Quizduell

Dominik Schaufelberger

20. April 2017

Inhaltsverzeichnis

1 Projekt

Im Rahmen der Veranstaltung *Verteilte Systeme* an der *Dualen Hochschule Baden-Württemberg* müssen alle Teilnehmer eine eigene Version des Spiels *Quizduell*[1] entwickeln. Quizduell ist eine App für Android, Windows Phone und Apple iOS. Es ist ein Spiel indem sich zwei Spieler gegeneinander antreten und mit ihrem Wissen Punkte sammeln können. In Abbildung 1 ist die Frage-Anwort-Ansicht von Quizduell zu sehen. Die App ist in 16 Ländern und Sprachen verfügbar [1].

Beim Erstellen der Anwendung sollen die in der Veranstaltung erlernten Konzepte und Techniken eingesetzt werden. Die Anwendung soll als verteiltes System umgesetzt werden. Diese Ausarbeitung dient als Dokumentation des Projektes. Zuerst werden die Spielregeln vorgestellt und die technischen Anforderungen festgelegt. Danach wird eine architektonische Übersicht über die Anwendung gegeben und die eingesetzen Technologien ausgewählt. Anschließend werden ausgewählte Teile der Implementierung gezeigt. Am Ende der Ausarbeitung wird noch ein Fazit gezogen und ein Ausblick gegeben.

[1] `http://www.quizduell-game.de/`

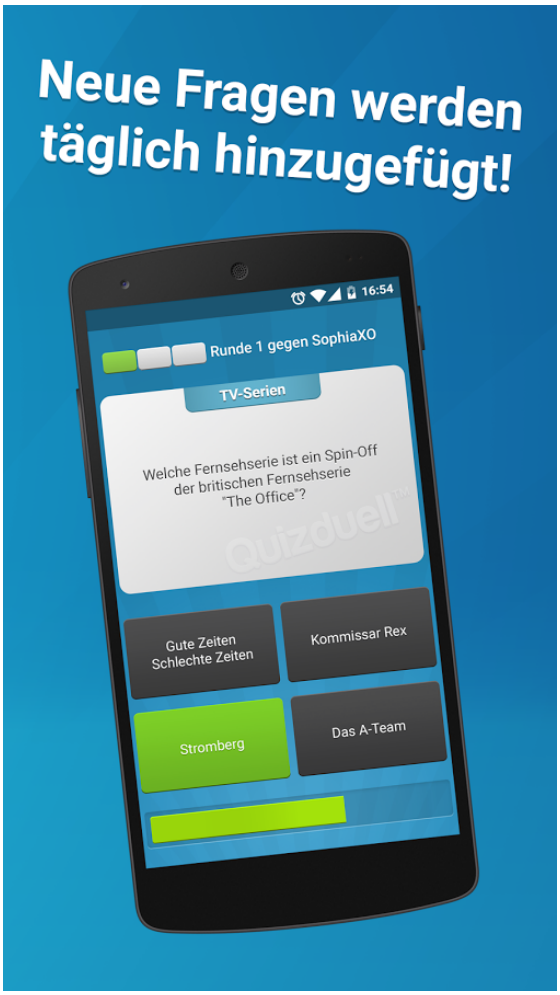

Abbildung 1: Ansicht einer Frage mit Antworten der Quizduell App (Quelle: Google Play [2]).

2 Anforderungen

In diesem Abschnitt werden die Spielregeln des Originalspiels Quizduell vorgestellt und die Anforderungen an die zu entwickelnde Anwendung spezifiziert.

