HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Crowdsourced Quizzes 2

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühlingssemester 2015

Autor(en): Rico Beti, Simon Zingg
Betreuer: Prof. Frank Koch

Quizzenger 2 - Technischer Bericht

Rico Beti (rbeti@hsr.ch)
Simon Zingg (szingg@hsr.ch)

Dienstag, 26. Mai 2015

1. Abstract

Während der vorhergehenden Studienarbeit "Quizzenger 1" wurde ein Quiz-System auf Basis von Crowdsourcing nach dem Bilde Wikipedias[1] geschaffen, wobei Fragen im Zentrum stehen. Dies erlaubt es den Benutzern, ihr Wissen in verschiedenen Fachgebieten mittels verfügbaren Fragen und Quizzes zu testen und zu erweitern, sowie auch selbst neue Fragen der Crowd zur Verfügung zu stellen.

Die hier vorliegende Studienarbeit "Quizzenger 2" erweitert und verbessert dieses System und behandelt insbesondere folgende zwei Kernfragen:

1. Wie können Benutzer wirklich dazu motiviert werden, Fragen zu verfassen und sich aktiv an der Community zu beteiligen?

2. Wie kann das System interessanter und ansprechender gestaltet werden, um Benutzern eine bessere Learning-Experience zu bieten?

Um den Anforderungen gerecht zu werden, wurde im Rahmen dieser Studienarbeit ein ausgeklügelter Ansatz zur Gamification implementiert. Dabei werden Spieler aktiv durch ihre Teilnahme an der Community mit der Vergabe von Punkten, Achievements und Rängen belohnt und können sich so untereinander vergleichen. Mit der Einführung eines Spiele-Modus können mehrere Benutzer in Echtzeit gegeneinander zum Duell antreten und ihren aktuellen Wissensstand unter Beweis stellen.

Viele weitere Features sind während diesem Semester umgesetzt worden und werden ebenfalls nachfolgend in diesem Bericht beschrieben. Zusätzlich wurden am bestehenden System "Quizzenger 1" mehrere Refactorings durchgeführt, wobei identifizierte Mängel behoben und Sicherheitslücken geschlossen wurden.

[1] `http://www.wikipedia.org/`

2. Änderungsnachweis

Datum	Version	Änderung	Autor(en)
26.02.2015	1.0	Initial	Rico Beti, Simon Zingg
15.05.2015	1.1	Ergänzungen	Rico Beti, Simon Zingg
25.05.2015	1.2	Abgabe	Rico Beti, Simon Zingg

Inhaltsverzeichnis

3. Ausgangslage

Tests dienen nicht nur der Erfolgskontrolle, sondern vor allem dem Lernen. Als Quizzes können Testfragen via Web oder als mobile App angeboten und automatisch korrigiert werden. Dabei entstehen umfangreiche Daten zur Qualität der Fragen, der Autoren sowie der Benutzer. Das Erstellen guter Testfragen ist für diesen Zweck aufwendig, weshalb die Crowd für die Erstellung und Qualitätssicherung des Inhaltes genutzt werden soll.

Im Rahmen der Semesterarbeit "Crowdsourced Quizzes 1" wurde eine webbasierte Lernplattform erstellt, in der die Crowd/Community Fragen für verschiedene Themenbereiche erfassen, diskutieren und bewerten sowie in Form von Quizzes beantworten kann. Das System bewertet die Qualität der einzelnen Fragen, Autoren und Benutzer mittels statistischer Analyse. Die Benutzer haben zudem die Möglichkeit, Fragen zu bewerten, zu melden sowie Feedback an deren Autor zu richten. Im Rahmen der Semesterarbeit "Crowdsourced Quizzes 2" soll das bestehende System wie folgt erweitert werden:

1. Ein Gamification-Ansatz soll die Attraktivität der Plattform steigern sowie die Verbreitung des Systems fördern.
2. Ein Anreizsystem für das Einstellen neuer Fragen soll erstellt werden.
3. Ein Reporting zu Fragen und Usern soll erstellt werden.
4. Das Exportieren und Importieren von Fragebeständen soll ermöglicht werden.
5. Die Fragetypen sollen multimedial erweitert werden.
6. Optionale Erweiterungen
   (a) Anpassung des Dialogs zwischen Benutzern und Autoren für das Kommentieren der in Quizzenger verfügbaren Fragen.
   (b) Machbarkeitsanalyse einer Social Media Integration.
   (c) Benutzer sollen News posten und diese gegenseitig "abonnieren" können.

4. Wie funktioniert Quizzenger?

Aufgrund der grossen Menge an funktionalen Anforderungen, die in "Crowdsourced Quizzes 1" umgesetzt wurden, sind andere Punkte wie kontinuierliches Refactoring, das aussagekräftige Kommentieren von Code, die Erstellung einer ausführlichen Installationsanleitung usw. zu kurz gekommen. Dadurch war unter anderem die Einarbeitung in den Code des bestehenden Systems schwierig und daher sehr zeitintensiv.

Dieser Abschnitt hat das Ziel, das neu gewonnene Know-how über Quizzenger weiterzugeben um Studenten zukünftiger Studienarbeiten die Einarbeitung zu erleichtern.

An dieser Stelle sei auf Kapitel 4.2 des technischen Berichtes von "Crowdsourced Quizzes 1" verwiesen, wo die hier aufgeführten Themen aus einer anderen Perspektive erklärt werden.

4.1. Routing

Quizzenger wurde als Single Page Applikation entwickelt. Der Systemeintrittspunkt ist also immer die Seite `index.php`. Typische URLs sehen immer wie folgt aus:

```
1 // Generelle Struktur.
2 index.php?view=[ControllerName]
3     &[param1]=[value1] &[param2]=[value2]
4 // Beispiel eines Aufrufes innerhalb eines Requests.
5 index.php?view=question&id=171
```

Nachfolgend in Abbildung 1 wird der Ablauf eines Requests gezeigt.

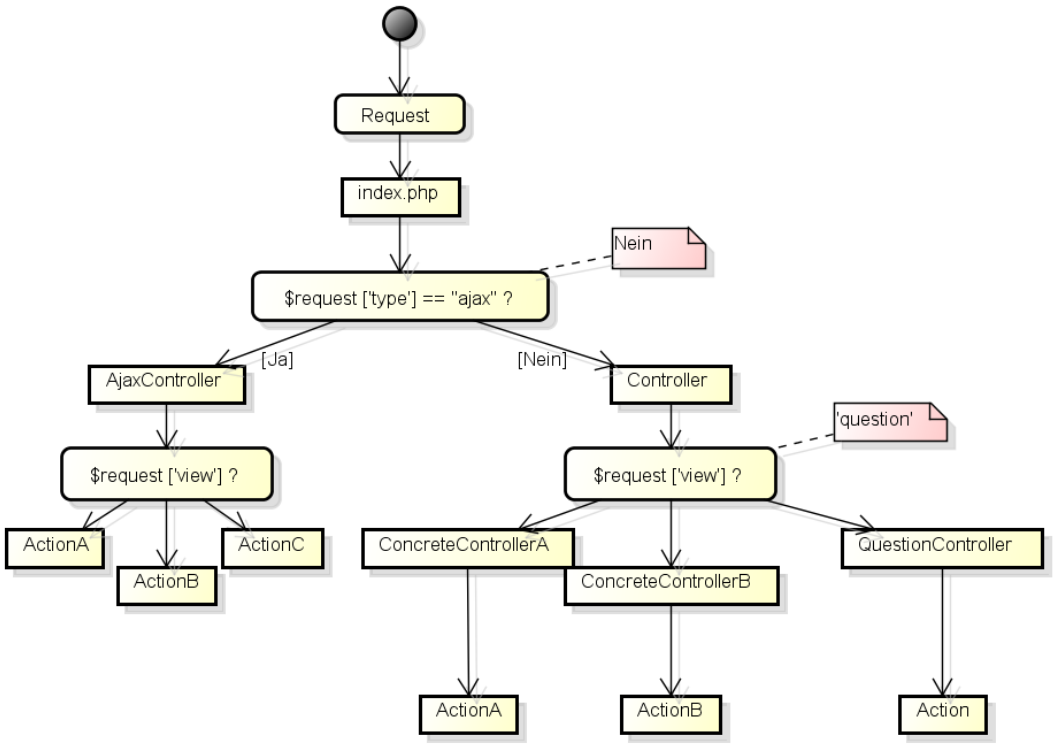


Abbildung 1: Aufrufe innerhalb eines Requests

Bei einem Request auf `index.php?view=question&id=171` wird zuerst die Datei `index.php` aufgerufen. Diese überprüft, ob ein Parameter "type" mit dem Wert "ajax" gesetzt ist. Da dies beim aktuellen Request nicht der Fall ist, wird der "Controller" aufgerufen. Dieser prüft seinerseits den Wert des Parameters "view" und leitet den Request an den "QuestionController" weiter.

Nebenbei sei erwähnt, dass der "AjaxController" und der "Controller" als Front-Controller fungieren. Diese beiden Controller sind zentral und kapseln das gemeinsame Verhalten aller Requests. So lädt der Controller das "Skeleton", also das Grundgerüst der Seite.

Alle anderen Controller, wie z.B. der "QuestionController", sind Page Controller. Diese sind jeweils ausschliesslich für eine individuelle Seite zuständig. Sie laden die

benötigten Daten vom Model und geben sie an das Template (View) weiter.

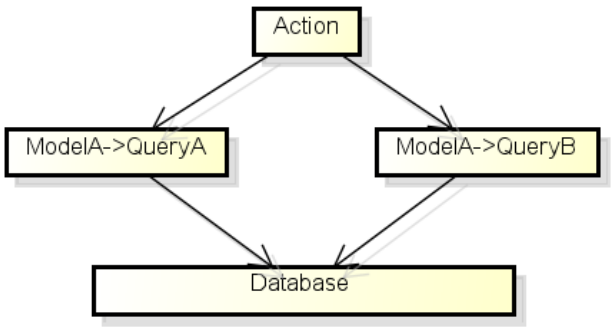


Abbildung 2: Aufrufe der Controller Action-Methode

In Abbildung 2 sieht man, wie die Action-Methode des jeweiligen Page Controllers, im vorliegenden Fall des "QuestionControllers", alle benötigten Daten vom Model lädt. Bei Quizzenger verschmelzen der Business- und der Persistence-Layer in den Models zusammen. Ein gutes Beispiel hierfür ist das "SessionModel", welches sowohl Datenbankabfragen als auch Business Logik enthält. Diese Lösung entspricht wohl am ehesten dem *Active Record Pattern*[2], mit dem Unterschied, dass die Quizzenger-Models vorwiegend auf das Domain Model und nicht auf das Datenbank Model bezogen sind.

4.2. Template-System

Den Kern des Template-Systems bildet die Klasse `View` aus Abbildung 3.

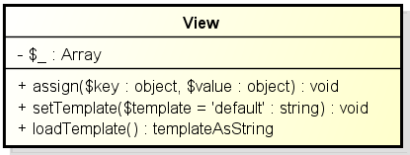


Abbildung 3: Klassendiagramm View.php

Wir knüpfen am vorherigen Beispiel an. Der Controller lädt das Skeleton als Grundgerüst, es erscheint also auf jeder Seite:

```
1 $viewOuter = new View(); // Neue View erstellen.
2 $viewOuter->setTemplate('skeleton'); // Setzen des Templates.
3
4 // Zuweisen von Platzhaltern und Werten,
5 // welche an das Template übergeben werden.
6 $viewOuter->assign('userid', $_SESSION['user_id']);
7 $viewOuter->assign('username', $_SESSION ['username']);
```

Anschliessend erstellt der Controller eine innere View, welche er der zuständigen Klasse `QuestionController` übergibt. `QuestionController` setzt seinerseits ein Template:

[2] `http://www.martinfowler.com/eaaCatalog/activeRecord.html`

```
$viewInner = new View();
$controller = new QuestionController($viewInner);

// In der Methode render() setzt der QuestionController das
// Template 'question' und füllt Platzhalter wie Frage,
// Antworten, usw. mit Werten.
$viewInner = $controller->render();
```

Jetzt verknüpft der Controller die äussere und die innere View, indem er den Output der inneren View (`loadTemplate()`) der äusseren View als String übergibt.

```
$viewOuter->assign('csq_content', $viewInner->loadTemplate());
// Ausgabe im Tempalte 'skeleton' mittels der Anweisung:
// echo $_->['csq_content'];
```

Zuletzt muss noch die äussere View als String ausgegeben werden:

```
echo $viewOuter->loadTemplate();
```

Mithilfe der Klasse View können Templates nun beliebig tief verschachtelt werden, was die Arbeit mit komplexen Ansichten extrem erleichtert.

4.3. Dateistruktur des Systems

An dieser Stelle soll die Strukturierung des Codes sowie der Assets erläutert werden.

