Projektdokumentation Quizduell

Von Kassandra Frank und Ruth Weber

Einleitung

Im Rahmen der Vorlesung Verteilte Systeme an der DHBW Karlsruhe sollen die verschiedenen Zweiergruppen jeweils ihre eigene Vision von Quizduell entwickeln. Quizduell ist eine App, in der jeweils zwei Spieler gegeneinander spielen und Fragen beantworten.

Bei der Entwicklung sollen die in der Vorlesung erlernten Konzepte und Technologien eingesetzt werden. Dabei soll das Spiel als verteiltes System umgesetzt werden. Die Projektdokumentation soll die Spielregeln, die Anforderungen und die Umsetzung beschreiben.

Allgemein

Spielregeln

- Pro Spiel werden sechs Runden mit je drei Fragen gespielt.
- Die jeweiligen Runden werden von den Spielern abwechselnd gespielt.
- Beide Spieler bekommen die gleichen Fragen gestellt.

- Jeder Spieler bekommt für gewonnene Spiele Punkte und für verlorene Spiele Punktabzug. Die Punktzahl hängt unter anderem vom Ranking des Gegners ab.
- Wer sein Spiel aufgibt bekommt 24 Punkte Abzug. Derjenige, gegen den aufgegeben wurde, bekommt die Punkte, als ob er gewonnen hätte.
- Jedes Spiel muss innerhalb von 48h beendet werden. Wird das Limit überschritten und du lagst in Führung bekommst du die Punkte. Lagst du nicht in Führung, werden dir aber keine Punkte abgezogen.
- Der Gewinner bekommt 5 Punkte, dem Verlierer wird ein Punkt abgezogen.

Konzept

Anforderungen (Lastenheft)

- das System muss Fragen, Spieler und Punktestände speichern
- das System muss hochverfügbar sein
- das System muss skalierbar sein
- das System muss verteilt sein
- gegeneinander spielende Spieler müssen die gleichen Fragen erhalten

Pflichtenheft

1. redundante Speicherung
2. verteilte Datenbank mit MySQL
3. Load Balancer
4. RestServer, die die Anfragen weiterleiten

Geplantes Vorgehen

Die Entwicklung des Spiels ist wie folgt geplant. Zuerst sollen die Model Klassen für die Erstellung der Tabellen in der Datenbank erstellt werden. Dafür soll ein Persistence-Framework (Hibernate) verwendet werden, das aus den Klassen, die Tabellen mit Primärschlüssel und Fremdschlüssel erstellt.
Nach dem die Datenbank aufgesetzt wurde, soll die Logik implementiert werden. Das heißt, die verschiedenen Funktionen, um das Spiel zu spielen.
Wenn die Logik implementiert ist und funktioniert, soll ein Load Balancer (nginx) integriert werden, der mehrere Server verwaltet, ausbalanciert und somit die Last verteilt. Damit wäre die Hochverfügbarkeit abgedeckt. Wenn mehrere Server angesprochen werden können, dann soll eine verteilte Datenbank eingesetzt werden, damit die Daten ebenfalls abgesichert sind und von den Servern aus darauf zugegriffen werden kann. Dieser Punkt wird eher nach hinten geschoben, weil wir zuvor schon viel Zeit damit verbracht haben Cassandra zum Laufen zu bringen und wird daher zuerst eine SQL Datenbank verwenden und dies später austauschen.
Am Ende soll ein simpler Client implementiert werden, durch den die verschiedenen Funktionen aufgerufen werden können.