2.1 Spielregeln

Bei Quizduell können Spieler landesweit gegeneinander antreten. Ein Duell wird dabei immer zwischen zwei Spielern ausgeführt und geht über sechs Runden. Jede Runde besteht aus drei Fragen, die an die beiden Kontrahenten gestellt werden. Zu jeder Frage gibt es vier Antwortmöglichkeiten von denen eine korrekt ist. Die Fragen einer Runde sind immer jeweils einer Kategorie zugeordnet, die abwechselnd von den Spieler ausgewählt werden kann. Hierzu bekommen sie drei Kategorien zur Auswahl. Das Beantworten einer Frage unterliegt einem Zeitlimit von 20 Sekunden. Der Spieler, der über alle sechs Runden hinweg am meisten Fragen beantwortet gewinnt. Abhängig von einem Rankingsystem bekommen die Spieler dann nach Spielende Punkte angerechnet oder abgezogen.

Gegenspieler können aus den eigenen Freunden ausgewählt werden – über eine Kopplung mit einem Facebook-Konto – mittels einer Namens-Suchfunktion oder zufällig gefunden werden. Es können nur Spieler mit derselben Sprachversion der App gegeneinander antreten.

2.2 Funktionale Anforderungen

Anmelden am Spielserver Damit zwei Spieler gegeneinander antreten können müssen sich sich am Spieleserver anmelden können. Beim Anmelden an den Spielerserver bekommt jeder Benutzer automatisch einen Spielernamen generiert und zugewiesen.

Zuweisung eines Gegenspielers Nachdem sich ein Spieler angemeldet hat wird ihm ein Gegenspieler zugewiesen. Die Zuweisung zweier Kontrahenten erfolgt, im Gegensatz zum Original Spiel, immer zufällig.

Starten eines Spiels Sobald die Zuweisung zweier Spieler zueinander erfolgt ist wird das Duell der beiden gestartet.

Spielen einer Runde Beide Spieler erhalten jede Runde quasi gleichzeitig die gleichen drei Fragen dieser Runde. Eine Runde gilt als beendet wenn beide Spieler alle drei Fragen beantwortet haben. Am Ende einer Runde wird beiden Spieler der Zwischenstand mit den korrekten Antworten beider Kontrahenten mitgeteilt. Ein Zeitlimit ist vorerst nicht vorgesehen.

Verteilen der Fragen Der Spieleserver verteilt pro Runde die gleichen Fragen an jeden Spieler eines Spiels. Dabei werden die Fragen nacheinander verteilt, nachdem die jeweils vorherige Frage beantwortet wurde.

Beantworten von Fragen Ein Spieler kann zu jedem Zeitpunkt immer genau eine Frage beantworten. Er muss diese beantworten um die nächste Frage zu erhalten bzw. um in die nächste Runde zu kommen. Es gibt vorerst kein Zeitlimit zum Beantworten einer Frage, im Gegensatz zum Originalspiel.

Ende des Spiels Sobald alle 18 Fragen eines Spiels beantwortet wurden gilt das Spiel als beendet. Die Spieler bekommen den Ausgang des Spieles (*Sieg*, *Niederlage* oder *Unentschieden* mitgeteilt.

2.3 Nichtfunktionale Anforderungen

Die Nichtfunktionalen Anforderungen sind im Grunde identisch mit den Anforderungen und Eigenschaften eines verteilten Systems und werden im Folgenden kurz genannt.

Heterogenität Die verwendeten Komponenten (CPUs, Laufzeitumgebungen, Betriebssysteme, Programmiersprache, etc.) müssen nicht identisch sein.

Offenheit Es gibt eine fest definierte, öffentliche Schnittstelle die ein potentieller Teilnehmer implementieren kann um die Anwendung zu nutzen.

Verfügbarkeit Das System muss zu jeder Zeit verfügbar und einsatzfähig sein. Ausfälle einzelner Komponenten werden kompensiert und bringen das Ge-

samtsystem nicht zu einem Absturz.

Skalierbarkeit Hier geht es innerhalb des Projektes nur um die horizontale Skalierbarkeit. Dies bedeutet, dass einzelne, unabhängige Komponenten hinzugefügt werden können um die Leistungsfähigkeit zu erhöhen.

Korrektheit Das System soll zu jedem Zeitpunkt korrekt arbeiten.