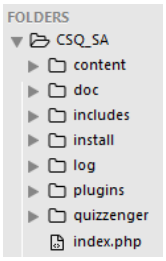


Abbildung 4: Projekt Ordner

Im Ordner `content` sind alle öffentlichen (public) Dateien wie Bilder, Stylesheets und JS-Scripts abgelegt. Der Ordner `doc` enthält die Doxygen-Dokumentation des Codes, im Ordner `log` sind die Log-Dateien abgelegt und im Verzeichnis `plugins` befinden sich alle Code-Plugins, durch welche sich das System dynamisch erweitern lässt (derzeit auf Achievements beschränkt, wie in Abschnitt 7.1.2 beschrieben).

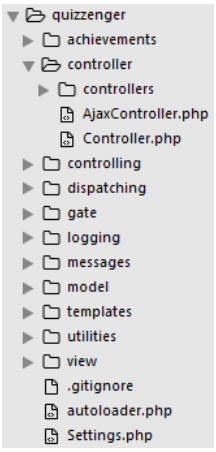


Abbildung 5: Der Quizzenger-Ordner

Der Ordner `quizzenger` enthält den gesamten selbstgeschriebenen PHP-Code des Quizzenger-Projekts. Eigenständige Teile, die im gesamten Code immer wieder verwendet werden, befinden sich in separaten Ordnern. Dazu gehören die Utilities, das Achievement-System (mit den entsprechenden Verzeichnissen `achievements`, `controlling` und `dispatching`), das Logging und das Messaging-System. Die Hauptordner `controller` und `model` enthalten zahlreiche Spezialisierungen. Meistens wird pro Requests nur eine Spezialisierung benötigt (siehe oben genanntes Beispiel mit dem `QuestionController`, welcher das `QuestionModel` benötigt).

5. Funktionale Anforderungen

Dieser Abschnitt erklärt die funktionalen Anforderungen an das System im Rahmen der Semesterarbeit *Crowdsourced Quizzes 2*.

5.1. Gamification

In der Aufgabenstellung wurde folgende Anforderung definiert:

- Ein Gamification-Ansatz soll die Attraktivität sowie die Verbreitung des Systems fördern.
  - *Entweder in der Art*, dass mehrere Benutzer gleichzeitig ein Quiz lösen. Eine Statistik über den Fortschritt des Spiels müsste dann in Echtzeit angezeigt werden. Diese Anzeige könnte via Beamer einen Live-Wettkampf im Klassenraum erzeugen. Ein Beispiel für diese Anforderung liefert das 'Space Race' von `http://www.socrative.com`.
  - *Oder in der Art*, dass Benutzer gegeneinander spielen. Dabei könnten mittels Filter verschiedene Sets von Fragen generiert werden (z.B. nach Schwierigkeitsgrad oder nach Bewertung). Das Spielschema erfolgt wie bei Quizduell (`http://www.androidpit.de/quizduell-ratespiel`).

In der aktuellen Gamification-Lösung wurde versucht, die Vorteile beider Anforderungen zu kombinieren. Es ist sowohl möglich, dass mehrere Benutzer gleichzeitig ein

'Game' spielen, als auch, dass sich zwei Spieler duellieren. In beiden Fällen wird dem Spieler eine Statistik in Echtzeit angezeigt, die ihn über den aktuellen Spielfortschritt aller Teilnehmer informiert. Dadurch besteht die Möglichkeit eines Live-Wettkampfs, der via Beamer angezeigt werden kann.

5.1.1. Game erstellen

Da Games im Grunde gleich aufgebaut sind wie Quizzes, werden neue Games aus bestehenden Quizzes erstellt. Will ein Benutzer also ein Game erstellen, so muss er zuerst über ein entsprechendes Quiz verfügen bzw. dieses anlegen. Der Vorteil besteht darin, dass dadurch Games in sehr kurzer Zeit erstellt und wiederholt werden können. Games sind dementsprechend Quizzes, mit dem Unterschied, dass sie einen eigenen Namen und eine klar definierte Start- und Endzeit haben.

5.1.2. Game-Lobby

Über die Game-Lobby kann an offenen (d.h. noch nicht gestarteten) Games teilgenommen werden. Die Liste wird dabei in Echtzeit aktualisiert.

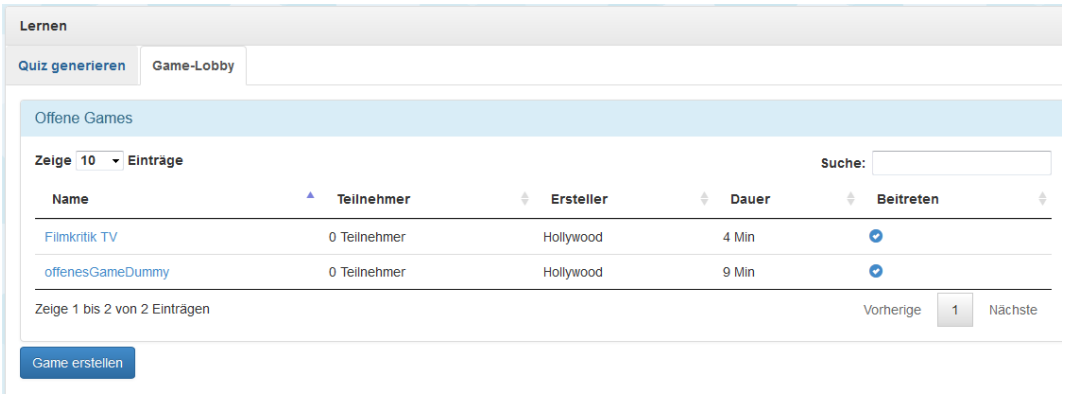


Abbildung 6: Game-Lobby

5.1.3. Game-Report

Der Game-Report in Abbildung 7 ist die Echtzeit-Statistik, welche den eigentlichen Gamification-Anreiz während eines Spieles ausmacht.

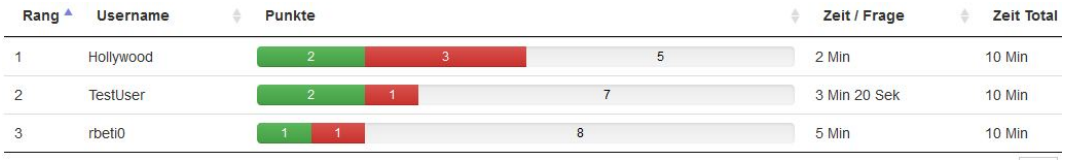


Abbildung 7: Game-Report

Der Report ist nach aktuellem Rang sortiert. Den ersten Rang erhält derjenige Teilnehmer, der momentan die höchste Punktezahl besitzt. Liegen zwei Teilnehmer gleich auf, so gewinnt der mit der kürzeren Antwortzeit pro Frage. Während die Spieler Fragen beantworten läuft ein Zähler mit, der den Teilnehmern laufend die

verbleibende Zeit bis zum Ende des Spieles anzeigt.

Eine grosse Herausforderung stellte die Datenbankabfrage für den Game-Report dar. Sie ist über 30 Zeilen lang, besitzt zwei Subqueries und enthält fünf Joins. Folgende Fälle mussten abgedeckt werden:

- Spieler, die *nicht alle* Fragen beantwortet haben
  - Game läuft noch
    Zeit-Total = Aktuelle Zeit – Startzeit
  - Game beendet
    Zeit-Total = Endzeit – Startzeit
- Spieler, die *alle* Fragen beantwortet haben
  - Zeit Total = 'Zeit letzte beantwortete Frage' – Startzeit

5.1.4. Live-Anzeige

Um die im Game-Umfeld geforderte Live-Anzeige realisieren zu können, werden die Daten periodisch über AJAX gepollt. Damit die gepollten Daten komfortabel in die aktuelle Ansicht eingebunden werden können, wurde das JavaScript-Template-Framework *doT* (siehe Abschnitt 7.3.2) eingesetzt.

5.1.5. Herausforderung Achievements

Durch das erfolgreiche Abschliessen eines Games können diverse Achievements erlangt werden, beispielsweise das Achievement "Gewinne ein Game mit 10 Teilnehmern". Da beim Game-End-Event mehrere Benutzer involviert sind, stellt sich die Frage, wer nun dieses Event auslöst. Dabei sind verschiedene Ansätze möglich:

- Jeder Spieler löst für sich selbst das Event aus.
  Das Problem besteht darin, dass ein Spieler möglicherweise nicht mehr online ist, wenn das Spiel zu Ende ist. Dies kann u.a. dann passieren, wenn ein Spieler erst 10 Minuten nach dem Gewinner fertig wird. Bei dieser Situation könnte der Gewinner ein Achievement verpassen.
- Ein Spieler löst das Event für alle anderen aus.
  Diese Lösung schränkt die Anzahl problematischer Fälle extrem ein, da nur noch ein einziger Spieler online sein muss, sobald das Game fertig ist. Trotzdem ist die Lösung nicht ausreichend, falls alle Spieler das Spiel vorzeitig abbrechen.
- Das System löst das Event für alle Spieler aus.
  Bei dieser Lösung muss keiner der Spieler online sein, wenn das Spiel zu Ende ist. Der spätmöglichste Zeitpunkt zum Game-End ergibt sich aus der Startzeit plus die maximale Spieldauer. Dadurch ist garantiert, dass jedes Game endet und jeder Spieler die Achievements erhält, die ihm zustehen.

Als Folge dieser Lösung müssen Nachrichten über den Request und die PHP-Session hinaus verschickt werden können. Deshalb wurde die Datenbanktabelle `message` erstellt, welche Benutzermeldungen persistiert und mittels der Tabelle `translation` übersetzt und entsprechend formatiert.

5.2. Anreizsystem zum Erstellen von Fragen

Eine der grössten konzeptionellen Herausforderungen von *Crowdsourced Quizzes 2* ist es, konsumierende User zu motivieren, selbst Fragen zur Crowd beizusteuern. Aus diesem Grund wurden Hintergrundinformationen zum Thema "Motivieren von Usern in Communities und Crowdsources Systemen" ausgewertet. Es wurde versucht, die gewonnenen Erkenntnisse aus dieser Recherche in die Studienarbeit einfliessen zu lassen.

In einem Bericht über die Beteiligung von Usern im Web werden unter anderem drei Beweggründe für aktive Teilnahme genannt[3]:

- *Eine erwartete Gegenleistung*
  Aktive Benutzer in einem Forum schneller Antworten als passive Benutzer.
- *Ein Gefühl der Gemeinschaft*
  "Ich gehöre dazu; ich möchte etwas zum grossen Ganzen beitragen."
- *Eine erhöhte Anerkennung*
  "Meine aktive Teilnahme wird zur Kenntnis genommen und belohnt."

5.2.1. Gegenleistung und Gemeinschaftsgefühl

> "Bei ergebnisbezogenen Entlohnungen unter Einsatz von Gamification sind Geldprämien, kleine monetäre Belohnungen, Vergünstigungen oder exklusive Informationen üblich (Paid Crowdsourcing). Allerdings gibt es viele Crowdsourcing-Projekte ohne finanzielle Anreize. Diese motivieren die Freiwilligen durch berufliche Vorteile, den Wunsch, Neues zu lernen, Wissen mit anderen zu teilen und gemeinsame Ziele zu erreichen. Bei jeglichen Projekten ist die Wahrnehmung der sozialen Anerkennung, einer sinnvollen und kreativen Arbeit, des Spaßes am gemeinsamen Handeln sowie des Gemeinschaftsgefühls für die Partizipierenden motivierend." [4]

Das Freischalten des Features "Reporting" (siehe **??**) ist eine Gegenleistung, die bei Quizzenger einen Anreiz bieten soll, Moderator zu werden. Trotzdem spielt eine solche Freischaltung eine eher untergeordnete Rolle.
Der ultimative Anreiz – welcher jedoch keiner technischen Realisierung bedarf – ist die Aussicht eines Schülers, bei grossem Engagement einen Notenzuschlag zu erhalten. Eine weitere Empfehlung ist, den Schülern bei jeder Übung die Aufgabe zu geben, 1-2 Fragen zum aktuellen Thema auf Quizzenger zu erfassen. Hier liegt also die Aufgabe bei den Dozenten, die Studenten zu motivieren.

5.2.2. Anerkennung

Eine Online-Community soll eine Funktion bieten, welche die Benutzer nach ihren Leistungen und Beiträgen analysieren und quantifizieren lässt. Dies vermittelt den Mitarbeitern das Gefühl, einzigartig und nützlich zu sein.

[3] `http://en.wikipedia.org/wiki/Online_participation`
[4] `http://de.wikipedia.org/wiki/Crowdsourcing#Strategischer_Einsatz`

Auf diese Art der Motivation wird in dieser zweiten SA viel Wert gelegt. Das Achievement- und das verbesserte Rangsystem (siehe Abschnitt 5.3) sollen dem Benutzer das Gefühl der Anerkennung verleihen indem er seine Leistungen abfragen und sich mit seinen Kollegen vergleichen kann.

