**M165 Pokémon Quiz Dokumentation**

Inhaltsverzeichnis

[Unser Thema 2](#_Toc157170896)

[Vorgehensweise 2](#_Toc157170897)

[Quiz Ablauf 3](#_Toc157170898)

[Beispiellauf 3](#_Toc157170899)

[DB erstellen 5](#_Toc157170900)

[DB testen 5](#_Toc157170901)

[DB werte lesen in Java 8](#_Toc157170902)

[Höchste oder tiefste Werte 8](#_Toc157170903)

[Datenbank durchlaufen 9](#_Toc157170904)

[Von BSON zu Java 9](#_Toc157170905)

[Highscore 10](#_Toc157170906)

[Quiz Logs 10](#_Toc157170907)

[Top 5 Spieler 10](#_Toc157170908)

[MongoDB Atlas 11](#_Toc157170909)

[Source Code 11](#_Toc157170910)

## Unser Thema

Beim Projektauftrag, welcher einen Grossteil unserer Note ausmacht, entschieden wir uns für ein Java basiertes Projekt. Das Projekt: eine Quiz Applikation welche auf eine MongoDB lesend und schreibend zugreifen kann.

Wir wählten als Thema unseres Quiz Pokémon Spielkarten. Die Attribute oder Themenbereiche, nach welchen die Fragen gestellt werden, wählten wir wie folgt:

1. Health
2. Strength
3. Energy
4. Stage

Wir haben uns für diese 4 Kategorien entschieden, weil dies Werte sind, die auf Pokémon Karten stehen. Für Health haben wir die HP, für Strength haben wir den Schaden der stärksten Attacke, für Energy ist es die Anzahl Energie Karten, die man braucht, um diese Attacke auszuführen und für Stage haben wir die Stufe der Evolution, wobei wir für die Stufe "Basic " Stage 1 haben und für "Stage 1" Stage 2 usw.

## Vorgehensweise

Ein Bild, das Text, Schrift, Screenshot, Design enthält.

Automatisch generierte BeschreibungAls aller erster Schritt in der Erarbeitung unseres Projekts haben wir erst ein config file für einen Docker-Container erstellt. Ebenfalls haben wir eine init.js Datei erstellt, welche unsere Pokémons beinhaltet. Nach dem wir eine laufende DB hatten haben wir kontrolliert, ob diese auch die Werte drinnen hat.

Unsere nächste Aufgabe war es 3 zufällige Pokémons aus der Datenbank zu holen, dies haben wir zum Anfang so gemacht, dass wir alle Pokémons in eine Liste getan haben, um zu sehen ob überhaupt die Verbindung funktioniert. Nach dem haben wir dann entschieden diese 3 Pokémons nicht in der Java Liste zufällig auszuwählen, sondern über Aggregationen und haben das Implementiert. Danach haben wir für unsere 4 Kategorien, welche wir am Anfang definiert haben, eine Liste von Fragen erstellt, aus welcher man eine zufällige Frage erhält, je nach welcher Kategorie man wählt.

Ein Bild, das Screenshot, Text, Schrift, Informationen enthält.

Automatisch generierte BeschreibungBeim Punkt wo es darum ging welche Kategorie man auswählt sind wir auf das Problem gestossen, das wir bei einer Falschen Antwort keine Fragen zu den Kategorien zurückbekamen, dies wurde behoben mit wenig Zeilen Code.

Nachdem wir an unserem UX, mit dem Terminal, angefangen haben zu arbeiten entschlossen wir die Logs, die im Terminal gezeigt werden, zu entfernen den diese verschieben die System.out.prints(); und die Frage wird nach oben verschoben, nach dem Pokémons aus der DB gewählt werden. Dies lösten wir mit einem Logback file, welches die Informationen zur Verbindung in den Hintergrund verschiebt, sodass unser Quiz ordentlicher aussieht.