Transparenz Der Begriff der Transparenz umfasst mehrere Untebegriffe wie z. B. *Ortstransparenz*, *Zugriffstransparenz* und *Skalierungstransparenz*. Hier wird davon ausgegangen, dass das System in jeglicher Form transparent für den Benutzer erscheint.

Sicherheit Die Aspekte der Sicherheit spielen in verteilten System zwar eine große Rolle, sie werden für dieses Projekt jedoch vernachlässigt um den Fokus auf die anderen Eigenschaften zu legen.

3 Entwurf

Zunächst wird in diesem Abschnitt die Netzwerkarchitektur des verteilten Systems vorgestellt. Anschließend wird noch der Entwurf der Anwendungslogik sowie die Kommunikationsschnittstelle spezifiziert.

3.1 Systemarchitektur

Abbildung 2 zeigt die Netzwerkarchitektur der Anwendung und deren Komponenten. Die Anwendung ist in drei Ebenen aufgeteilt.

Den Eintrittspunkt stellt dabei ein Load Balancer dar. Es wird bewusst nur ein Load Balancer eingesetzt, auch wenn dieser dann der Flaschenhals ist und die Schwachstelle im Sinne von Ausfallsicherheit darstellt. Im Grunde muss jede Komponente redundant vorhanden sein um eine hohe Verfügbarkeit garantieren zu können. Da dies im Rahmen dieses Projektes jedoch zu aufwendig wäre, wird nur ein Load Balancer eingesetzt.

In der zweiten Ebene sind die Quizduell Server. Dieser Teil der Anwendung ist horizontal skalierbar, indem neue Anwendungsserver gestartet bzw beendet werden. Der Load Balancer leitet dabei die Anfragen an die Anwendung an die entsprechenden Server weiter. Wenn in dieser Ebene eine Skalierung stattfindet muss dies in jedem Fall auch im Load Balancer nachkonfiguriert werden. Auch weitere Anforderungen wie Heterogenität und Ausprägungen der Transparenz werden hier gewährleistet. Die Hardware und das Betriebssystem unter denen die Anwendungen laufen kann sich dabei unterscheiden. Auch wird vor dem Benutzer verborgen ob es sich um einen oder mehrere Server handelt. Praktisch greift er nur auf den Load Balancer zu, den er quasi als „Anwendung“ sieht – dessen Einsatz ihm daher auch transparent erscheint.

Alle Quizduell Server greifen zur Persistierung auf einen Cluster einer verteilten Datenbank zu. Ein solcher Cluster besteht in der Regel aus mehreren Knoten, die auf unterschiedlichen Systemen laufen können um die Ausfallsicherheit der Anwendung zu erhöhen. Die Anwendungsserver kennen in diesem Fall alle Knoten des Clusters. Optional könnte auch hier wieder ein Load Balancer eingesetzt