5.2.3. Nielsen's 90-9-1 Regel

Ein Regelmässigkeit, die trotz aller Bemühungen um mehr Benutzerengagement nicht ausser Acht gelassen werden darf, ist die 90-9-1 Regel von Nielsen:

> "In most online communities, 90% of users are lurkers who never contribute, 9% of users contribute a little, and 1% of users account for almost all the action."[5]

Man muss sich also durchaus bewusst sein, dass auch die meisten Benutzer von Quizzenger sich ausschliesslich passiv verhalten und Inhalte konsumieren, ohne selbst eigene Fragen zu erstellen.

5.3. Scoring-Konzept

Im Zusammenhang mit dem Achievement- und Rangsystem musste eine konkrete Punkte-Verteilung konzipiert werden, die das Ziel verfolgt, die Benutzer zu mehr Teilnahme an der Community zu ermutigen.

Zu diesem Zweck wurden folgende Konzepte eingeführt:

Konzept	**Erklärung**
Rang	Der Rang widerspiegelt die Erfahrung eines Benutzers. Er entspricht dem Konzept der militärischen Ränge, welches eine grobe Einteilung von Personen erlaubt. Beispielsweise kann ein Benutzer vom "Neuling" über den "Alleskönner" zum "Profi" aufsteigen. Der globale Rang erlaubt den Vergleich zwischen Benutzern unterschiedlicher Kategorien, z.B. Informatiker und Elektrotechniker.
Rangliste	Unabhängig vom globalen Rang erscheint jeder Benutzer pro Kategorie auf einer Rangliste. Dadurch kann der Benutzer seine Leistung innerhalb einer Kategorie mit seinen direkten Konkurrenten messen. Gleichzeitig erfährt er, wieviele Punkte er noch zu erspielen braucht, um die Konkurrenten zu überholen.
Achievement	Ein Achievement ist ein einmaliges Ereignis (z.B. "Beantworte 10 Fragen korrekt"), welches eine Bonus Score auslöst. Diese Achievements lösen ein exponentielles Score-Verhalten aus, welches die Spreu vom Weizen trennt.

[5] `http://en.wikipedia.org/wiki/Online_participation`

Consumer Score	Ein Benutzer erhält Consumer Score, wenn er von der Community konsumiert (z.B. “Beantworte eine Frage”).
Producer Score	Ein Benutzer erhält Producer Score, wenn er in der Community aktiv ist und etwas dazu beiträgt. Beispiele für Producer Score sind “Eine Frage erstellen” oder “Eine Frage bewerten”. Um die Benutzer zu einem höheren Engagement zu bewegen, wird Producer Score um einiges höher gewichtet als Consumer Score.
Bonus Score	Eine Bonus Score kann keiner Kategorie zugeordnet werden. Sie wird zu den Gesamtpunkten dazugerechnet und dient dem Erreichen eines höheren Ranges.
Moderator	Die Punktzahl innerhalb einer Kategorie bestimmt die Rechte des Benutzers. So kann ein besonders aktiver Benutzer Moderator einer oder mehrerer Kategorien werden. Ein Moderator verwaltet seine Kategorie(n), indem er dafür sorgt, dass nutzlose Fragen geändert oder gelöscht werden. Ausserdem erhält ein Moderator das Zusatztool “Reporting” (siehe Abschnitt 5.4), welches ihm genauere Auswertungsmöglichkeiten und eine einfachere Verwaltung der Fragen ermöglicht.

Das aktuelle Scoring-Konzept ist so ausgelegt, dass von einem durchschnittlichen Benutzer ausgegangen wird, der 5 Fragen erstellt und 40 Fragen beantwortet. Die Idee ist, dass ein solcher Benutzer einen mittleren Rang erreicht.

Die Architektur des Achievement-Systems und der Aufbau der Plugins wird in Abschnitt 7.1 ausführlich erläutert.

5.4. Reporting

Die Umsetzung des im Rahmen dieses Projektes geplante Reporting-Systems ist durch folgende zwei Punkte motiviert:

- Verbesserte Übersicht über das System und Schaffen von Auswertungsmöglichkeiten für Superuser und Moderatoren.
- Bieten eines weiteren Anreizes, durch aktive Teilnahme an der Crowd Moderator einer oder mehreren Kategorien zu werden.

Die nachfolgenden Abschnitte beschreiben die vier Teile des Reporting-Systems.

5.4.1. Benutzerauflistung

Es soll eine Liste aller Benutzer angezeigt werden können. Diese soll nebst einer globalen Ansicht auch nach Kategorie sortiert werden können. Wichtige Daten sollen hierbei pro User leicht ersichtlich sein. Dazu gehören ID, Name, Erstellungsdatum, der Rang und die Punktzahl (aufgeteilt in Producer und Consumer Score).

5.4.2. Frageauflistung

Für eine bessere Content-Übersicht sollen alle Fragen aufgelistet werden können. Diese müssen sortierbar sein nach ID, Kategorie, Autor, Erstellungs- und Änderungsdatum, Bewertung, Schwierigkeitsgrad, sowie nach der Anzahl Durchführungen.

5.4.3. Autorenauflistung

Alle Autoren, d.h. Benutzer, welche sich mit mindestens einer Frage an der Crowd beteiligt haben, sollen ebenfalls aufgelistet werden können. Diese müssen unter anderem sortierbar sein nach Name, Anzahl erstellter Fragen, Durchschnitt der Bewertungen und dem Durchschnitt der Schwierigkeitsgrade über alle Fragen.

5.4.4. Systeminformationen

Zusätzlich zum Reporting der angemeldeten Benutzer und des hochgeladenen Inhaltes sollen ebenfalls einige technische Informationen zum System angezeigt werden. Diese Ansicht wird nur Benutzern mit Superuser-Berechtigung angezeigt. Diese sollen damit eine Übersicht über den aktuellen Systemzustand erhalten. Zu den darzustellenden Informationen gehören der Speicherverbrauch der Anhänge, die Grösse der Datenbank, fehlgeschlagene Login-Versuche innerhalb der letzten 24 Stunden sowie eine Auflistung aller Log-Dateien.

5.5. Import & Export von Fragen

Dieser Abschnitt beschreibt das `*.quizzegner`-Dateiformat, welches für den Import und Export von Fragen zwischen verschiedenen Quizzenger-Systemen entwickelt wurde. Die nachfolgend aufgeführte Spezifikation beschreibt das Dateiformat in der Version `1.0` und dient als Grundlage für zukünftige Erweiterungen.

5.5.1. Zweck & Nutzen

Das auf XML basierende Format soll es ermöglichen, Fragen versionsunabhängig zwischen verschiedenen Quizzenger-Systemen austauschen zu können. Dabei wird lediglich auf den Inhalt der Fragen, deren Antworten und Anhänge Wert gelegt. Auf konkrete Referenzen zu Benutzern oder anderen Systemdaten wurde bewusst verzichtet, damit die Eigenständigkeit gewährleistet werden kann.

5.5.2. Beispiel einer Datei

Das unten aufgeführte Beispiel stellt die zugrundeliegende Struktur des XML-Formates dar. Direkt anschliessend in Abschnitt 5.5.3 werden sämtliche Knoten und Attribute definiert und beschrieben.

```
<?xml version="1.0" encoding="utf-8" ?>
<quizzenger-question-export version="1.0">
    <meta>
        <system>http://www.quizzenger.hsr.ch</system>
        <date>2015-04-24 12:00:00</date>
    </meta>
    <questions>
        <question uuid="21f28554-f3df-11e4-84c0-0050568053a1"
                  type="SingleChoice" difficulty="34.2857">
            <author>Mike</author>
            <created>2015-01-25 18:25:00</created>
            <modified>2015-01-26 12:45:00</modified>
            <category first="Naturwissenschaften"
                      second="Geographie"
                      third="Länderkunde" />
            <text>
                Welches ist das flächenmässig grösste Land?
                [attachment]
            </text>
            <answers>
                <answer correctness="0">
                    <text>Kanada</text>
                </answer>
                <answer correctness="0">
                    <text>USA</text>
                </answer>
                <answer correctness="100">
                    <text>Russland</text>
                    <explanation>
                        Ganzer Norden Asiens.
                    </explanation>
                </answer>
                <answer correctness="0">
                    <text>China</text>
                </answer>
            </answers>
            <attachment type="local">
                <![CDATA[000000000000000000000000000000000]]>
            </attachment>
        </question>
    </questions>
</quizzenger-question-export>
```

5.5.3. Spezifikation des XML-Formates

Knoten	Beschreibung
`root`	Der Wurzelknoten `quizzenger-question-export` beinhaltet die gesamten Informationen des Dokumentes, wobei das Attribut `version` die für den Import relevate Version des Dokumentes angibt. Dies erlaubt die Unterstützung für das Laden verschiedener Dateiversionen und gewährleistet so die Rückwärtskompatibilität zu älteren Formaten.
`meta`	Der Knoten `meta` enthält Metainformationen zum Ursprungssystem und zum Export. Der Unterknoten `system` verweist mittels URL auf die Herkunft der Fragen (`APP\_PATH`-Konfiguration); `date` gibt an, zu welchem Zeitpunkt die Fragen exportiert wurden.
`questions`	Enthält die Auflistung aller Fragen.
`question`	Alle Informationen in Knoten vom Typ `question` beziehen sich immer nur auf eine einzelne Frage, wobei jede Frage komplett eigenständig ist und alle Angaben beinhaltet.
`author`	Name des Erstellers der Frage.
`created`	Datum, an dem die Frage erstellt wurde.
`modified`	Datum der letzten Änderung.
`category`	Die Attribute `first`, `second` und `third` geben die Stufen der Kategorie an, welchen die Frage untergeordnet ist.
`text`	Der eigentliche Text der Frage; kann mittels Markdown formatiert sein oder als Plain Text vorliegen.
`answers`	Beinhaltet alle definierten Antworten zu der jeweiligen Frage.
`answer`	Stellt eine einzelne Antwort da. Der Unterknoten `text` gib dabei den Text der Antwort an. Zusätzlich kann der optionale Unterknoten `explanation` definiert werden, welche zusätzliche Informationen zur Antwort liefert.
`attachment`	In diesem Knoten wird der Fragen-Anhang gespeichert, wobei das Attribut `type` den Typ des Anhanges spezifiziert. In Quizzenger sind momentan zwei Typen spezifiziert. Handelt es sich um eine externe Ressource, dann ist der Typ `‘‘url"` und der Knoteninhalt entspricht der URL zu dieser Ressource. Falls es sich jedoch um eine interne, d.h. vom Benutzer hochgeladene Ressource handelt, so ist der Typ `‘‘local"` und der Wert des Knotens ist die mit Base64 kodierte Binärdatei, verschachtelt in einer `CDATA`-Sektion.

5.5.4. Komprimierung

Bei einer grossen Anzahl Fragen oder bei der Verwendung vieler Anhänge ist es sehr wahrscheinlich, dass die XML-Datei stark aufgebläht wird. Aus diesem Grund werden die Exportdateien mittels dem GZIP-Algorithmus direkt vom Server komprimiert. Es handelt sich also bei `*.quizzenger`-Dateien eigentlich um `*.xml.gz`-Dateien. Bei internen Tests wurde dabei eine durchschnittliche Reduktion auf unter 20% des ursprünglichen Speicherverbrauches erreicht.

5.6. Multimediale Erweiterung der Fragetypen

5.6.1. Formatierung mit Markdown

Fragen können neu mittels eines Markdown-Dialektes formatiert werden. Bei Markdown handelt es sich um eine sehr einfache Auszeichnungssprache für Texte. Beispielsweise können Wörter ganz einfach `**fett**` oder `*kursiv*` geschrieben werden. Durch den Tag `[attachment]` werden Anhänge wie Bilder oder Videos direkt an dieser Stelle eingebettet. Markdown erlaubt eine Vielzahl weiterer Formatierungen, die wichtigsten werden dabei in der Markdown-Guideline beschrieben, welche ebenfalls mit dieser Arbeit eingereicht wurde.

5.6.2. Fragen-Anhänge

Durch die Erweiterung um das Hinzufügen von Anhängen zu Fragen wird die Möglichkeit geschaffen, multimediale Inhalte in die Fragestellung miteinzubinden. Dazu gehört das direkte hochladen oder verlinken von Bilddateien sowie das Einbetten externer Ressourcen wie Videos von diversen Portalen.

Das bestehende System setzt bereits auf eine Validierung der Daten vor dem Abschicken eines Formulars. Dies ist sinnvoll aus Sicht der Usability und soll auch von der neuen Funktion unterstützt werden. Die Schwierigkeit bestand nun darin, vor dem Abschicken des Formulars zu wissen, ob der Anhang korrekt hochgeladen wurde. Ansonsten besteht die Gefahr, dass ein Benutzer eine Frage formuliert, diese speichern will und dann erfährt, dass er die Frage nochmals erfassen muss, weil beim Hochladen des Bildes ein Fehler aufgetreten ist. Um dies zu verhindern wurde ein zusätzliches Formular in Form eines Popup-Fensters erstellt, welches das Bild bereits zum Server in ein temporäres Verzeichnis hochlädt, bevor die gesamte Frage abgeschickt wird.

