zu erleichtern werden können möglichst viele Parameter in einer Konfigurationsdatei angepasst werden.

Diese Datei befindet sich unter »/include/config.php«. Untenstehend befinden sich die Standardeinstellungen sowie zusätzliche Erklärungen wo diese nicht selbsterklärend sind.

Die Info- und Fehlermeldungen wurden entfernt.

[13] http://bootstrapvalidator.com/license/

```
// Benutzernamen welcher für die Verbindung zur Datenbank genutzt werden soll
define ( "dbuser", "csq" );
// Passwort für den besagten Benutzer
define ( "dbpassword", "********" );
// Datenbanknamen
define ( "db", "csq" );
// IP Adresse zu dem Server auf dem die Datenbank
define ( "dbhost", "localhost" );
// Port auf dem die SQL Instanz läuft
define ( "dbport", "3306" );

// Legt den FQDN der Anwendung fest, wird benutzt um Links etc zu genereiren
define ("APP_PATH","https://sinv-56035.edu.hsr.ch/csq");

// Anzahl Antworten für den Typ SingleChoice
define ("SINGLECHOICE_ANSWER_COUNT","4");
// Typ SingleChoice
define ("SINGLECHOICE_TYPE","SingleChoice");
define ("QUESTION_INPUTFIELD_MAX_LENGTH","320");
define ("QUESTION_INPUTFIELD_MAX_ROWCOUNT","3");
define ("ANSWER_INPUTFIELD_MAX_LENGTH","160");
define ("ANSWER_INPUTFIELD_ROWCOUNT","2");
define ("ANSWER_EXPLANATION_INPUTFIELD_MAX_LENGTH","320");
define ("ANSWER_EXPLANATION_INPUTFIELD_ROWCOUNT","1");
// Text der einem Benutzernamen angefügt wird, wenn dieser als inaktiv gesetzt wird
define ("USER_INACTIVE_NAME_ADDITION","(inaktiv)");
define ("DIFFERENT_QUESTION_WEIGHTS", "5");
define ("QUESTION_ANSWERED_SCORE", "2");
define ("QUESTION_CREATED_SCORE", "5");
define ("ADD_RATING_SCORE", "1");
define ("QUIZ_TAKEN_SCORE", "1");
define ("MODERATION_SCORE", "100");
define ("RATING_MAX_STARS", "5");
define ("QUESTIONTEXT_CUTOFF_LENGTH",75);
// Anzahl der Datensätze ab welchem eine Difficulty
// als Aussagekräfig gilt und angezeigt wird
define ("MIN_DIFFICULTY_COUNT_NEEDED_TO_SHOW",5);
// Anzahl der angezeigten Einträge der History
define ("QUESTIONHISTORY_NEWEST_SHOWNCOUNT",10);
// Blendet die Erstellungszeit der aktuellen Seite ein
define ("SHOW_PROCESSING_TIME",true);

// Log Settings
// ------------
define ("LOGGING_ACTIVE",true);
define ("LOGPATH","log/");

// Security Login / Register
// -------------------------
// Setzt Cookie Sicherheits Policy
define ( "SECURE", TRUE ); // FALSE only for debugging
// Setzt die Zeit in Sekunden für die ein Account gesperrt wird
define ( "BRUTE_FORCE_COOLDOWN", "600"); // in seconds (600 = 10 minutes)
// Setzt die Anzahl Versuche bis ein Account gesperrt wird bei falschem PW
define ( "BRUTE_FORCE_MAX_ATTEMPTS","5");
// Hiermit kann man die Bruteforce Überprüfung de/aktivieren
define ( "BRUTE_FORCE_CHECK","TRUE");
```

Bis auf einige Parameter kann diese Datei ohne Änderungen so eingesetzt werden, lediglich die Datenbank und der FQDN[14] muss vor der Installation benutzerspezifisch angepasst werden.