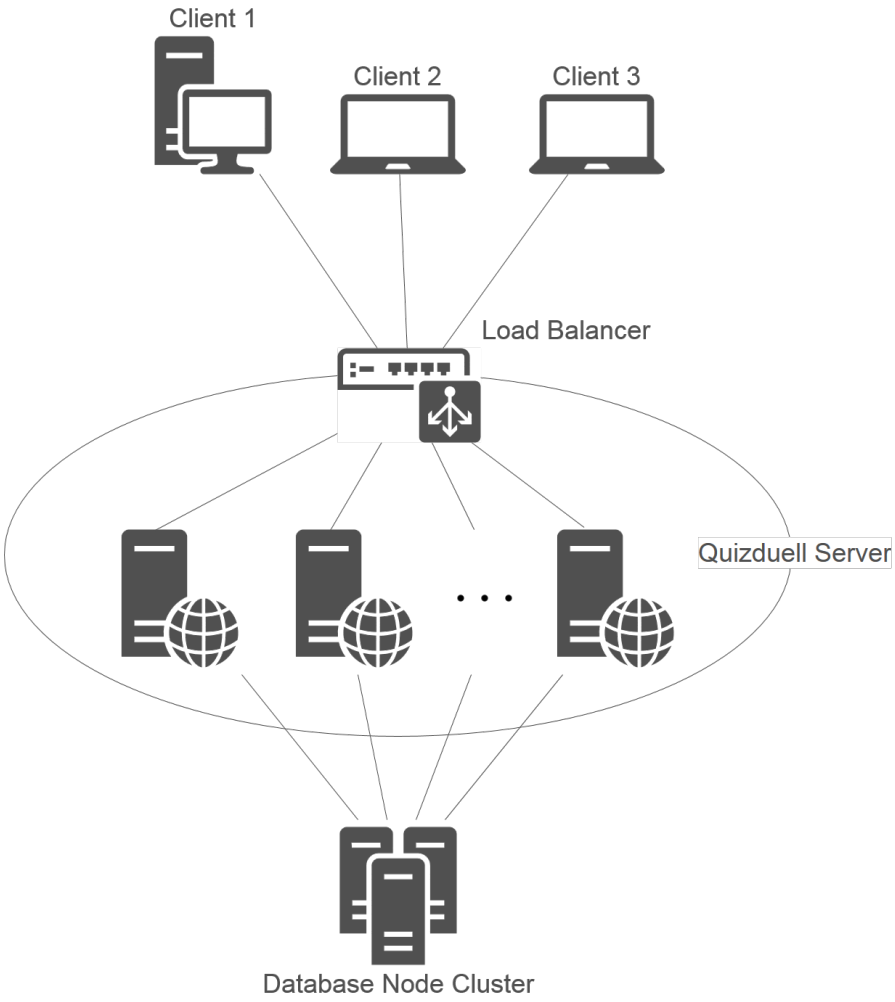

Abbildung 2: Netzwerkarchitektur der Quizduell Anwendung

werden um auch den Datenbankzugriff für die Anwendungsserver transparent zu gestalten, dies wurde hier jedoch bewusst weggelassen.

3.2 Spiellogik

Die Art der Umsetzung der Spiellogik unterscheidet sich zwischen Client und Server stark. Der Client muss die Anforderungen und Eigenschaften eines verteilten System nicht einhalten und ist daher deutlich weniger komplex. Der Server jedoch muss durch die zuvor vorgestellte Architektur entsprechend implementiert werden.

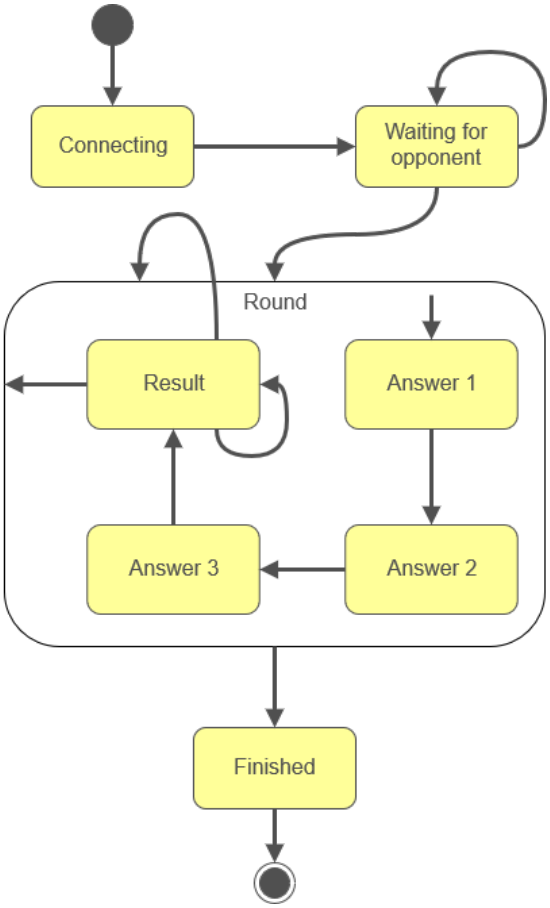

Abbildung 3: Zustandsautomat des Clients zur Umsetzung der Spiellogik.