Die momentan unterstützten externen Multimediaformate sind Filme von YouTube[6] und Vimeo[7] sowie Bilder in allen gängigen, von Browsern darstellbaren Formaten. Die Anhänge lassen mit der oben erwähnten Markdown-Sprache einbinden. Der im Text zu verwendende Tag zur Einbindung lautet `[attachment]`.

[6] `https://www.youtube.com/`
[7] `https://www.vimeo.com/`

6. Nichtfunktionale Anforderungen

Die nachfolgenden nichtfunktionalen Anforderungen wurden an diese Semesterarbeit gestellt, hier sinngemäss übernommen aus der ursprünglichen Aufgabenstellung:

1. Die Anwendung soll auf HTML5 basieren und der Verwaltung von Fragen dienen. Zusätzlich soll sie als Web-App auf Smartphones zur Beantwortung von Fragen einsetzbar sein.
2. Das Design soll *responsive* mittels Bootstrap[8] gestaltet werden.
3. Die Studienarbeit adressiert die Scalability-Requirements im Hinblick auf für crowdbasierte Systeme übliche grosse Datenmengen. Es werden auch bei diesem Projekt Performance-Tests durchgeführt. Der Rahmen der zu behandelnden Datenmengen entspricht dem der vorhergehenden Arbeit, wobei eine zusätzliche Skalierbarkeitsanalyse erarbeitet werden soll.
4. Der neue Code dieser Studienarbeit wird mit Rücksicht auf möglichst grosse Flexibilität und Erweiterbarkeit entwickelt.
5. Informationen sollen dynamisch geladen werden und gefiltert werden können, ohne dass dabei die Seite komplett neu aufgebaut werden muss (übernahme des bisherigen Systems).
6. Ease-of-Use ist ein kritisches Requirement für ein crowdsourced System. Neue Benutzer sollen sich intuitiv zurechtfinden mit dem System und dessen Handhabung (GUI & Konzept).
7. Momentan ist das System ausschliesslich deutschsprachig vorgesehen.

6.1. Performance Testing

Auch während dem Projekt *Crowdsourced Quizzes 2* wurden Performance-Tests durchgeführt. Besonders genau wurde dabei die Datenbank untersucht, da diese das Rückgrat der Anwendung darstellt. Der Fokus lag dabei auf den Fragen und den dazugehörenden Abfragen.

6.1.1. Vorbereitungen

Als erstes wurde die Datenbank für die bevorstehenden Tests künstlich "aufgebläht". Das heisst, es wurden mittels eines Skriptes 50'000 Fragen inkl. Antworten angelegt (Datenbankgrösse: ca. 250 MB). Dabei wurden der Fragetext und die Texte der Antworten zufällig generiert, so dass jede Frage einzigartig ist. Die Einzigartigkeit ist wichtig um Hintergrundoptimierungen des Datenbanksystems zu verhindern, damit die Resultate nicht verfälscht werden.

Zur Durchführung der Tests wurden zwei Skripte angelegt. Die Datei `bloat.php` füllt die Datenbank mit zufällig generierten Fragen und die Datei `perf.php` führt

[8] `http://getbootstrap.com/`

eine Reihe von Abfragen durch um das Laufzeitverhalten unter Last zu messen.

Die Tests wurden auf einem Virtual Server der HSR durchgeführt. Dabei muss erwähnt werden, dass dieser lediglich 15 GB Festplattenspeicher und nur 1 GB Arbeitsspeicher zur Verfügung stellt. Die Hardware liegt damit am absolut unteren Ende der Anforderungen für ein grösseres Crowd-Projekt.

6.1.2. Ausführungsgeschwindigkeit

Die Performance der Queries lag stets im unteren Millisekundenbereich. Eine interessante Entdeckung dabei war, dass die Grösse der Datenbank keinen bemerkbare Fluktuation der Ausführungsgeschwindigkeit bewirkte. Dies lässt sich dadurch erklären, dass die meisten Abfragen relativ einfach aufgebaut sind und nur kleine Result-Sets ohne komplexe Abhängigkeiten liefern (z.B. eine einzelne Frage mit ihren Antworten). An dieser Stelle sei auf Abschnitt 6.2.3 verwiesen, wo Performance-Probleme beim Abfragen von Listen behandelt werden.

Für die Messungen wurde die Datei `perf.php` eingesetzt. Individuelle Queries wurden dabei ebenfalls manuell getestet und führten zu den selben Ergebnissen.

6.1.3. Speicher- und CPU-Auslastung

Die Auslastung des Arbeitsspeichers sowie des Prozessors scheint bei einer regulären Verwendung absolut bedenkenlos zu sein. Dies liegt daran, dass sämtliche Queries sehr schnell ausgeführt werden und somit zu keinen langen Blockaden führen. Der Speicherverbrauch ist Aufgrund der kleinen Datenmengen einzelner Benutzer ebenfalls unproblematisch und kann ggf. hardwareseitig verbessert werden.

Probleme traten jedoch bei den Tests mit der Datei `perf.php` auf. Das Skript lädt sehr viele Daten auf einmal und da PHP/MySQL nicht automatisch “Cursors” zur Abfrage verwendet, muss entsprechend viel Speicher zur Verfügung stehen (ca. 230 MB bei 50’000 Fragen) um die Daten halten zu können. Die CPU-Auslastung ist dabei auf 100% angestiegen. Diese Situation tritt jedoch nicht im Normalbetrieb auf, da jeder Benutzer nur sehr beschränkt Zugriff auf grössere Datenmengen hat. Auch hier sei jedoch nochmal auf die Schwachstelle der Listen-Abfragen verwiesen, welche in Abschnitt 6.2.3 beschrieben wird.

6.1.4. Vergleich mit Quizzenger 1

Leider ist es nur bedingt möglich, die Performances von *Quizzenger 2* und *Quizzenger 1* miteinander zu vergleichen. Die Tests der vorherigen Gruppe sind leider in keiner Weise nachvollziehbar und somit auch nicht erneut durchführbar. Aufgrund der Screenshots aus dem technischen Bericht von *Quizzenger 1* gehen wir davon aus, dass die Performance in etwa in einem ähnlichen Rahmen liegt.

Die genannten Performance-Schlussfolgerungen aus dem Bericht von *Quizzenger 1* treffen grundsätzlich auch für dieses Projekt zu:

1. Die Grösse der Tabellen bzw. der Datenbank wächst proportional mit der Anzahl Fragen.
2. Die Antwortzeiten der Abfragen sind auch hier vollkommen unbedenklich.
3. Die Arbeitsspeicherauslastung wächst geringfügig und kann auch hier mit stärkerer Hardware kompensiert werden.

Der Performance-Level konnte also gehalten werden. Neu erstellte Queries weisen dieselben Eigenschaften auf wie die alten, sind also kurz und kompakt und datenbanktechnisch nicht komplex genug, um Performance-Probleme zu erzeugen.

6.2. Skalierbarkeit

Während dieser Studienarbeit wurde die Skalierbarkeit des Systems analysiert. Dabei wurden positive, als auch negative Aspekte aufgedeckt, welche in diesem Abschnitt beschrieben werden.

6.2.1. Datenbank

Die Datenbank bildet das Herzstück der Anwendung und ist sehr flexibel und skalierbar. Bei der Erstellung der Datenbank wurde von Anfang an Wert auf die grundlegende Struktur und deren Normalisierung gelegt. Die zahlreichen Erweiterungen, welche an der Datenbank zur Implementierung der neuen Funktionen durchgeführt wurden, bestätigen deren Flexibilität und Erweiterbarkeit. Aufgrund der Tests scheint die Datenbank als Ursache von Performance-Probleme ausgeschlossen werden zu können.

Allfällige Verbesserungen können hier von Datenbankexperten am Datenbanksystem selbst durch“Tweaking” der Einstellungen vorgenommen werden.

6.2.2. Gamification

Auch die Bestandteile der Gamification folgen den Grundsätzen der Erweiterbarkeit und Skalierbarkeit. Das neu implementierte Dispatching-System beispielsweise ist nahezu eigenständig und hat deshalb sehr wenige Abhängigkeiten zu externen Klassen. Durch die Einführung von Plugins kann neue Funktionalität sehr einfach hinzugefügt werden (siehe Abschnitt 7.1).

6.2.3. Schwachstellen

Die Skalierbarkeit des Systems ist zwar grundsätzlich gegeben, jedoch besteht noch eine schwerwiegende Schwachstelle aus *Quizzenger 1*, welche nicht ohne Aufwendiges Refactoring behoben werden kann.

Bei dieser Schwachstelle handelt es sich um ein Problem mit der Abfrage von grösseren Datenmengen in Kombination mit den Data-Tables, welche sehr oft verwendet werden (siehe dazu Abschnitt 5.10.2 des technischen Berichts von *Quizzenger 1*).

Die Schwachstelle machte sich erst bei der Durchführung der Performance-Tests bemerkbar, weshalb sie aus Zeitgründen nicht mehr behoben werden konnte.

Die von *Quizzenger 1* angepriesene Paginierung von Listen bildet eine Schlüsselfunktion, da Tabellen an vielen Orten auf der Seite eingesetzt werden. Beispielsweise unter "Fragepool", "Mein Profil" sowie an weiteren Stellen. Die Paginierung funktioniert grundsätzlich einwandfrei, hat jedoch einen schwerwiegenden Haken: *paginiert wird ausschliesslich auf der Seite des Clients. Es werden also sämtliche zu paginierenden Daten abgefragt und übertragen.*

Während dem Performance-Testing wurde festgestellt, dass das System ab einer bestimmten Anzahl Fragen *unbenutzbar* wird. Existieren also 50'000 Fragen auf dem System, so werden alle Fragen beim Aufruf des Question-Pools übertragen.

Vorschlag zur Behebung: Durch die Verwendung der SQL-Befehle `LIMIT` und `OFFSET` kann die Datenmenge begrenzt und aufgeteilt werden. Bei der eigentlichen Paginierung müssen also laufend neue Daten via AJAX-Request der Datenbank entnommen werden. Somit ist sichergestellt, dass das blosse Ansehen der Frageliste oder des Reportings nicht zu einer Überlastung des Servers führt.

6.3. Wartbarkeit und Flexibilität

Wie in Abschnitt 8 über Refactoring ausführlich beschrieben wird, wurde das Projekt schwer wartbar übernommen. Um künftigen Studenten nachfolgender Studienarbeiten Unannehmlichkeiten bei der Einarbeitung zu ersparen, wurde ein umfangreiches Refactoring durchgeführt, wodurch Wartbarkeit und Flexibilität grundlegend verbessert wurden.

Neuer Code, insbesondere das Dispatching-System (siehe Abschnitt 7.1) wurden sehr abstrakt definiert, was eine maximale Erweiterbarkeit garantiert. Das Konzept zum Erweitern des Systems mit neuen Controllers, Models und Templates konnte vom bestehenden Projekt übernommen werden, wurde jedoch ebenfalls vereinfacht und übersichtlicher gestaltet.

6.4. Usability Testing

Usability-Tests sind unerlässlich, wenn es darum geht, der Anforderung eines intuitiven Designs gerecht zu werden. Im Verlauf dieser Studienarbeit wurden drei solche Tests mit Testpersonen ganz unterschiedlicher Hintergründe durchgeführt.

Diejenige Person, welche im Alltag am meisten mit Computern arbeitet konnte dabei die Tests am besten meistern. Bei den Überlegungen über Usability muss das Zielpublikum einer Anwendung klar sein. Da Quizzenger primär für eine technische Hochschule entwickelt wird, darf der Anspruch an die Benutzer tendentiell höher sein als beispielsweise bei einer Anwendung wie "Whatsapp".

Nachfolgende Auflistung zeigt die Probleme und Unschönheiten, welche während dieser Arbeit behoben wurden:

1. Der Benutzername wird jetzt angemeldeten Benutzern angezeigt.
2. Weitere Tooltips wurden hinzugefügt um Funktionalitäten zu erklären (u.a. bei Schaltflechen zur Löschung und Bearbeitung).
3. Die Lernen-Ansicht wurde in "Quiz/Game" umbenannt und die Ansicht "Quiz-Generieren" sowie der Suchmodus wurden klarer beschrieben. Dies verbessert die allgemeine Benutzerführung.
4. Die Kommentar/Bewertungs-Funktion war keiner der Testpersonen klar. Ein Kommentar bzw. eine Bewertung wurde über AJAX erfasst. Die einzige Rückmeldung an den Benutzer bestand darin, dass die Schaltfläche "Bewertung abschicken" nach Betätigung "Bewertung abgeschickt" anzeigte. Dieses Problem wurde durch eine Aktualisierung der Seite behoben. Eine Bewertung wird also direkt aufgenommen und angezeigt.