Systemarchitektur

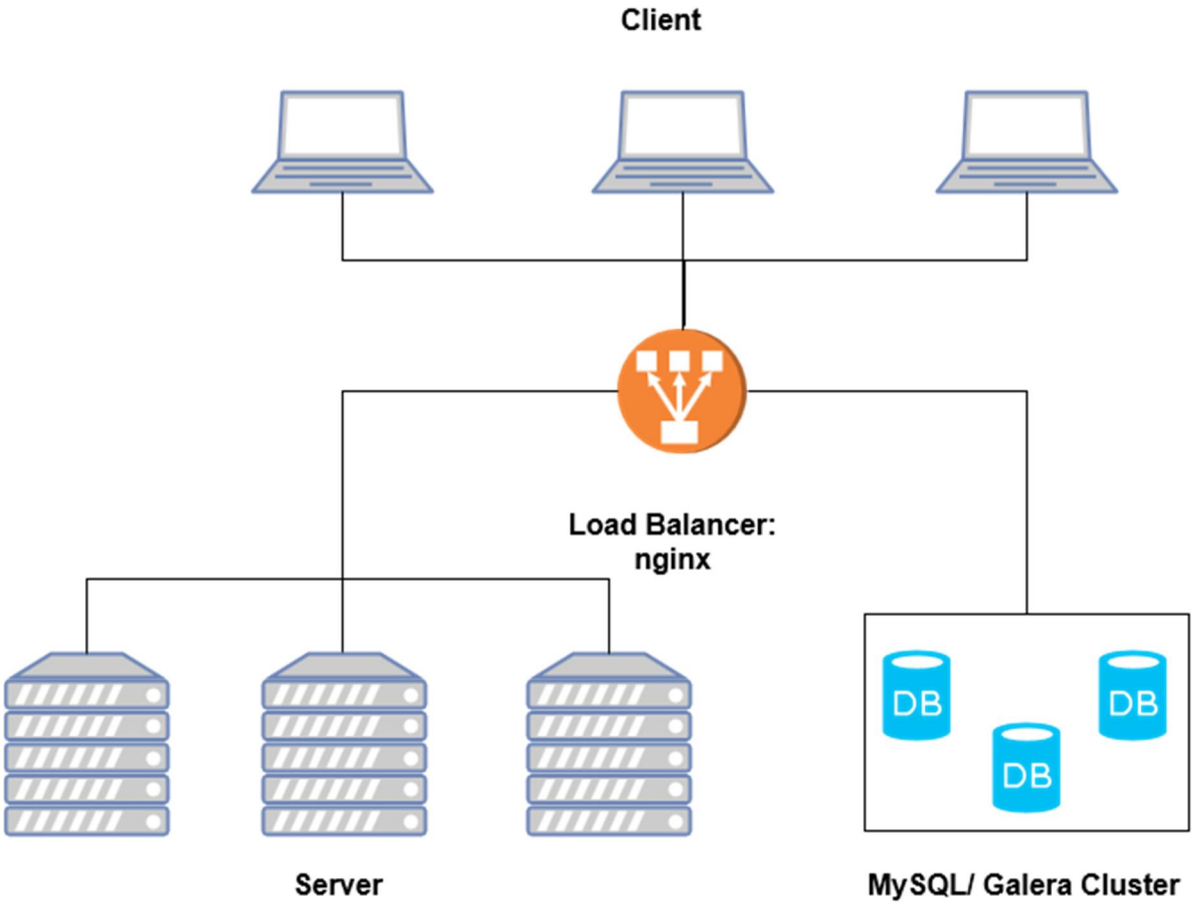

Abbildung 1 Netzarchitektur von Quizduell

Die Systemarchitektur ist wie in Abbildung 1 dargestellt aufgebaut.
Oben befinden sich die Clients, die Benutzer die Quizduell spielen möchten. Der Client ruft das Spiel über eine Internetadresse, unter der der Server erreichbar ist, auf.
Um die Hochverfügbarkeit des Systems bereitzustellen, wird zwischen den Servern und den Clients ein Load Balancer eingesetzt, in unserem Fall wird dies der *nginx* sein. Dieser verteilt die Anfragen auf die verschiedenen Server, so dass diese ausgeglichen ausgelastet sind. Dabei kann ein Ausfall abgefangen werden. Die Anfragen des Clients werden an den Server übermittelt. Um einen Datenverlust zu vermeiden werden die benötigten Informationen aus den Anfragen direkt in die Datenbank geschrieben. Als Datenbank wird in unserem Fall MySql mit Galera Cluster eingesetzt. Damit ist der Ausfall der Datenbank und damit das nicht verfügbar sein der Daten abgefangen.