Client

Der Client ist als Zustandsautomat konzipiert. Dieser ist in Abbildung 3 dargestellt. Die Zustandsübergänge sind mir Anfragen an den Server verbunden. Nachdem der Spieler die Anwendung gestartet hat befindet er sich im Zustand *Connecting*. der Client meldet sich dann beim Spieleserver an. Nach einer erfolgreichen Anmeldung wechselt der Client in den Zustand *Waiting for opponent* bis der Server einen Kontrahenten gefunden und dem Spieler zugewiesen hat. Wurde ein Gegenspieler gefunden, beginnt direkt die erste Runde. Die Runden sind als eigene, kleine Zustandsautomaten konzipiert. Anschließend wird durch die *Answer*-Zustände durch die Fragen und Antworten gegangen. Am Ende einer Runde steht der *Result*-Zustand. Solange der Gegenspieler nicht auch alle Fragen beantwortet hat, verbleibt er in diesem Zustand. Ist die aktuelle Runde nicht die sechste, so wird in die nächste Runde gewechselt. Sind alle Runden gespielt wird in den finalen Zustand *Finished* übergegangen und das Spiel beendet.

Server

Für die Umsetzung des Servers muss darauf geachtet werden, dass der Server sogesehen *zustandslos* sein muss. Würde ein Server den Spielzustand halten und fällt wegen eines Fehlers aus, wäre der Spielzustand verloren und könnte auch von einem anderen Server nicht rekonstruiert werden. Eine Redundanz auf $1 + n \quad (n \in \mathbb{N})$ Servern wäre eine Möglichkeit um die Verfügbarkeit zu erhöhen und mit Ausfällen transparent umzugehen, bringt jedoch einen sehr hohen Kommunikationsaufwand zur Synchronisation des Spielzustands zwischen den Servern mit. Diese Form der Redundanz beeinträchtigt gleichzeitig auch die Leistungsfähigkeit des Systems.

Die Lösung ist daher den Spielzustand nicht auf den Anwendungsserver zu halten, sondern in der Datenbank. Die Anwendung muss demnach pro Anfrage den Zustand aus der Datenbank lesen, die Aktion hinter der Anfrage ausführen und anschließend den Zustand wieder in der Datenbank persistieren. Somit können Server wegfallen oder hinzukommen ohne den Zustand eines Spiels zu gefährden. Anfragen können dann auch immer an wechselnde Server weiter geleitet werden, für den Benutzer bleibt dies transparent.

3.3 REST-Schnittstelle

Für die zustandslose Kommunikation mit der Anwendung bietet sich eine *REST*-Schnittstelle an. REST steht für *Representational State Transfer*. Dabei enthält eine Nachricht alle benötigten Informationen um die Anfrage ohne weiteren Kontext zu verstehen. Im Folgenden werden die URLs der Schnittstelle zur Interaktion mit der Anwendung aufgelistet. Als URL-Schema wird *http* eingesetzt und der Host wird auf *dhbw-quizduell.de* festgelegt. Daher ist der erste Teil der URL jeweils *http://dhbw-quizduell.de* und wird nicht mit dargestellt. Die Ausdrücke in geschweiften Klammern entsprechend entweder der Spiel-ID ($gameId$) oder der Rundennummer ($round\#$). Die Rundennummer ist dabei für jedes Spiel eine Zahl von 1 bis 6 und stellt damit keinen globalen Identifiert dar – im Gegensatz zu der Spiel-ID.

/enter Anmelden am Spieleserver.