[14] http://www.itwissen.info/definition/lexikon/fully-qualified-domain-name-FQDN.html

5.13 Sicherheit

Da es sich um eine öffentliche Webapplikation handelt wurde viel Wert auf die Sicherheit gelegt und versucht, das gelernte aus den Modulen Infsi1/2 einzubringen.

5.13.1 Authentifizierung

Die Authentifizierung erfolgt über ein HTML Form welches die Logindaten sendet.

Die PHP Seite nimmt die Daten entgegen, hasht das Passwort und prüft die Anzahl fehlgeschlagener Loginversuche sowie die Richtigkeit der Logindaten.

Falls alles in Ordnung ist, wird eine sichere PHP Session aufgebaut.

5.13.2 Sicherheitsmassnahmen

HTTPS Um eine gesicherte Verbindung zwischen dem Browser des Benutzers und dem Server herzustellen wird HTTPS verwendet.

Im Rahmen unserer Studienarbeit wurde ein selbst signiertes Zertifikat erstellt.

Um sicherzustellen, dass der Benutzer die Seite nur verschlüsselt aufrufen kann, wurde in index.php folgender Code verwendet:

```
// We don't want HTTP connections
if ((FORCE_HTTPS_CONNECTION && $_SERVER ['HTTPS'] != "on" )) {
    header ( "HTTP/1.1 301 Moved Permanently" );
    $redirect = "https://" . $_SERVER ['HTTP_HOST'] . $_SERVER ['REQUEST_URI'];
    header ( "Location: $redirect" );
    die ();
}
```

Dieser Code leitet auf konforme Art [15] den User auf HTTP:// um falls dieser versucht mit HTTP:// sich zu verbinden.

Brute Force Schutz Um Brute Force zu erschweren werden nach einer bestimmten Anzahl falscher Login Versuche per Usernamen dieser User für eine bestimmte Zeitdauer gesperrt.

Die Überprüfung, falls aktiv, wird bei jedem Login überprüft und hindert bei zu vielen falschen Versuchen den User temporär beim Anmeldevorgang.

Schutz vor SQL Injections Um SQL Injections vorzubeugen werden alle Queries auf die Datenbank durch 'Prepared Statements' ausgeführt.

Dies bietet einen effektiven und effizienten Schutz vor SQL Injections in Quizzenger.

Um redundanten Code zu vermeiden wurde eine Funktion erstellt die sich um das Binding sowie das Ausführen der Statements kümmert.

Da bei PHP die Bindings mittels Varargs geschehen, ist das auslagern in eine dynamische Funktion nicht ganz einfach, wie hier zu sehen ist:

```
$a_params = array();
$param_type = '';
$n = (count($a_param_type)==null)?0:count($a_param_type);
for($i = 0; $i < $n; $i++) {
        $param_type .= $a_param_type[$i];
}
$a_params[] = & $param_type;
for($i = 0; $i < $n; $i++) {
        $a_params[] = & $a_bind_params[$i];
}
if($n!=0){ // only if there are parameters to bind
```

[15] https://www.ietf.org/rfc/rfc2616.txt - ' 10.3.2 301 Moved Permanently'

```
            call_user_func_array(array($stmt, 'bind_param'), $a_params);
    }
```

Dies liefert einen assoziativen Array zurück, so wie bei einfachen Querys.

Das entgegennehmen des Resultates ist Dank des MySQLnd (Native Driver), ziemlich einfach:

```
$result = $stmt->get_result();
```

Filtern von URL Parametern URL Parameter die eventuell ausgegeben werden oder weiterverarbeitet werden, werden als erstes »escaped«, diese Funktionalität bietet PHP 5 selbst:

```
$error = filter_input(INPUT_GET, 'err', $filter = FILTER_SANITIZE_SPECIAL_CHARS);
```

Persistenz der Passwörter Die Passwörter werden als erstes mit SHA512 auf dem Server gehasht.