Entity Relationship Diagramm

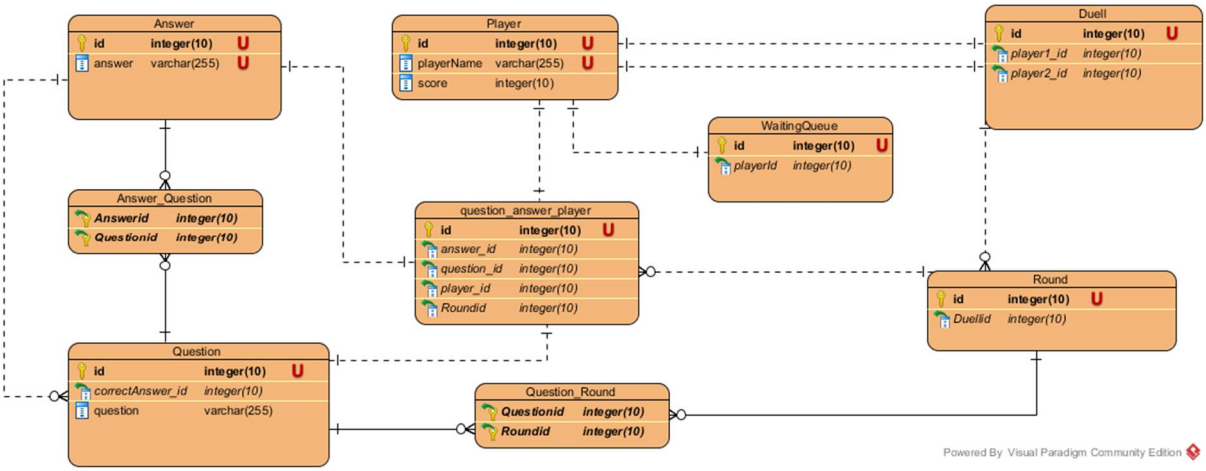

Abbildung 2 ERD der Datenbank für Quizduell

In Abbildung 2 ist das ERD für die Quizduell Datenbank dargestellt.
Da alle neuen Spielstände und Veränderungen direkt in der Datenbank persistiert werden sollen, haben wir nun folgendes Entity Relationship Diagramm (siehe Abbildung 2) erstellt. Zunächst wird eine Tabelle **Player** benötigt, welche alle registrierten Spieler mit ihrem Spielernamen und dem aktuellen Punktestand bereithält. In der Tabelle **WaitingQueue** werden alle Spieler eingetragen, die sich angemeldet haben und nun auf einen Gegenspieler warten. Hier sollen dann jeweils zwei Spieler herausgenommen werden und in der Tabelle **Duell** als Spieler1 und Spieler2 eingetragen werden. Die benötigten Runden für das Duell sind in der Tabelle **Round** persistiert. Eine Runde hat jeweils eine Referenz auf das zugehörige Duell über die **DuellId**. Zusätzlich muss eine Runde noch mit Fragen und den dazugehörigen Antworten befüllt werden. Hierfür gibt es die Tabellen **Question** und **Answer**. Zwischen **Round** und **Question** besteht eine n:m-Beziehung, da in einer Runde mehrere Fragen enthalten sind (konkret hier 3) und eine einzelne Frage aber auch in verschiedenen Runden auftreten kann und nicht nur zu einer konkreten gehört. Auch zwischen den Tabellen **Question** und **Answer** besteht eine n:m-Beziehung, da zu einer Frage mehrere Antwortmöglichkeiten geboten werden (konkret 4), aber eine Antwort auch für mehrere Fragen als sinnvoll erachtet werden kann. Zusätzlich beinhaltet die Tabelle **Question** eine Referenz auf die **Answer**-Tabelle, welche die korrekte Antwort für diese Frage markiert. Damit auch jeweils zu einem Duell die Antworten der Spieler zu einer bestimmten Frage eingespeichert werden können wurde zusätzlich die Tabelle **question_answer_player** erstellt. Diese Tabelle besitzt jeweils Referenzen auf **Question, Answer, Player** und **Round**.

Umsetzung

Hibernate

Zum Auslesen, Löschen und Persistieren der Daten soll das ORM- und Persistence-Framework Hibernate verwendet werden. Hibernate bietet die Möglichkeit Objekte anzulegen, welche dann als Entitäten in der Datenbank gespeichert werden. Der Benutzer selbst muss sich keine Gedanken darüber machen, wie er die Objekte in die Datenbank persistiert, sondern kann dazu den EntityManager von Hibernate verwenden.
In Listing 1 ist beispielhaft die Klasse **Question** mit den benötigten Hibernateannotationen gezeigt. Zunächst wird die Klasse mit @Entity annotiert, so dass Hibernate diese als Entität behandelt und eine entsprechende Tabelle mit Referenzen in der hinterlegten Datenbank anlegt. Für jedes Attribut wird automatisch eine Spalte in der Datenbanktabelle angelegt. Zusätzlich ist es nötig ein Attribut als ID (also als Primärschlüssel(zu annotieren, damit die Objekte sich jeweils eindeutig zuordnen lassen. Durch die Annotationen @ManyToMany wird eine n:m-Beziehung gekennzeichnet. In unserem Beispiel wird dies dann als Liste von Answer-Objekten im Objekt des Typ Question gehalten. Durch die Annotationen @ManyToOne und @OneToMany lassen sich auch die 1:n-Beziehungen abbilden.

```
@Entity
public class Question {

   @Id
   private long id;

   private String question;

   @ManyToMany(fetch = FetchType.EAGER)
   @Fetch(FetchMode.SELECT)
   private List<Answer> answers;

   @ManyToOne(fetch = FetchType.EAGER)
   private Answer correctAnswer;

   public Question() {}

   public long getId() { return id; }

   public void setId(long id) { this.id = id; }

   public String getQuestion() { return question; }

   public void setQuestion(String question) { this.question = question; }

   public List<Answer> getAnswers() { return answers; }

   public void setAnswers(List<Answer> answers) { this.answers = answers; }

   public Answer getCorrectAnswer() { return correctAnswer; }

   public void setCorrectAnswer(Answer correctAnswer) {
       this.correctAnswer = correctAnswer;
   }
```

```
}
```