/game/{gameId} Abrufen des Zustandes eines Spiels. Ein Spiel kann sich in ei-

nem von drei Zuständen, *noch nicht gestartet*, *gestartet* oder *beendet* befinden. Zusätzlich dazu werden auch Informationen zu den Spielern abgerufen.

/game/{gameId}/result Abrufen des Ergebnisses eines gesamten Spiels.

/game/{gameId}/round/{round#} Abrufen von Informationen zu einer Runde. Dies sind, die Nummer der Runde und die Anzahl an beantworteten Fragen beider Spieler zurück.

/game/{gameId}/round/{round#}/result Abrufen des Ergebnisses einer Runde.

/game/{gameId}/round/{round#}/question Abrufen der aktuellen Frage einer Runde.

/game/{gameId}/round/{round#}/answer Übermitteln einer Antwort zu einer Frage.

4 Implementierung

4.1 Load Balancer

Als Load Balancer wird der Webserver nginx[2] eingesetzt. Nginx ist ein leichtgewichtiger HTTP Server und *Reverse Proxy*[3]. Durch die Reverse Proxy Funktionalität lässt er sich wie in Listing 4.1 zu sehen, sehr einfach als Load Balancer konfigurieren. Das Scheduling-Verfahren zur Lastverteilung ist standardmäßig ein Round-Robin[4] Verfahren. Die Konfiguration kann zur Laufzeit des Gesamtsystems angepasst werden indem Server hinzugefügt oder entfernt werden und anschließend mit dem Befehl `nginx -s reload` neugeladen werden. Die Reverse Proxy Funktion von nginx führt auch sogenannte *Health Checks* durch und kann dann Server, die nicht mehr erreichbar sind, von der Lastverteilung ausschließen. [3]

```
1  http {
2      upstream quizduell {
3          server srv1.dhbw-quizduell.de;
4          server srv2.dhbw-quizduell.de;
5          server srv3.dhbw-quizduell.de;
6      }
7
8      server {
9          listen 80;
10         server_name dhbw-quizduell.de;
11         location / {
12             proxy_pass http://quizduell;
13         }
14     }
15 }
```

Listing 4.1: Konfiguration des nginx-Webservers als Load Balancer (nach [3])

[2] https://nginx.org/en/
[3] https://de.wikipedia.org/wiki/Reverse_Proxy
[4] https://en.wikipedia.org/wiki/Round-robin_scheduling

4.2 Anwendungsserver

Die REST-API des Anwendungsservers wird über das *Restlet Framework*[5] implementiert. Für jede URI der REST-API gibt es einen Handler der Anfragen an den Server über diese URI verarbeitet. Die jeweilige Klasse ist dabei eine Unterklasse von `ServerResource` aus dem Restlet Framework. Die Handler-Methoden sind mit Java Annotationen entsprechend der HTTP-Verben[6] versehen. Die Handler greifen dann auf das darunterliegende Domänenmodell zu, in dem die Spiellogik implementiert ist. Die REST-API ist JSON-basiert, d. h. der Payload der Antworten wird in JSON formatiert.[7]

In Abbildung 4 ist das Klassendiagramm des Domänenmodells zu sehen. Auf die Klassen und ihre Funktionen wird hier nicht näher eingegangen. Neben der Implementierung der Spiellogik dienen sie auch direkt als Entitäten zur Persistierung über die Java Persistence API (JPA)[8]. Als JPA Provider kommt Hibernate zum Einsatz. Dies ist ein weit verbreitetes Mapping-Framework und bietet neben objekt-relationalem Mapping auch eine Implementierung für NoSQL Datenbanken. Somit lässt sich im Optimalfall die Datenbank-Technologie im Hintergrund ersetzen.