Wenn man das Quiz nun starten würde, wählt man eine Kategorie aus, erhält eine Frage und Drei zufällige Pokémons. Unser nächster Schritt war es zu vermeiden das Pokémons mit demselben **gewinn Wert** (winningCondition) nicht als mögliche Antwort für eine Frage kommen. Hier stossen wir auf ein Zweites Problem, denn wir wollten nicht nur highest und lowest abfragen machen, sondern auch clossest to. Dabei entschieden wir uns für die Fragen selbst ein Objekt zu erstellen mit den werten für Question, Value und Category. Wir lösten schlussendlich 2 Probleme und eine Aufgabe, die wir noch machen wollten, indem wir beim zufälligen Picken des Pokémons direkt den gewinn Wert berechneten, und beim nächsten Pokémon nur drauf achteten, ob die denselben gewinn Wert haben. Dann wiederholte sich der Loop bis endlich 3 Pokémons da waren und die richtige Antwort ist der mit der tiefsten Differenz zu Value der Frage = gewinn Wert.

Wir mussten noch implementieren, dass man einen User erstellen könne, welcher in einer DB gespeichert wird. Gespeichert wird dieser mit Namen, Punkte und der gebrauchten Zeit. Wir haben ein Leaderboard erstellt, welches die top fünf Spieler am Ende jedes Durchlaufs ausgibt. Diese sind erst nach Punktzahl und dann nach Zeit sortiert.

Da wir soweit fertig mit unserem Projekt waren, entschieden wir uns noch einiges extra zu machen. Daher haben wir die DB online mit MongoDB Atlas online hochgeladen. Ebenfalls haben wir noch einige Test geschrieben und schliesslich mehr MongoDB Aggregations eingefügt.

## Quiz Ablauf

Beispielsweisse wird dann für das Quiz die Frage kommen für welche Kategorie man befragt werden will, man gibt die Antwort im Terminal ein und bekommt eine Frage mit 3 Pokémons zur Auswahl.

### Beispiellauf

Ein Bild, das Text, Schrift, Screenshot, Grafiken enthält.

Automatisch generierte Beschreibung

Der Spieler gibt einen Namen ein.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Er wählt eine Kategorie.

Ein Bild, das Text, Schrift, Reihe, Screenshot enthält.

Automatisch generierte Beschreibung

Und erhällt drei zufällige Pokémons von welchen er eins als

Antwort wählt.

Ein Bild, das Text, Screenshot, Reihe, Schrift enthält.

Automatisch generierte Beschreibung

Das Quiz gibt nun die Antwort aus und fragt, ob man aufhören möchte. Wenn man mit "yes" antwortet, endet der Quiz und der Name, Score und die Zeit werden in einer DB geloggt. Wenn man nicht antwortet, geht es weiter mit der nächsten Quiz Frage.

Man kann nur 5 Fragen beantworten danach endet das Quiz selbst.

Jetzt wird uns unser Resultat und die Top 5 Resultate angezeigt.

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

## DB erstellen

Mit einem config file erstellen wir die Anforderungen für unsere Mongodb in welcher wir die Pokémons und die dazugehörigen Daten speichern. Die Daten zu den Pokémons haben wir in einem init.js File notiert. Beim erstellen des Docker-containers werden die Pokémons mit ihren Werten in der Datenbank gespeichert.

Als Anmeldedaten wählten wir für Nutzernamen: "root" und "1234" als Passwort was auch im config file definiert ist.

Mit dem command: "docker-compose up" konnten wir schliesslich den Container erstellen, welcher bei uns auf Docker Desktop am laufen ist.

Ein Bild, das Text, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

*# Monog DB and Mongo-Express and initial pokemondb*version: '3.9'  
  
services:  
  
 mongodb:  
 image: mongo  
 restart: always  
 ports:  
 - 27017:27017  
 environment:  
 MONGO\_INITDB\_ROOT\_USERNAME: root  
 MONGO\_INITDB\_ROOT\_PASSWORD: 1234  
 volumes:  
 - ./init.js:/docker-entrypoint-initdb.d/mongo-init.js:ro  
  
 mongo-express:  
 image: mongo-express  
 restart: always  
 ports:  
 - 8081:8081  
 environment:  
 ME\_CONFIG\_MONGODB\_ADMINUSERNAME: root  
 ME\_CONFIG\_MONGODB\_ADMINPASSWORD: 1234  
 ME\_CONFIG\_MONGODB\_URL: mongodb://root:1234@mongodb:27017/

### DB testen

Um zu testen ob die Pokémons nun in der Datenbank vorhanden sind, verwendeten wir MongoDBCompass. Um sich da zu verbinden, geben wir nur den Connection String ein.