Listing 1 Klasse Question mit Hibernate Annotation

nginx

Als Load Balancer wird nginx eingesetzt, der die Anfragen auf Quizduell auf die Server verteilt. Es sollen drei Server parallel laufen. In Listing 2 ist die Konfiguration des Servers dargestellt.

```
http {
      upstream restserver {
          server server1:8085;
          server server2:8085;
      }
      server {
          listen 80;

          location / {
              proxy_pass http://restserver;
              proxy_connect_timeout 2;
          }
      }
}
```

Listing 2 Konfiguration des nginx für den REST Server der Quizduell Anwendung

Datenbank

Wie schon oben beschrieben wurde zuerst versucht das Projekt mit Cassandra als verteilte Datenbank umzusetzen, nach dem es nach langem hin und her versuchen nicht funktioniert hat, weil die Hibernate Annotations noch nicht vollständig entwickelt ist, wurde auf MySQL umgestellt. Zunächst sollte damit nur die Entwicklungszeit überbrückt werden, jedoch gab es einen Hinweis darauf, dass man auch MySQL über Galera als Cluster[1] verwenden kann, dieses haben wir genutzt und 3 Nodes aufgesetzt.
Dafür benutzen wir Docker, in Listing 3 sind die Kommandozeilen Befehle dargestellt.

```
sudo docker run --detach=true --name node1 -h node1 erkules/galera:basic
--wsrep-cluster-name=local-test --wsrep-cluster-address=gcomm://
```

```
sudo docker run --detach=true --name node2 -h node2 --link node1:node1
erkules/galera:basic --wsrep-cluster-name=local-test --wsrep-cluster-
address=gcomm://node1
```

```
sudo docker run --detach=true --name node3 -h node3 --link node1:node1
erkules/galera:basic --wsrep-cluster-name=local-test --wsrep-cluster-
```

[1] http://galeracluster.com/products/

```
address=gcomm://node1
```

Listing 3 Die Docker Kommandozeilen Befehle zum Starten der MySQL/Galera Knoten

Die Verteilung der Daten auf die 3 Nodes wird über nginx gesteuert. Listing 4 zeigt das Konfigurationsfile.

```
user www-data;  ## Default: nobody
worker_processes  5;  ## Default: 1
worker_rlimit_nofile 8192;
events {
  worker_connections  4096;  ## Default: 1024
}
stream {
      upstream database {
          zone tcp_servers 64k;
          server node1:3306;
          server node2:3306;
          server node3:3306;
      }
      server {
          listen        1234;
          proxy_pass    database;
          proxy_connect_timeout 1s;
      }
}
```

Listing 4 Konfiguration des nginx für das Datenbank Cluster

Der nginx wird über die Docker Compose gestartet. Die dafür benötigte docker-compose.yml ist in Listing 5 dargestellt.

```
nginx:
 image: nginx
 container_name: nginx
 external_links:
  - node1:node1
  - node2:node2
  - node3:node3
 volumes:
  -
~/Data/docker/mount/nginx/config/nginx.conf/default.conf:/etc/nginx/ngin
x.conf:ro
 ports:
  - 1234:1234
```

```
restart: always
```

Listing 5 Die docker-compose.yml Konfiguration zum Starten des nginx in Docker

Anwendungsserver

Der Anwendungsserver hat eine REST-API. Beim Aufruf einer definierten URI wird diese Anfrage vom Server verarbeitet. Dieser greift dann auf die entsprechenden Ressource zurück. Die Antwort wird in JSON Notation zurückgegeben.
Damit Quizduell auf den Servern läuft wird über Maven eine .jar Datei erstellt und diese dann auf dem Server ausgeführt.