4.3 Verteilte Datenbank

Mit *Apache Cassandra*[9] kommt eine verteilte NoSQL-Datenbank zum Einsatz. Die Datenbank besteht dabei aus einem Cluster von einem oder mehrerer Knoten. Ein solcher Cluster aus drei Knoten ist in Abbildung 5 zu sehen. Dort ist die Abfragebearbeitung an einen Cassandra-Cluster zu sehen. Die Anfrage wird dabei konkret an den Knoten *Node 1*, der Datenbank gestellt – in der Grafik grün dargestellt und zusätzlich als *Coordinator* bezeichnet. Dieser Coordinator kommuniziert dann mit den anderen Knoten innerhalb des Cluster um die Anfrage des Benutzers zu erfüllen. Er liefert dann auch das ermittelte Ergebnis zurück.

[5] `https://restlet.com/open-source/`
[6] `http://www.restapitutorial.com/lessons/httpmethods.html`
[7] `http://www.json.org/`
[8] `https://docs.oracle.com/javaee/7/tutorial/persistence-intro.htm#BNBPZ`
[9] `https://cassandra.apache.org/`

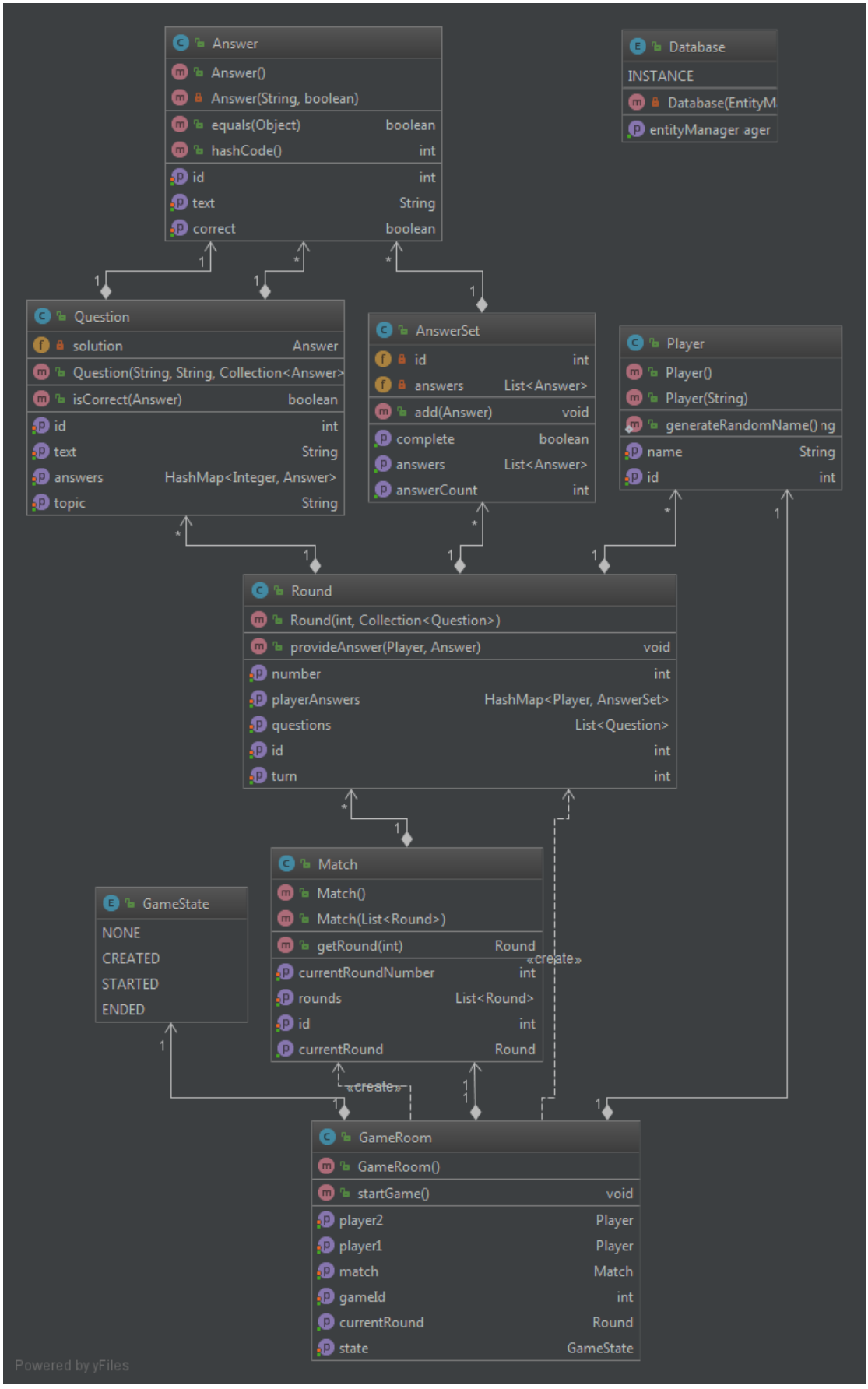

Abbildung 4: Klassendiagramm des Domänenmodells (Grafik autogeneriert durch *IntelliJ IDEA 2017*).

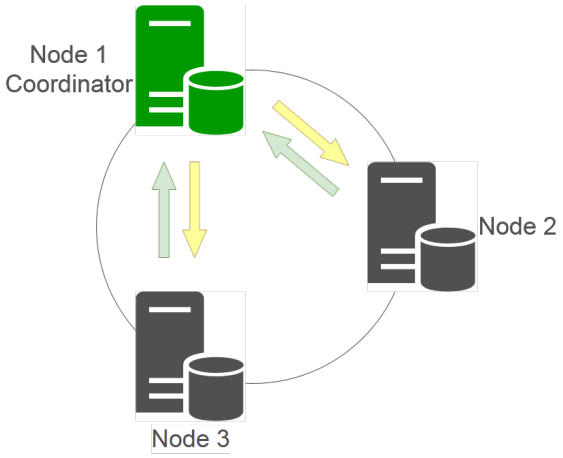

Abbildung 5: Anfragebearbeitung in einem Cassandra-Cluster mit drei Knoten [4]

5 Fazit

Die Implementierung des Projektes konnte nicht komplett fertig gestellt werden. Aktuell kommt noch eine MySQL-Datenbank anstelle der verteilten Datenbank Cassandra zum Einsatz. Das ist der Hardwareleistung des Systems geschuldet, auf welchem