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

Mit diesem Tool kann man sich mit der DB verbinden und die Werte betrachten/filtern.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte BeschreibungDas gleiche ist in IntelliJ selbst möglich. Dies ist so möglich, dass man in der IntelliJ auf der rechten Seite Database auswählt. Somit erscheint ein Fenster wo man das "+" Symbol antippt. Man wählt "Data Source" und sucht nach MongoDB. Damit erscheint erneut ein Fenster, mit welchem man sich mit der DB verbindet.

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte BeschreibungUm die Datenbank betrachten zu können, muss man unter anderem Host, Port, URL und in unserem Fall auch User eingeben. Danach kann man sich die Datenbank anschauen und aus IntelliJ aus editieren.

Ein Bild, das Text, Screenshot, Display, Software enthält.

Automatisch generierte BeschreibungUnd unter Pokémons sieht man die Liste aller Pokémons die in der Daten bank vorhanden sind.

## DB werte lesen in Java

Um jetzt mit den Pokémons aus der Datenbank arbeiten zu können, muss man die Werte mit Java einlesen. Wir haben als Vorbereitung eine Pokémon Klasse erstellt, um eine Vorlage für die Pokémons in der Datenbank zu haben. Zum Verbinden zur Datenbank haben wir die Mongodb-driver dependency genutzt, welche es uns erlaubt sich mit einer Mongo Datenbank zu verbinden.

String connectionString = "mongodb://root:1234@localhost:27017/?authSource=admin";

Mithilfe der Datenbank URI welche sowohl den Username als auch Password beinhaltete konnten wir uns erfolgreich mit dem MongoClient verbinden. Mit dem Mongoclient verbunden, wählten wir unsere Datenbank und unsere Pokémon collection.

// Connecting to database & get collection  
try (MongoClient mongoClient = MongoClients.*create*(connectionString)) {  
 MongoDatabase database = mongoClient.getDatabase("mongodb");  
 MongoCollection<Document> collection = database.getCollection("pokemons");

### Höchste oder tiefste Werte

Bei Jeder Frage, die in einem Quizablauf gestellt wird, hat 3 Werte, diese sind die Message an den Spieler, der Wert für den Gewinn und die Kategorie.

public List<Question> getStageQuestion() {

List<Question> list = new ArrayList();

list.add(new Question("Which Pokémon's evolution stage is the highest?", getDBValue(true, "stage"), "Stage"));

list.add(new Question("Which Pokémon's evolution stage is the lowest?", getDBValue(false, "stage"), "Stage"));

return list;

}

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungMan könnte für den Value bei min 0 einsetzten und max 999 oder eine höhere Zahl, aber im Falle eines neuen Pokémons der das übertrifft müsste man das auch im Code ändern. Deshalb wird hier der höchste/tiefste Wert direkt aus der Datenbank geholt, mit der Funktion getDBValue. Diese hat Aggregationen welche Pokémons nach dem Value der Kategorie sortiert. Und dann den höchsten oder je nach dem den tiefsten Wert herausliesst und zurückgibt.

### Datenbank durchlaufen

Um drei zufällige Pokémons zu erhalten, wird bei der Collection die Aggregation "$sample" verwendet. Mit dem Nutzen dieser Aggregation ist es möglich, die Anzahl der zufälligen Pokémons zu bestimmen.

var aggregation = Arrays.*asList*(Aggregates.*sample*(1));

Dieser Code wird drei Mal wiederholt, womit wir drei zufällige Pokémons erhalten.

Die Werte werden nun in ein BSON Document hinzugefügt. In unserem Fall die drei zufällige Pokémons. Zuerst wird ein "cursor" erstellt, welchem die drei Documents übergeben werden.

Um nicht 2 oder mehrere richtige Antworten zu haben, haben wir der Klasse Pokémon noch Points gegeben. Dies ist ein Attribut, das die Differenz ist von dem Wert der Frage und dem eigenem.

boolean isUnique = pokemons.stream().noneMatch(p -> p.getPoints() == currentPokemon.getPoints());

Wenn zwei Pokémons denselben wert oder gleich nach and den Wert sind, wird eins neu gewählt.