Aufgrund unzureichender Zeit für die Behebung sämtlicher Probleme müssen u.a. folgende offenen Punkte zukünftig behandelt werden:

- Der Vorgang zur Erstellung eines Quizzes sollte vereinfacht werden. Bei den Usability-Tests sind folgende Fragen aufgetreten:
  - Wieso wird ein generiertes Quiz nicht automatisch gespeichert?
  - Wie füge ich zu einem bestehenden Quiz neue Fragen hinzu?
- Ausserdem war der Unterschied zwischen einem Quiz und einem Game unklar.

Nach diesen Tests erscheint es sinnvoll, künftig eine Serie von Video-Tutorials zu erstellen. Diese dienen als Kurzeinführung in das System und sollten die Bedienung grundlegender Funktionen in jeweils 60 Sekunden erläutern. Dadurch könnten einige Unklarheiten im Voraus beseitigt werden.

6.5. Code-Dokumentation (Doxygen)

Doxygen ermöglicht das Erzeugen einer Entwickler-Dokumentation basierend auf PHPDoc-Kommentaren innerhalb der Code-Dateien. Um die Dokumentation zu erstellen bzw. zu aktualisieren muss lediglich der Befehl `doxygen doxygen` im entsprechenden Verzeichnis aufgerufen werden. Die generierte Dokumentation wird anschliessend im Ordner `doc/` abgelegt.

Weitere Informationen zur Anwendung und Konfiguration sind unter `http://www.doxygen.org/` verfügbar. Auf der Seite `http://www.stack.nl/~dimitri/doxygen/manual/docblocks.html` wird die grundlegende Syntax von PHPDoc-Kommentaren erklärt und veranschaulicht.

7. Architektur

7.1. Das Event- & Dispatching-System

Die im Rahmen dieser Studienarbeit zu erfüllenden Anforderungen konnten mit dem bestehenden System zur Punktevergabe nicht umgesetzt werden, da dieses viel zu starr und unflexibel war. So wurde beispielsweise nicht zwischen verschiedenen Aktions-Typen unterschieden und Punkte wurden lediglich für das korrekte Beantworten von Fragen vergeben. Um den Grundanforderungen Modularität und Erweiterbarkeit von Quizzenger 2 nachzukommen, wurde das Dispatching-System bewusst dynamisch und selbstregulierend entworfen und entwickelt. In diesem Abschnitt werden die Details des Systems erläutert.

7.1.1. Die Aufteilung in zwei Dispatcher

Das Event-System basiert auf dem Auslösen und Verarbeiten von Events durch einen zentralen Event-Controller. Der Event-Controller aggregiert dabei alle Events und verteilt diese weiter an die definierten Dispatcher. Zur Umsetzung der Anforderungen wurden für Quizzenger 2 zwei Dispatcher benötigt, der `AchievementDispatcher` und der `ScoreDispatcher`.

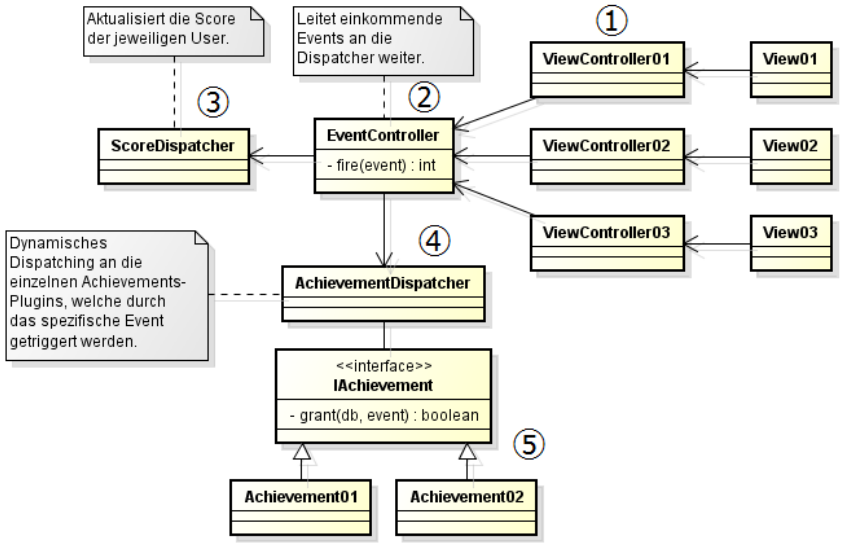


Abbildung 8: Dispatching-System

In Abbildung 8 werden die Struktur und die Interaktionen zwischen den Klassen grob dargestellt. Die ersichtlichen fünf Punkte sind dabei von besonderer Bedeutung und werden nachfolgend erläutert.

① Quizzenger wendet grundsätzlich das MVC-Konzept an. So ist jeder View ein Controller zugeordnet. Dieser erzeugt Events basierend auf den Aktionen des Benutzers und übergibt diese dem System. Beispielsweise wird der Event-Controller von einem View-Controller darüber informiert, wenn der Benutzer eine Frage beantwortet oder ein Spiel abgeschlossen hat.

② Nachdem ein Event von einem Controller ausgelöst wurde, wird dieses durch den Event-Controller an alle Dispatcher zur Verarbeitung weiterverteilt.

③ Die Aufgabe des Score-Dispatchers ist es, basierend auf den jeweiligen Events die korrekte Anzahl Punkte den Benutzern gutzuschreiben. Die benötigten Informationen dazu werden bei der Systeminstallation in die Datenbank übertragen.

④ Der Achievement-Dispatcher behandelt ebenfalls sämtliche Events und teilt Achievements unter den Benutzern aus, falls diese die Bedingungen der jeweiligen Achievements erfüllt haben (siehe ⑤).

⑤ Aufgrund der dynamischen und erweiterbaren Natur des Achievement-Systems werden die einzelnen Achievements über Plugins definiert. Dabei handelt es sich um eigenständige Klassen, welche das Interface `IAchievement` implementieren. Das Interface definiert dabei eine Methode `grant` mit Rückgabetyp `boolean`, welche bestimmt, ob die Bedingungen des geprüften Achievements erfüllt wurden. Der Aufbau, die Funktionsweise und die Implementierung der Achievement-Plugins werden in Abschnitt 7.1.2 genauer beschrieben.

7.1.2. Achievement-Plugins

Die Erstellung der Achievement-Plugins ist einfach und unkompliziert. Dies war ebenfalls eine bewusste Design-Entscheidung, um es Administratoren zu ermöglichen, eigene Achievements und dazugehörige Plugins zu definieren.

Bei den Plugins handelt es sich um eigenständige Klassen. Diese müssen im Ordner `./plugins/achievements/` angelegt werden und das Interface `IAchievement` implementieren, welches eine methode `grant` definiert mit folgender Signatur.

```
grant(database : SqlHelper, event : UserEvent) : boolean
```

Der erste Parameter verweist auf eine aktive Datenbankverbindung, der zweite Parameter spezifiziert das aktuelle Event, welches geprüft werden soll. Die Aufgabe der Methode ist es zu bestimmen, ob der im Event enthaltene User die Bedingungen des Achievements erfüllt hat. Ist dies der Fall und gibt die Methode `true` zurück, so wird dem Benutzer das Achievement erteilt, andernfalls erhält er es nicht.

Zur Veranschaulichung dient das nachfolgende Plugin, welches bestimmt, ob der Benutzer die Bestimmte Anzahl an Fragen korrekt beantwortet hat. Die genauen Einstellungen sind in der Datei `settings.xml` zu treffen.

```
1  namespace quizzenger\plugins\achievements {
2      use \SqlHelper as SqlHelper;
3      use \quizzenger\dispatching\UserEvent as UserEvent;
4      use \quizzenger\achievements\IAchievement as IAchievement;
5
6      class QuestionAnsweredCorrectAchievement
7          implements IAchievement
8      {
9          public function grant(SqlHelper $database,
10             UserEvent $event)
11         {
12             $database = $database->database();
13             $userId = $event->user();
14             $questionCount = $event->get('question-count');
15
16             $statement = $database->prepare('SELECT COUNT(*) AS
                   count FROM questionperformance WHERE user_id=?
                  AND questionCorrect>0');
17
18             $statement->bind_param('i', $userId);
19             $statement->execute();
20             return $statement->get_result()->fetch_object()
21                 ->count >= $questionCount;
22         }
23     } // class QuestionAnsweredCorrectAchievement
24 } // namespace quizzenger\plugins\achievements
```

Detaillierte Informationen zur Implementierung von Achievements können der mit Doxygen generierten Code-Dokumentation (Abschnitt 6.5) entnommen werden.

7.2. Das Datenbankschema

Das unten aufgeführte Diagramm zeigt die komplette Datenbankstruktur. Zur besseren Lesbarkeit sei hier auf die zu diesem Report mitgelieferten Dateien verwiesen.

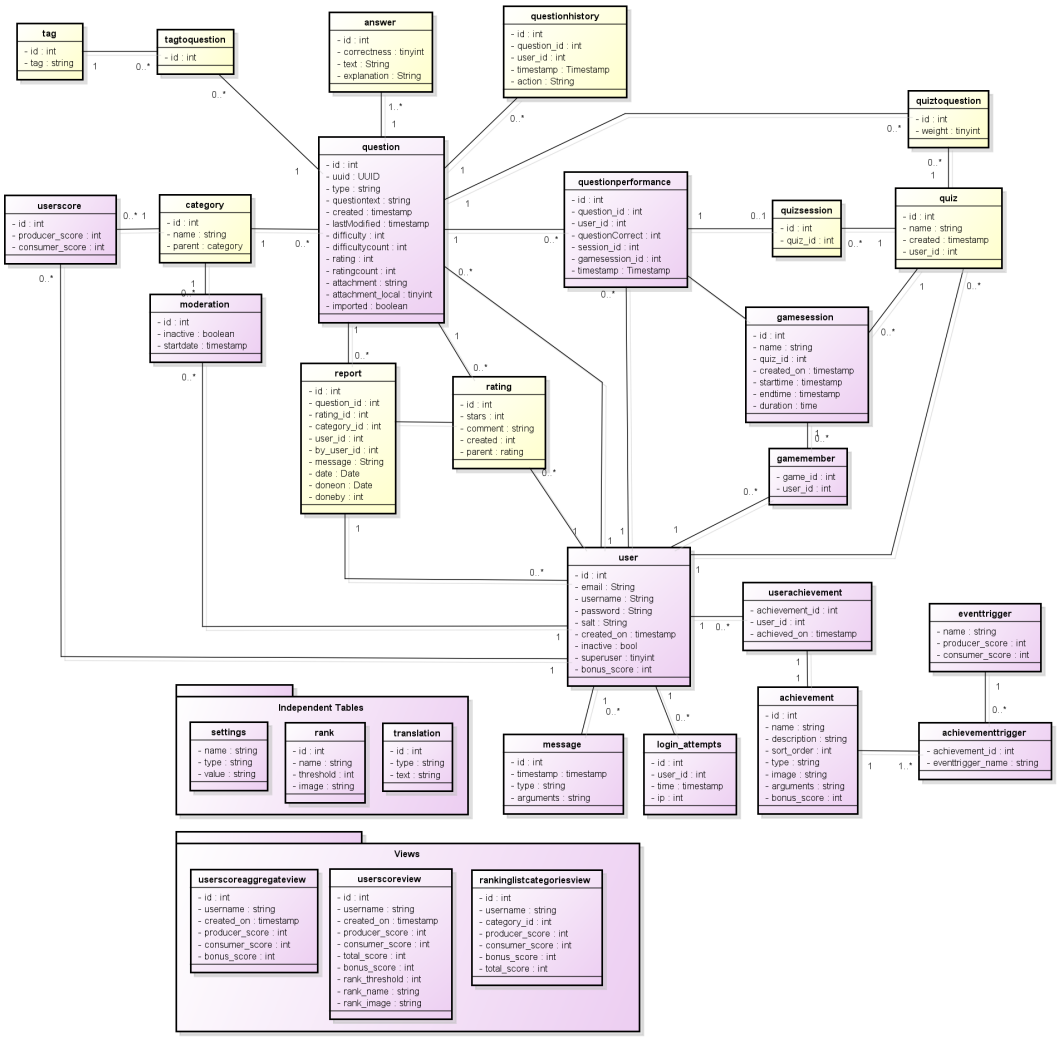


Abbildung 9: Datenbankschema

Das Datenbankschema wurde wo nötig erweitert. Alle Tabellen, welche erweitert oder neu erstellt wurden, sind rötlich markiert. Die Funktion jeder Tabelle, sowie jedes nicht selbsterklärende Feld, werden nachfolgend kurz erläutert.

achievement Enthält alle Achievements, welche vor der Installation in der Datei `settings.xml` spezifiziert wurden. Die Achievements werden unter der Ansicht "Mein Profil" angezeigt.

