**M165 Pokémon Quiz Dokumentation**

Inhaltsverzeichnis

[Unser Thema 2](#_Toc155609131)

[Quiz Ablauf 2](#_Toc155609132)

[DB erstellen 4](#_Toc155609133)

[DB testen 5](#_Toc155609134)

[DB werte lesen in Java 7](#_Toc155609135)

[Datenbank durchlaufen 7](#_Toc155609136)

[Von BSON zu Java 7](#_Toc155609137)

[Highscore 8](#_Toc155609138)

[Quiz Logs 8](#_Toc155609139)

[Aggregationen 9](#_Toc155609140)

## 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.

## 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.

Wir machen mal ein Beispiel Lauf

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

Automatisch generierte Beschreibung

Wir geben einen Namen für das spiel ein.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

wir wählen eine Kategorie aus.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Wir bekommen jetzt diese 3 zufällige Pokémons zur Auswahl und wählen eines davon als Antwort aus.

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

Automatisch generierte Beschreibung

Der Quiz gibt uns die Antwort und fragt, ob wir aufhören wollen. Wenn wir mit ja antworten, endet der Quiz und der Name, Score und die Zeit werden in einer DB geloggt. Wenn wir nicht antworten, geht es weiter mit der nächsten Quiz Frage.

Mann 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 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.

Und unter Pokémons sieht man die Liste aller Pokémons die in der Daten bank vorhanden sind.

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

Automatisch generierte Beschreibung

## 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");

### 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 generiert.

### 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)  
);