entwickelt wird. Dieses ist nicht in der Lage mehr als eine virtuelle Maschine auszuführen, was aber für eine korrekte, verteilte Anwendung notwendig ist. Umgesetzt wurde der Load Balancer, ein Teil des Domänenmodells inklusive Persistenz und der dazu gehörende Teil der REST-API. Austehend sind demnach noch die restliche Implementierung der API und Spiellogik, die Umsetzung mit der Cassandra-Datenbank sowie die Implementierung eines Clients.

Durch das Projekt wurden die Bestandteile eines verteilten System näher betrachtet und teilweise umgesetzt. Gerade im Bezug auf REST-APIs gab es einige Erkenntnisse durch das Paradigma *Alles ist eine Ressource*. Trotzdessen, dass das Projekt nicht beendet werden konnte, hat es ein besseres Verständnis solcher Systeme hervorgebracht.

Abbildungsverzeichnis

Literatur

[1] Wikipedia, Hrsg., *Quizduell*. Adresse: `https://de.wikipedia.org/w/index.php?oldid=164254114` (besucht am 12. 04. 2017).

[2] *Quizduell – Android-Apps auf Google Play*. Adresse: `https://play.google.com/store/apps/details?id=se.feomedia.quizkampen.de.lite` (besucht am 09. 04. 2017).

[3] *Using nginx as HTTP load balancer*. Adresse: `https://nginx.org/en/docs/http/load_balancing.html` (besucht am 12. 04. 2017).

[4] A. Mehra, *Introduction to Apache Cassandra's Architecture*. Adresse: `https://dzone.com/articles/introduction-apache-cassandras` (besucht am 06. 04. 2017).