achievementtrigger Verknüpft die Tabellen `achievement` und `eventtrigger`.

answer Enthält sämtliche Antworten aller Fragen. Das Feld `correctness:TINYINT` repräsentiert dabei den Korrektheitsgrad einer Frage und ist eine Zahl zwischen 0 (komplett falsch) und 100 (absolut richtig).

category Speichert die einzelnen Kategorien in einer Baumstruktur, welche durch Referenzierung des Parent-Nodes ausgedrückt wird. Falls eine Kategorie keinen Parentnode eingetragen hat (0), handelt es sich dabei um eine Hauptkategorie. Leaf-Nodes besitzen keine weiteren Unterkategorien.

eventtrigger Enthält die möglichen Ereignisse (Events), die von Benutzern ausgelöst werden können. Bei gewissen Events können direkt `producer_score` oder `consumer_score` gesetzt werden (z.B. für das Erstellen einer Frage). Die Punktzahl wird von der Klasse `ScoreDispatcher` automatisch verteilt.

gamemember Verknüpft die Tabellen `gamesession` und `user`. Sie gibt darüber Auskunft, welche Benutzer bei welchen Games teilnehmen.

gamesession Ist das Pendant zur Tabelle `quizsession` und bildet ein Game mit einer klar definierten `starttime`, `endtime` und `duratioon`. Der Game-Host ist immer derjenige Benutzer, der das zugrundeliegende Quiz erstellt hat.

login_attempts Enthält Informationen zu fehlgeschlagenen Loginversuchen. Dadurch können Bruteforce-Attacken vermindert werden. Diese Tabelle wurde um die Spalte `ip` erweitert.

message Enthält Nachrichten, welche einem Benutzer beim nächsten Request angezeigt werden. Diese Tabelle wurde erforderlich, weil Benutzer potentiell Benachrichtigungen für andere Benutzer auslösen können (z.B. “Game End”). Zusätzlich kann so sichergestellt werden, dass keine Nachrichten verlorengehen, falls der Benutzer nicht mehr angemeldet ist (siehe Abschnitt 5.1.5).

moderation Gibt Auskunft darüber, welche Benutzer Moderationsrechte haben. Moderationsrechte werden immer pro Kategorie und nur bei Erreichen einer bestimmten Punktzahl vergeben, welche in der Datei `config.php` definiert wird. Das Feld `inactive:TINYINT` zeigt an, ob einem Benutzer die Moderationsrechte entzogen (1) wurden oder ob diese normal aktiv sind (0). Die Tabelle wurde um das Feld `startdate` erweitert.

question Enthält die vom Benutzer erstellten Fragen. Um die Anzahl Abfragen auf die Datenbank zu minimieren werden die Bewertungen sowie die ermittelten Schwierigkeitsgrade im Record direkt in aggregierter Form gespeichert.

`type:STRING` beschreibt den Fragetyp. Im Rahmen dieser Studienarbeit wurde der Fragentyp `SingleChoice` implementiert.

`difficulty:DOUBLE` bezeichnet den Schwierigkeitsgrad der Frage. Der Wert 100 deutet auf eine einfache, der Wert 0 auf eine schwere Frage hin.

`difficultycount:INT` enthält die Anzahl der in `difficulty` aggregierten einzelnen Werten.

`rating:DOUBLE` bezeichnet eine Zahl zwischen 0-5, welche die Bewertung einer Frage beschreibt, wobei der Wert 0 "schlecht" und der Wert 5 "sehr gut" bedeutet.

`ratingcount:INT` enthält die Anzahl der in `rating` aggregierten einzelnen Werten.

Die Tabelle `question` wurde um folgende Spalten erweitert:

`attachment_local:TINYINT` gibt an, ob es sich um ein lokales Attachment (1) oder um eine URL (0) handelt.

`attachment:STRING` beinhaltet das eigentliche Attachment. Handelt es sich um ein lokales Attachment, so enthält dieses Feld eine Dateiendung. Die Datei im Attachment-Ordner besitzt den Dateinamen `question_id].[attachment]`. Andernfalls enthält dieses Feld eine URL.

`imported:TINYINT` gibt Auskunft darüber, ob die Frage mittels der Importfunktion importiert wurde.

`uuid:UUID` bezeichnet die eindeutige ID einer Frage über die lokale Datenbank hinweg. Dadurch kann das doppelte Importieren derselben Frage vermieden werden.

questionhistory Enthält Informationen zur Änderungsgeschichte von Fragen. Dies ist sinnvoll, 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 von `correctness` der gewählten `answer` gespeichert. Anhand dieser Daten werden Statistiken Bewertungen generiert.

`session_id:INT` gibt an, ob die Frage im Kontext eines *Quizzes* beantwortet werden kann. Das Feld ist entweder `NULL` oder die ID einer `quizsession`.

`gamesession_id:INT` gibt an, ob die Frage im Kontext eines *Games* beantwortet werden kann. Das Feld ist entweder `NULL` oder die ID der `gamesession`.

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 temporären Quizzes zu markieren.

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

quiztoquestion Dient der Zuordnung von Fragen zu Quizzes.

`weight:TINYINT` ermöglicht das Gewichten einer Frage im Kontext eines Quizzes. Dies ermöglicht die Vergabe einer unterschiedlichen Anzahl Punkten basierend auf der Schwierigkeit der Frage.

rank Enthält alle Ränge, welche vor der Installation in der Datei `settings.xml` spezifiziert wurden.

rankinglistallcategoriesview Dies ist eine Hilfs-View, die für die Abfrage der Ranglisten für alle Kategorien benötigt wird. Sie wurde erstellt, da MySQL keine `WITH`-Queries kennt und temporäre Tabellen aufgrund ihrer Restriktionen für diese Abfrage ungeeignet sind.

rating Ermöglicht die Bewertung einer Frage sowie deren Diskussion.

`parent:INT` bezeichnet die Bedeutung des Eintrages. Falls der Wert 0 gesetzt ist, dann bezieht sich der Eintrag direkt auf eine Frage und ist somit eine Bewertung. Bei Werten ungleich 0 ist der Eintrag ein Kommentar zu einer Bewertung.

report Enthält Informationen zu von Benutzern gemeldenten Inhalten. Dabei kann ein Report durch einen Benutzer als erledigt markiert werden und wird danach nicht mehr angezeigt.

settings Stellt eine Alternative zur Config-Datei dar und kann insbesondere für Werte verwendet werden, welche direkt in Queries benötigt werden.

tag Dient der feineren Kategorisierung einzelner Fragen. Die Tags hätten auch als Zeichenkette in der Frage selbst gespeichert werden können. Dies würde die Abfragen vereinfachen, allerdings würde dadurch die Normalform verletzt und das Suchen verlangsamt werden.

tagtoquestion Enthält Informationen zur Verlinkung von einem oder mehreren Tags einer Frage.

translation Enthält die Übersetzungen zu einzelnen Message-Types.

user Enthält die Benutzerinformationen.

`inactive:TINYINT` gibt an, ob der Benutzer gesperrt ist. Ist dies der Fall (Wert 1), so erhält der Benutzer eine entsprechende Meldung beim Versuch sich anzumelden.

`superuser:TINYINT` gibt an, ob der Benutzer ein Super User ist. Bei Wert 1 hat der Benutzer Zugriff auf diverse mächtige Tools zur Verwaltung und Moderation.

userachievement Verknüpft die Tabellen `achievement` und `user`. Dabei wird zusätzlich Auskunft darüber gegeben, wann ein Benutzer ein bestimmtes Achievement erworben hat.

userscore Enthält die gesammelten Punkten eines Benutzers pro Kategorie.

`producer_score:INT` ist das Total der Punktzahl, welche der Benutzer durch Beiträge für die Community erarbeitet hat.

`consumer_score:INT` ist das Total der Punktzahl, welche der Benutzer beim Benutzen der Inhalte (z.B. durch Fragen beantworten) erworben hat.

userscoreaggregateview Wird von der View `userscoreview` benötigt.

userscoreview Ist eine Hilfs-View, welche den globalen Rang und die totale Punktzahl jedes Benutzers aggregiert.

7.3. Frameworks & Libraries

An dieser Stelle seien nochmals die Frameworks & Libraries erwähnt, die bereits in *Quizzenger 1* eingesetzt wurden. Genauere Details können im technischen Bericht von *Quizzenger 1* in Abschnitt 5.10 nachgelesen werden.

7.3.1. Quizzenger 1: Fremdcode Deklarierung

- CSS
  `/bootstrap-theme.min.css` (Bootstrap)
  `/bootstrap.css` (BootStrap)
  `/bootstrap.min.css` Bootstrap)
- DataTables (DataTables)
- Fonts (Bootstrap)
- Includes
  `/includes/sqlhelper.php` (Stack Overflow, Public Domain)
- JavaScript
  `/js bootstrap.min.js` (Bootstrap)
  `/jquery-1.11.1.min.js` (JQuery)

7.3.2. Quizzenger 2: Fremdcode Deklarierung

Um die neuen Anforderungen zu bewältigen, wurden zwei Frameworks eingesetzt:

markdown.js Das Framework markdown.js[9] ermöglicht die Formatierung von Text mittels eines vorgegebenen Markdown-Dialekts. Durch einen speziell implementierten Tags können Attachments eingebunden werden (siehe Abschnitt 5.6.2). Das Framework wird in `skeleton.php` eingebunden.

doT.js doT[10] ermöglicht es, JavaScript-Templates zu erstellen. Aufgrund von Vorkenntnissen bei der Verwendung dieses Frameworks war es die erste Wahl. Die Einbindung geschieht ebenfalls in der Datei `skeleton.php`.

[9] `https://github.com/evilstreak/markdown-js`
[10] `http://olado.github.io/doT/`

8. Refactoring

Wie jedes IT-Projekt unterliegt auch Quizzenger dem Wandel der Zeit. Durch neue Anforderungen und Features müssen bestehende Konzepte überdenkt und teilweise angepasst werden. Dieses Kapitel gibt einen Überblick über das durchgeführte Refactoring in der Studienarbeit *Crowdsourced Quizzes 2*.

8.1. Includes

Der Code war abgesehen von den Models, der View-Klasse und den beiden Front-Controllers prozedural programmiert. Statt der Partitionierung des Codes mittels Namespaces, Klassen und Funktionen, wurde Code auf mehrere Dateien aufgeteilt, die bei Bedarf eingebunden (`include`) wurden. Dadurch waren die Dateien nicht eigenständig, Abhängigkeiten wurden geerbt anstatt eingebunden zu werden. Vor allem bei den Controllern führte das zu unübersichtlichen, kaum dokumentierten und teilweise bis zu 50 Zeilen langen If-Else-Statements.

Diesen Mängeln wurde wie folgt Abhilfe geschaffen:

- Ein Autoloader wurde implementiert, der die benötigten Klassen bei Bedarf automatisch lädt. Somit werden Klassen ohne explizite Angabe von `require_once` in das Programm bei Verwendung ("lazy-loading") an einer zentralen Stelle eingebunden.
- Die Gliederung wurde überarbeitet. Öffentlich zugreifbare (public) Dateien wie Bilder, Stylesheets und JS-Skripte wurden gemeisam im Ordner `content` abgelegt. Der gesamte selbstgeschriebene Programmcode wurde in den Ordner `quizzenger` verschoben und somit logisch abgekapselt.
- Die einzelnen Controller-Skripts wurden in Namespaces und Klassen umstrukturiert. Dadurch wird eine zukünftige Einarbeitung vereinfacht, da nun jederzeit klar ist, woher eine Variable kommt. Mit dieser Massnahme wurden alle Includes aus der Programm-Logik entfernt.
- Einzelne global benötigte Funktionen wie `redirect()`, `formatNumber()` oder `checkLogin()` wurden in globale Utility-Klassen ausgelagert. Im Gegensatz zur vorherigen Include-Lösung ist jetzt sofort klar, wo diese Funktionen definiert wurden.
- Da im ganzen Code die Logger-Klasse benötigt wird, wurden die Logger-Funktionen static gemacht, wodurch sie von überall her aufgerufen werden können. Obwohl das Singleton-Pattern als Anti-Pattern in Verruf steht, ist es richtig eingesetzt sehr nützlich. Folgende Alternativen dazu hätten bestanden:
  - Include im globalen Namespace. Diese Lösung wurde durch dieses Refactoring gerade versucht abzuschaffen.
  - Logger-Instanz wird jeder Klasse über den Konstruktor übergeben.
  - Dependency-Injection.

Berücksichtigt man alle diese Möglichkeiten, so erscheint das Singleton-Pattern am sinnvollsten. Es sei hier verwiesen auf das .NET-Framework, welches häufig Singletons wie `Application` oder `Console` für Verwaltungs-Klassen zur Verfügung stellt.

- Für den Zugriff auf die Models wurde ebenfalls eine Singleton-Klasse mit Namen `ModelCollection` erstellt. Diese stellt eine Fascade dar und trennt den Business Layer (Models) vom Presentation Layer (Views, Controllers).