REST

Für die Kommunikation zwischen Client und Server wird eine REST Schnittstelle eingesetzt. Dies hat den Vorteil, dass kein Kontext benötigt wird, um die Anfragen zu verarbeiten. Es gibt verschiedene URLs die zur Interaktion benötigt werden. Standardmäßig wird die IP Adresse, der Port angegeben und folgender Pfad `/rest` angegeben.
Im Folgenden werden die daran anhängenden Pfade näher erläutert.

`/game/login` Hier wird die Startseite auf der sich der Spieler einloggen kann aufgerufen.
`/game/player/{playerId}` Mit dieser URL werden die Duelle des Spielers zurückgegeben
`/game/player/{playerId}/round/{roundId}/question/{questionId}/answer/{answerId}` Über diese URL wird die Antwort eines Spielers zu einer Frage, einer Runde, eines Duells gesetzt.
`/game/player/{playerId}/duell/{duellId}` Es wird geprüft, ob es beendete Duelle gibt und wenn es diese gibt, dann wird der Gewinner ermittelt, der neue Score jedes Spielers berechnet und das beendete Duell gelöscht.
`/game/player/{playerId}/newDuell` Beim Aufruf wird geprüft, ob es in der Waiting Queue zwei offene Spielanfragen gibt. Für diese werden dann ein Duell angelegt und die Runden generiert.

Client

Wenn die URL /game/login aufgerufen wird, dann soll im Browser Abbildung 3 angezeigt werden. Um spielen zu können, muss der Benutzer seinen Namen eingeben. Dieser wird dann an den Server gesendet, der die verfügbaren Duelle ermittelt.

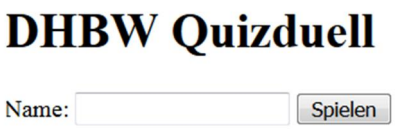

Abbildung 3: Login Seite von Quizduell

Wenn der Spielername existiert sollen alle verfügbaren Duelle des Benutzers, wie in Abbildung 4 dargestellt, angezeigt werden, indem er auf die URL /game/player/{playerID} weitergeleitet wird.

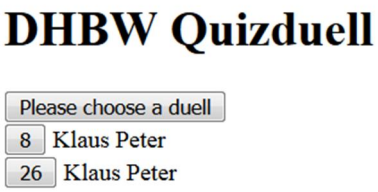

Abbildung 4 Die offenen Duelle des Benutzers zum Wählen

Wenn der Benutzer ein Spiel auswählt, dann kann eine Runde jeweils über den Button, siehe Abbildung 5 auswählen.

Abbildung 5 Die Runden eines Duells

Nach Wahl der Runde werden dem Benutzer die drei Fragen dieser Runde mit den jeweils vier Antwortmöglichkeiten angezeigt.

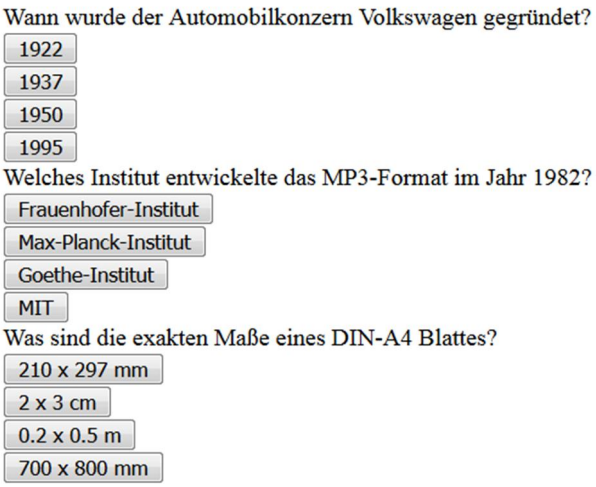

Abbildung 6 Die drei Fragen einer gewählten Runde mit den Antwortmöglichkeiten

Fazit

Am Ende muss man sagen, dass die Zeit knapp geworden ist, da sowohl die Klausurenphase, als auch die Endphase der Studienarbeit begonnen hat. Wenn zu Beginn weniger Zeit mit dem Versuch verbracht worden wäre, Cassandra zum Laufen zu bringen, hätte die Entwicklung weiter fortgeschritten sein können.
Bei der Implementierung der Logik war es ebenfalls so, dass wir um zu viele Ecken gedacht und damit zu kompliziert gedacht haben und viel Zeit bei der Überlegung, wie man etwas umsetzt verbracht haben.
Ebenfalls ist sehr viel Zeit auf das Eindenken in die verschiedenen Technologien geflossen.