### Von BSON zu Java

Nun da man die Werte des Pokémons in einem BSON Document zur verfügung hat, kann man diese in ein Java Objekt umwandeln. Dazu wird in der Methode das Document als Parameter mitgegeben. Mit den Settern werden die Werte gesetzt und ein fertiges Pokémon Objekt ist bereit.

private Pokémon documentToPokemon(Document document) {  
 Pokémon pokemon = new Pokémon();  
 pokemon.setName(document.getString("name"));  
 pokemon.setHealth(document.getInteger("health"));  
 pokemon.setStrength(document.getInteger("strength"));  
 pokemon.setEnergy(document.getInteger("energy"));  
 pokemon.setStage(document.getInteger("stage"));  
 return pokemon;  
}

## Highscore

Nach den Anforderungen sollen am Schluss die besten Resultate angezeigt werden, wie man auch im Ablauf sehen kann. Dazu gibt es den Schritt, wo man ein Ablauf speichern muss und den Teil wo man die aus der DB abrufen muss, ähnlich wie beim ein lesen der Pokémon Objekte, aber hier braucht es extra Aggregationen.

### Quiz Logs

Nach jedem Spiel wird das Ergebnis geloggt und an die DB geschickt unter Statistics. Am Anfang des Spieles wird man nach einem Namen gefragt und dort wird auch die Zeit gestartet.

String name = scanner.nextLine();  
long start = System.*currentTimeMillis*();

Nach dem man mit dem Quiz fertig ist wird die Zeit gestoppt und die Punkte gespeichert und eine Methode mit diesen 3 werten ausgeführt.

long finish = System.*currentTimeMillis*();  
double time = (double) (finish - start) / 1000;  
sdb.saveWinLog(name, *points*, time);

Mit dieser Zeile code bestimmen wir, auf welcher DB unser Log gespeichert wird.

MongoClientSettings.*builder*().applyConnectionString(new ConnectionString("mongodb://root:1234@localhost:27017/?authSource=admin")).build()

Und in diesen Zeilen von code wie und wo darunter.

MongoDatabase database = mongoClient.getDatabase("statistics");  
try {  
 MongoCollection<Document> carDocs = database.getCollection("statistics");  
 Document doc = new Document();  
 doc.append("winner", winnerName);  
 doc.append("points", points);  
 doc.append("time", time);  
 carDocs.insertOne(doc);  
}

Wobei wir nur ein einfaches Json Objeckt erstellen mit 3 werten und die ID wird generieren.

### Top 5 Spieler

Ebenfalls für die Rangordnung haben wir Aggregationen genutzt. In diesem Fall $sort und $list welche die Datenbank Collection nach den Spielern mit den Meisten punkten sortiert und danach die ersten fünf ausgibt.

var aggregation = Arrays.*asList*(  
 Aggregates.*sort*(Sorts.*descending*("points")),  
 Aggregates.*limit*(5)  
);

## MongoDB Atlas

Um unser Projekt zu erweitern haben wir die Idee bekommen, die Datenbank online laufen zu lassen. Nach einiger Recherche konnten wir MongoDB Atlas ausfindig machen. Dort kann man seine eigene DB deployen. Der Preis variiert von der Speichergrösse und Leistung welche die Server aufbringen sollten. Doch sie bieten ebenfalls gibt es eine Möglichkeit eine Kleinere Datenbank welche für unseren Zweck gut genug ist Kostenlos zu erstellen.

Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

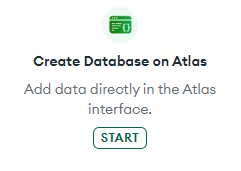
Man hat ebenfalls drei Anbieter(AWS, Google Cloud und Azure) und Standorte der Server zur Auswahl. Wir entschieden uns für AWS mit den Servern in Frankfurt was von uns am nächsten ist.

Als nächstes haben wir Benutzername(root) und Passwort(1234) sowie eine IP-Adresse(0.0.0.0/0) gesetzt.

Nun hat man ein Database Deployment. Durch einen "Connect" button kommt man auf dem Connection String für unser Java Projekt und MongoCompass.

Der letzte Schritt ist es die Datenbank zu befüllen. Dafür wählt man "Add Data" und "Create Database on Atlas". Dort kann man die Datenbank und Collection benennen. Danach kann man die Datenbank mit eigenen Werten füllen. Es ist ebenfalls möglich ein File mit den Werten zu importieren.

Ein Bild, das Text, Schrift, Screenshot, Logo enthält.

Automatisch generierte BeschreibungEin Bild, das Diagramm, Screenshot, Design enthält.

Automatisch generierte Beschreibung

## Source Code

<https://github.com/Erjon207/PokemonQuiz.git>