Zuerst wollte clientseitiges Hashing verwendet werden, doch mehrere Probleme (nicht crossplattform taugliche Implementationen und zu frühes absenden der Formulare) sprachen dagegen.

Das gehashte Passwort wird beim Server mittels Salt erneut gehasht und verglichen, dies verhindert das benutzen von Rainbow Tables.

Der Salt Wert wird mit einer möglichst hohen Entropie und wird für jeden User eigens generiert. So erhöht sich der Aufwand für einen potentiellen Angreifer sehr. Erstellung des Salt und erstellen des Passwort Hashes bei der Registration:

```
$random_salt = hash('sha512', uniqid(mt_rand(), true));
$password = hash('sha512', $password . $random_salt);
```

Fehlerbehandlung Bei Fehlern wird auf die Error Seite verwiesen und danach sofort mittels die() jegliche weiteren Anweisungen unterbrochen, sodass allfällige ungewünschte Nebeneffekte verhindert werden können.

Hier ein Beispiel:

```
header('Location: ../index.php?view=error&err=err_register_insert');
die();
```

Konstante URL Parameter Um Cross Site Scripting in den URL Parametern zu unterbinden werden alle Parameter als Konstanten in config.php erfasst.

Auf den auszuwertenden Seiten werden alle nicht bekannten Parameter verworfen.

Definition einer Variablen:

```
define ( "err_register_insert",
"Erstellen Ihres Benutzers ist fehlgeschlagen - Insert fail" );
```

Weiterleitung auf die Error Page mit der Konstanten:

```
header('Location: ../index.php?view=error&err=err_register_insert');
```

Mittels constant erhält man nun den Wert der Konstanten, falls dies nicht klappt können wir davon ausgehen, dass der Parameter verändert wurde und verworfen werden kann,

```
$errorMessage= constant($error);
```

6 Quellverzeichnis

1 http://www.wikipedia.org . . . 5
2 http://getbootstrap.com . . . 5
3 http://cloc.sourceforge.net/ . . . 9
4 http://gitstats.sourceforge.net/ . . . 10
5 http://www.nngroup.com/articles/website-response-times/ . . . 13
6 http://php.net/manual/de/book.mysqli.php . . . 19
7 http://refresh-sf.com/yui/ . . . 22
8 http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html . . . 22
9 https://de.wikipedia.org/wiki/Variadische_Funktion . . . 27
10 http://datatables.net/ . . . 28
11 http://datatables.net/license/mit . . . 28
12 https://github.com/twbs/bootstrap/blob/master/LICENSE . . . 28
13 http://bootstrapvalidator.com/license/ . . . 29
14 http://www.itwissen.info/definition/lexikon/fully-qualified-domain-name-FQDN.html . . . 30
15 https://www.ietf.org/rfc/rfc2616.txt - ' 10.3.2 301 Moved Permanently' . . . 31

7 Anhänge

Alle Anhänge sind im Unterordner »Anhänge« zu finden.

Dokument	**Beschreibung**
1_Eigenständigkeitserklärung	Die unterschriebene Eigenständigkeitserklärung
2_Aufgabenstellung	Die von dem Betreuer abgegeben Aufgabenstellung
3_Screenshots	Ordner mit allen Browsertest Screenshots
4_GIT_Statistics	Ordner mit den generierten Git Statistiken (index.html)
5_Benutzerhandbuch	Das Benutzerhandbuch für Quizzenger
6_Sitzungsprotokolle	Ordner mit allen Sitzungsprotokollen
7_Poster	Zusammenfassung der Arbeit auf einem Poster
8_Kurzfassung	Kurzfassung
9_Installationsanleitung	Die Installationsanleitung für Quizzenger
10_Projektplan	Projektplan
11_Persoenliche_Statements	Die persönlichen Statements zur Studienarbeit
12_Quizzenger_Source	Der komplette Sourcecode sowie Inhalte von Quizzenger
13_API	Die API von Quizzenger für Entwickler