8.2. Sicherheit

Obwohl einige Sicherheitmassnahmen wie der Einsatz einer verschlüsselten Verbindung (HTTPS), saubere Login-Checks und Vorkehrungen zum Schutz vor SQL-Injection getroffen wurden, sind trotzdem noch einige, teils gravierende Sicherheitsmängel aufgedeckt worden.

8.2.1. XSS – Cross-Site-Scripting

Im technischen Bericht der Studienarbeit *Crowdsourced Quizzes 1* wurde zwar erwähnt, dass Parameter "escaped" werden; leider wurde dies nur beim Login-Vorgang durchgezogen. Dabei wurde gesagt, dass URL-Parameter, die eventuell ausgegeben oder weiterverarbeitet werden, durch eine PHP-interne Funktion "escaped" würden:

```
1 $error = filter_input(INPUT_GET, 'err',
2     $filter = FILTER_SANITIZE_SPECIAL_CHARS);
```

Allerdings wurden von Benutzern stammenden Daten an keiner anderen Stelle des Programmes behandelt. So war es beispielsweise möglich, JavaScript als Fragetext anzugeben. Das Skript wurde anschliessend beim Laden der Frage direkt im Browser des Benutzers ausgeführt. Dies stellt ein erhebliches Sicherheitsproblem dar und wurde umgehend behoben. Dazu wurde die Funktion `htmlspecialchars()` an den kritischen Stellen eingesetzt.

8.2.2. AJAX-Requests

Bei den Aktionen, die über normale HTTP-Requests ausgelöst werden, wurden in der Regel die Berechtigungen überprüft. Diese Überprüfung ging bei der Klasse `AjaxController` für sämtliche Requests vergessen. Dies ist insofern tragisch, da alle Lösch-Funktionen über AJAX-Requests gelöst wurden. Sprich mit dem folgenden JavaScript-Code hätten alle Fragen des Systems gelöscht werden können. Dazu hätte man nicht einmal eingeloggt sein müssen.

```
1 for(var i = 0; i < 10000; i++){
2     deleteQuestion(i);
3 }
```

Das Beheben dieser Sicherheitslücke war ziemlich aufwändig, da bekanntlich ein nachträgliches Überarbeiten der Security-Vorkehrungen nicht trivial ist. Beispielsweise durfte der Datensatz nicht einfach direkt aus der View gelöscht werden. Zuerst musste die Rückmeldung des Servers abgewartet werden. Dies forderte eine völlig neue Struktur der AJAX-Requests.

8.2.3. Öffentliche Log-Files

Bisher mussten die Log-Files öffentlich zugreifbar sein, damit sie vom Super User über den Browser angezeigt werden konnten. Dadurch konnte jede Person, die den Pfad zu einer Log-Datei wusste, diese einsehen, ohne dabei eingeloggt zu sein oder sonstige Berechtigungen zu besitzen.

Diese Sicherheitslücke wurde behoben, indem die Log-Files nur noch indirekt über einen "Log Viewer" allen berechtigten Personen (Super Users) angezeigt werden. Zusätzlich wurde eine `.htaccess`-Datei in das Standard-Log-Verzeichnis gelegt, die jeglichen direkten Zugriff verweigert.

8.3. Nachrichten

Die bisherige Lösung sah vor, dass alle Info- und Error-Nachrichten in der Config-Datei definiert und über die URL (z.B. `&info=mes_logout_success`) aufgerufen werden. Diese Vorgehensweise hat zwei Nachteile:

- Bei einem Reload (F5) der Seite wurden diese Nachrichten erneut angezeigt.
- Nachrichten konnten nur von demjenigen Benutzer ausgelöst werden, den den die Nachricht auch gerichtet ist.

Durch die Einführung des Achievement-Systems wurde es notwendig, dass Nachrichten zu einem späteren Zeitpunkt angezeigt werden können. Ein Beispiel wäre der Fall, dass ein Benutzer zwar ein Spiel gewinnt, aufgrund der langen Gamelaufzeit jedoch nicht bis zum Game-End eingeloggt ist. Damit ihm sein errungenes Achievement dennoch angezeigt wird, welches durch das Event `game-end` ausgelöst wird, mussten die Messages umstrukturiert werden. Neu werden Nachrichten in der Datenbank gespeichert, bis diese gelesen werden. Die neue Lösung hat zusätzlich den Vorteil, dass grundsätzlich alle Texte (Nachrichten, Anzeigetexte, usw.) in der Datenbank gehalten werden können. Dadurch wird es auch zukünftig möglich, Nachrichten in mehreren Sprachen anzuzeigen.

8.4. Einheitliches Design

Damit das Design des User-Interfaces einheitlicher aussieht und wie "aus einem Guss" daherkommt, werden alle Inhalte von Quizzenger in einem Panel dargestellt. Das Resultat ist im Vergleich in Abbildung 10 ersichtlich.

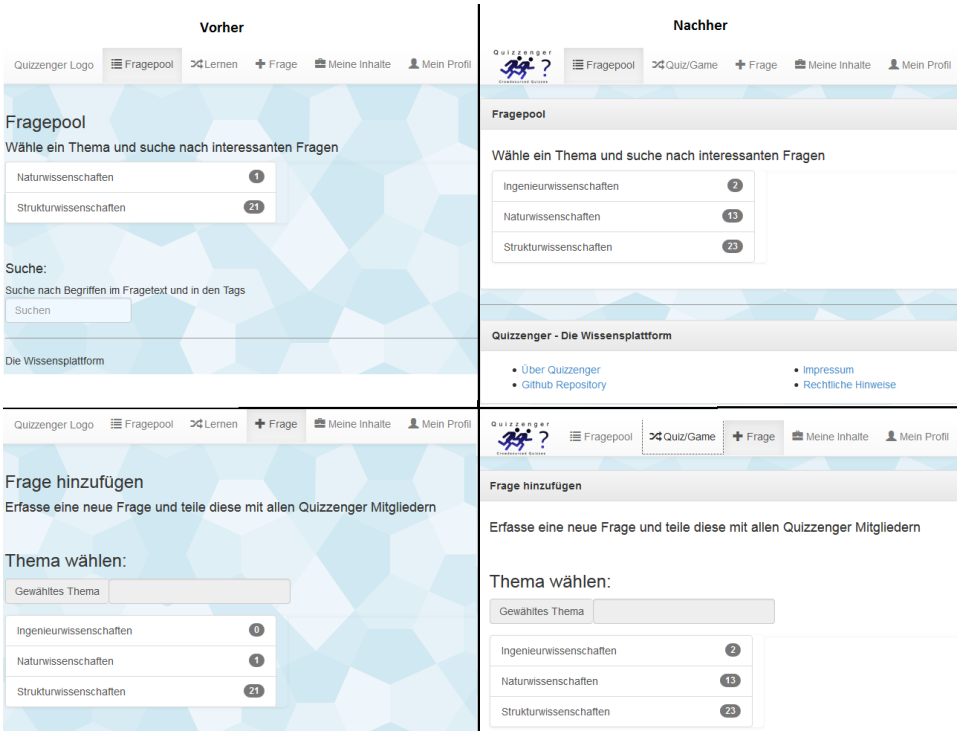


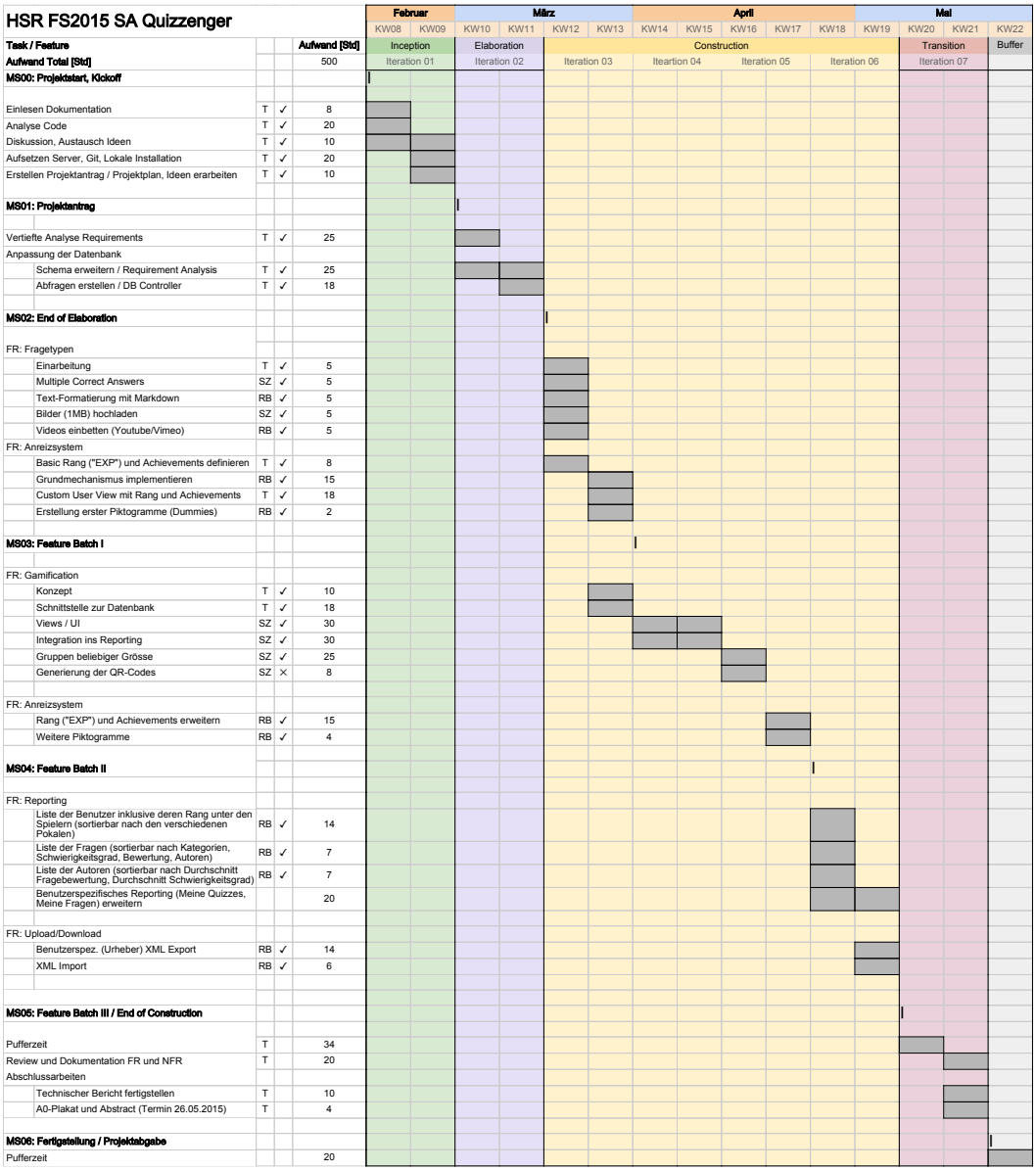
Abbildung 10: Refactoring User Interface

9. Project Management

Der folgende Abschnitt behandelt abschliessend das dieser Arbeit zugrundeliegende Projektmanagement und gibt Anhaltspunkte über die Vorgehensweise.

9.1. Zeitplan

Der zu Anfang erstellte Zeitplan sieht wie folgt aus:



HSR FS2015 SA Quizzenger				Februar		März				April					Mai			
				KW08	KW09	KW10	KW11	KW12	KW13	KW14	KW15	KW16	KW17	KW18	KW19	KW20	KW21	KW22
Task / Feature			**Aufwand [Std]**	Inception		Elaboration		Construction								Transition		Buffer
Aufwand Total [Std]			500	Iteration 01		Iteration 02		Iteration 03		Iteartion 04		Iteration 05		Iteration 06		Iteration 07		
MS00: Projektstart, Kickoff																		
Einlesen Dokumentation	T	✓	8															
Analyse Code	T	✓	20															
Diskussion, Austausch Ideen	T	✓	10															
Aufsetzen Server, Git, Lokale Installation	T	✓	20															
Erstellen Projektantrag / Projektplan, Ideen erarbeiten	T	✓	10															
MS01: Projektantrag																		
Vertiefte Analyse Requirements	T	✓	25															
Anpassung der Datenbank																		
Schema erweitern / Requirement Analysis	T	✓	25															
Abfragen erstellen / DB Controller	T	✓	18															
MS02: End of Elaboration																		
FR: Fragetypen																		
Einarbeitung	T	✓	5															
Multiple Correct Answers	SZ	✓	5															
Text-Formatierung mit Markdown	RB	✓	5															
Bilder (1MB) hochladen	SZ	✓	5															
Videos einbetten (Youtube/Vimeo)	RB	✓	5															
FR: Anreizsystem																		
Basic Rang ("EXP") und Achievements definieren	T	✓	8															
Grundmechanismus implementieren	RB	✓	15															
Custom User View mit Rang und Achievements	T	✓	18															
Erstellung erster Piktogramme (Dummies)	RB	✓	2															
MS03: Feature Batch I																		
FR: Gamification																		
Konzept	T	✓	10															
Schnittstelle zur Datenbank	T	✓	18															
Views / UI	SZ	✓	30															
Integration ins Reporting	SZ	✓	30															
Gruppen beliebiger Grösse	SZ	✓	25															
Generierung der QR-Codes	SZ	×	8															
FR: Anreizsystem																		
Rang ("EXP") und Achievements erweitern	RB	✓	15															
Weitere Piktogramme	RB	✓	4															
MS04: Feature Batch II																		
FR: Reporting																		
Liste der Benutzer inklusive deren Rang unter den Spielern (sortierbar nach den verschiedenen Pokalen)	RB	✓	14															
Liste der Fragen (sortierbar nach Kategorien, Schwierigkeitsgrad, Bewertung, Autoren)	RB	✓	7															
Liste der Autoren (sortierbar nach Durchschnitt Fragebewertung, Durchschnitt Schwierigkeitsgrad)	RB	✓	7															
Benutzerspezifisches Reporting (Meine Quizzes, Meine Fragen) erweitern			20															
FR: Upload/Download																		
Benutzerspez. (Urheber) XML Export	RB	✓	14															
XML Import	RB	✓	6															
MS05: Feature Batch III / End of Construction																		
Pufferzeit	T		34															
Review und Dokumentation FR und NFR	T		20															
Abschlussarbeiten																		
Technischer Bericht fertigstellen	T		10															
A0-Plakat und Abstract (Termin 26.05.2015)	T		4															
MS06: Fertigstellung / Projektabgabe																		
Pufferzeit			20															

Um gleich zu Beginn mit der bestehenden Code-Base vertraut zu werden, wurde als Erstes die Multimedia-Erweiterung der Fragetypen angegangen. Dadurch bestand die Möglichkeit, sich in den Code einzuarbeiten und gleichzeitig das erste neue Feature zu implementieren. Diese Aufgabe wurde im Team erledigt.

Anschliessend wurden die einzelnen Features untereinander aufgeteilt, um diese möglichst selbständig und effizient implementieren zu können. Diejenigen Features, die auf anderen aufbauen oder andere Features benötigen, wurden gegen Ende des Projektes eingeplant. Die genauen Details können dem Zeitplan (siehe Abschnitt 9.1) entnommen werden.

9.2. Statistiken

9.2.1. Feature-Statistiken

In dieser Grafik ist die Verteilung der Zeit pro Feature zu sehen.

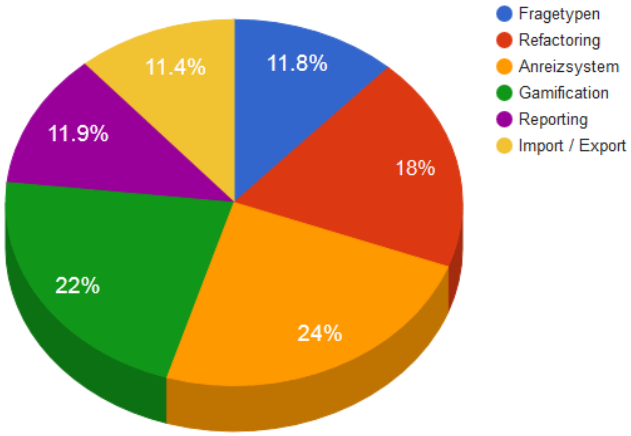


Abbildung 11: Feature-Auswertung

Die effektiv benötigte Zeit pro Feature wird in nachstehender Tabelle ausgewiesen.

Feature	**Geplant [h]**	**Effektiv [h]**
Multimedia-Erweiterung der Fragetypen	25	35.25
Anreizsystem	62	71.5
Gamification	121	65.75
Reporting	48	35.5
Import/Export	20	33.95
Refactoring	0	81.75

Es fällt auf, dass die Zeit pro Feature grundsätzlich gut veranschlagt wurde. Leider wurde keine Zeit für das Refactoring des bestehenden Systems eingeplant. Gemäss Projektauftrag sollte es kein “Bgg-Fix-Project” sein, jedoch war dies unvermeidbar.

9.2.2. Tätigkeits-Statistik

In Abbildung 12 ist die Verteilung der Zeit pro Tätigkeitsgebiet zu sehen.

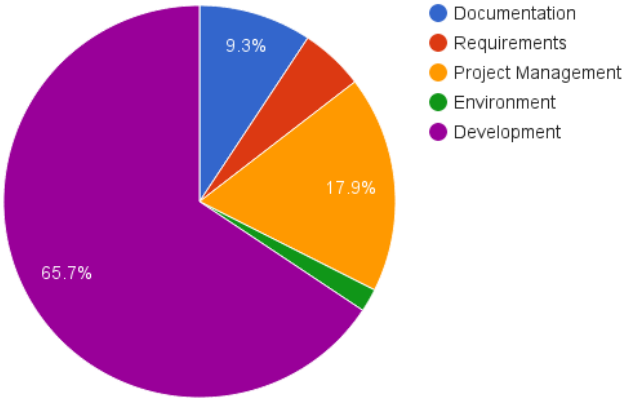


Abbildung 12: Auswertung der Tätigkeit

Die effektiv benötigte Zeit pro Tätigkeit wird in nachstehender Tabelle ausgewiesen.

Tätigkeit	**Geplant [h]**	**Effektiv [h]**
Documentation	ca. 34	47.5
Requirements	ca. 90	26.75
Project Management	0	91.25
Environment	20	9.75
Development	ca. 280	334.95

Documentation Verglichen mit dem Zeitplan in Abschnitt 9.1 wurde die Zeit für die Dokumentation zu knapp eingeplant.

Requirements Mindestens 25 zusätzliche Requirements-Stunden sind im Development zu suchen, da die Anforderungsanalyse laufend pro Feature durchgeführt wurde und somit pro Feature unter der Tätigkeit Development deklariert wurde. Trotzdem wurden ca. 40 Stunden zu viel dafür eingeplant.

Project Management Wie man sieht, ging in der Planung das Project Management mit den wöchentlichen Meetings zwischen Rico Beti und Simon Zingg sowie die Treffen mit dem Betreuer Prof. Koch vergessen. Diese konnten zwar teilweise durch den zu hoch veranschlagten Aufwand bei den Requirements und dem Development kompensiert werden. Schlussendlich ist der deutlich höhere Gesamtzeitaufwand auf diese Fehlplanung zurückzuführen.

9.3. Fazit Zeitmanagement

Obwohl die Studienarbeit mit 8 ETCS-Punkten bzw. 240 Stunden pro Teammitglied veranschlagt ist, wurde deutlich mehr Zeit benötigt. Rico Beti und Simon Zingg arbeiteten je ca. 280 Stunden am Quizzenger Projekt. Dies ist zum Teil zu erklären mit dem unerwarteten Auftreten von Bugs. Ausserdem wurde nicht genug Zeit für

das Project Management eingeplant.

Für zukünftige Projekte wie die anschliessende Bachelorarbeit müssen diese gewonnenen Erkenntnisse berücksichtigt werden.

9.4. Infrastruktur

Um grössere Inkonsistenten zu vermeiden, wurde immer auf derselben Datenbank gearbeitet. Des Weiteren wurden *Git / Github*[11] für die Versionierung eingesetzt. Sobald lokal eine stabile Änderung gemacht wurde, konnte diese mit dem Development-Branch "gemerged" werden. Über ein von Github angestossenes Skript wurde der Development-Branch direkt auf dem von der HSR bereitgestellten Linux Server deployed. Durch dieses Vorgehen wurde versucht, die Fehleranzahl zu minimieren und immer eine stabile laufende Version auf dem Server zu haben. Dies hat sich bei den Meetings mit dem Betreuer als sehr Vorteilhaft erwiesen.

Lokal wurde mit Eclipse PDT[12] und Sublime Text[13] auf Windows entwickelt.

9.5. Risikomassnahmen

Um das Projekt reibungslos und ohne grössere Überraschungen durchführen zu können, wurden verschiedene Massnahmen getroffen.

9.5.1. Security

Da unserer Vorgängergruppe gerade im Bereich Security Fehler unterlaufen sind, wollten wir uns diesbezüglich absichern.

.htaccess Massnahme Damit sich in der Entwicklungsphasen keine Angreifer auf unserem System austoben können, wurde der Zugriff über eine `.htaccess`-Datei eingeschränkt. Dadurch hatten nur berechtigte Personen Zugriff, welche über die nötigen Credentials verfügten. Es handelte sich dabei also um einen zusätzlichen Security-Layer.

Eingeschränkter Datenbank-Benutzer Der Database User, welcher von der Quizzenger-Applikation für die Datenbank-Abfragen verwendet wurde, verfügte nur über eingeschränkte Rechte. Wir folgten damit dem "Least Privilege Principle", welches besagt, dass ein User nur die unbedingt benötigten Rechte haben darf.

9.5.2. Projekt-Meetings

Es wurden wöchentlich Projektmeetings durchgeführt, um sich gegenseitig im Team auf den neusten Stand zu bringen und das weitere Vorgehen zu planen. Ausserdem wurde der Arbeitsstand kontinuierlich mit dem Zeitplan verglichen. Nach sieben Wochen – die Hälfte der Zeit – machte es den Anschein, als würden wir frühzeitig

[11] `https://github.com/`
[12] `http://www.eclipse.org/pdt/`
[13] `http://www.sublimetext.com/`

fertig werden und somit noch Zeit für die Zusatz-Features haben. In den anschliessenden Wochen wurde jedoch viel Zeit für scheinbar kleinere Änderungen und mühsame Bugs aufgewendet, wodurch der Zeitvorsprung wieder verlorenging. Ausserdem erwies sich das zuletzt geplante Import/Export-Feature als Knacknuss; auch dort wurde viel Zeit investiert.

10. Ausblick

Während der Bearbeitung der Studienarbeit wurden einige neue Features realisiert und viele Punkte verbessert. Aufgrund der gemachten Erfahrung mit dem Quizzenger-System können folgende Empfehlungen abgegeben werden, die als weitere Features sinnvoll sind:

10.1. Meldungs-System

Das Meldungssystem (Benutzer melden, Frage melden) steckt noch in den Kinderschuhen. Beispielsweise ist es nicht möglich, auf eine Meldung zu antworten. Es besteht nur die Möglichkeit, den Benutzer bzw. die Frage zu löschen oder die Meldung zu verwerfen.

Es wird hier empfohlen, das Meldungs-System soweit auszubauen, dass ein Melder und der Moderator privat – nach dem Vorbild von Facebook – kommunizieren können. Ausserdem könnte ein News-Feed gemäss Twitter erstellt werden. Die damit verbundenen Informationen können direkt auf einer attraktiveren gestalteten Startseite dargstellt werden.

10.2. Social Media Integration

Quizzenger könnte in diverse Social Medias integriert werden. Beispielsweise könnte die Möglichkeit geschaffen werden, errungene Achievements auf Facebook zu teilen. Dies war ein optionales Feature dieser ARbeit, welches jedoch aus Zeitgründen nicht umgesetzt wurde.

10.3. Markdown Editor

Zwar wurde eine Funktionalität zur Textformatierung eingebaut, diese ist momentan jedoch rein Textbasierend. Um die Benutzerfreundlichkeit zu erhöhen, wäre ein grafischer Editor wünschenswert. Dabei könnte eine Toolbar mit den wichtigsten Funktionen in Kombination mit einer Vorschau angezeigt werden, wie dies beispielsweise bei Wordpress[14] der Fall ist.

10.4. Lern-Modus ausbauen

Der derzeit angebotene Lern-Modus könnte durch ein effektiveres System ersetzt werden. Dafür würde sich eine auf dem Karteikärtchen-Prinzip basierende Lösung besonders gut eignen. Dies würde den Lernenden einen zusätzlichen Anreiz bieten, Quizzenger für die Prüfungsvorbereitung zu nutzen.

[14] `https://wordpress.org/`

10.5. Usability verbessern

Wie bei den Usability-Tests bemerkt wurde, besteht noch ein gewisses Verbesserungspotential bezüglich Benutzeroberfläche und Usability. Siehe dazu Abschnitt 6.4.

Anhang

Folgende Anhänge wurden dieser Semesterarbeit beigelegt:

1. SA Aufgabenstellung
2. SA Kurzfassung
3. SA Poster
4. Einverständniserklärung zur Publikation auf `http://eprints.hsr.ch/`
5. Persönliche Stellungnahmen
6. Benutzerhandbuch
7. Einrichtung Entwicklungsumgebung Windows
8. Installationsanleitung Quizzenger
9. Das gesamte Entwicklungsverzeichnis (Code und Abhängigkeiten)
10. Die generierte Code-Dokumentation.