



Protokoll zu Monster Trading Cards Game

Von Miriam Unger (if23b205)

2024/2025

Inhaltsangabe

Beschreibung	3
UML-Diagramm	4
Entscheidungen	5
GitHub-Link.....	8

Beschreibung

In diesem Projekt entwickle/entwickelte ich einen REST-basierten HTTP-Server für ein Monster-Trading-Card-Game (MTCG) in Java. Der Server ermöglicht es Nutzern, sich zu registrieren, sich anzumelden und grundlegende Aktionen im Zusammenhang mit dem Spiel durchzuführen.

Da dies „nur“ die Intermediate Submission ist, steht im Vordergrund die Benutzerregistrierung und Authentifizierung über entsprechende REST-API-Endpunkte zu implementieren. Der Server verarbeitet HTTP-Anfragen für das Anlegen von Benutzern und das Einloggen, indem er Input annimmt und eine Authentifizierung mit Token-System (-mtcg) bereitstellt und die die Datenbank speichert. Für diese Teilabgabe werden folgende Funktionen bereitgestellt:

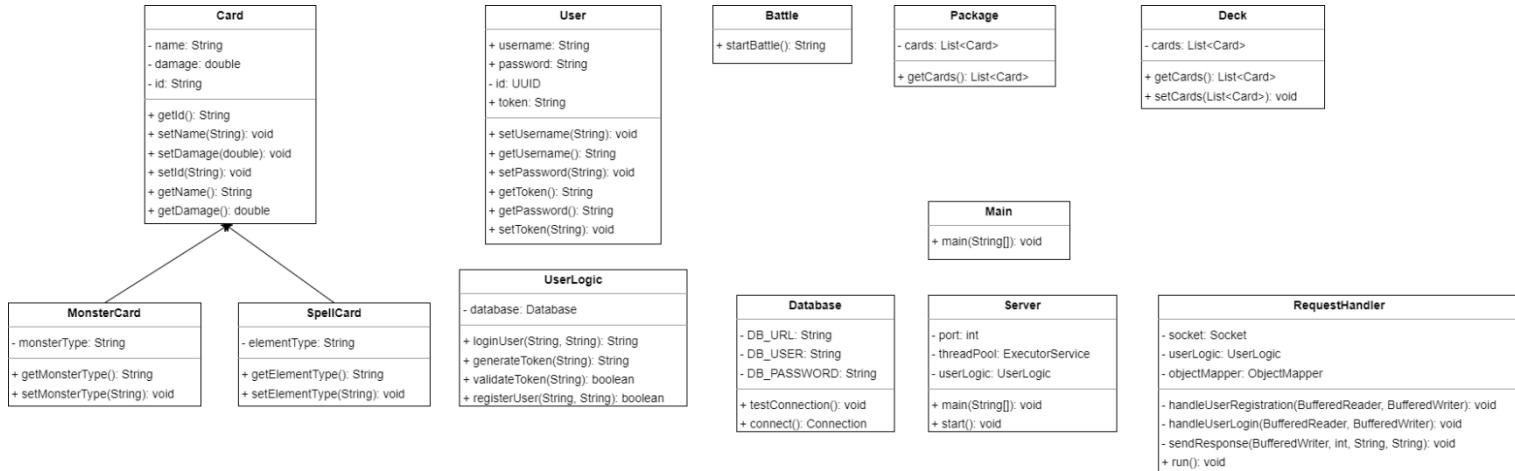
- Registrierung von Benutzern mit einzigartigen Benutzernamen und Passwörtern
- Anmeldung von Nutzern mit Rückgabe eines Authentifizierungstokens

UML-Diagramm

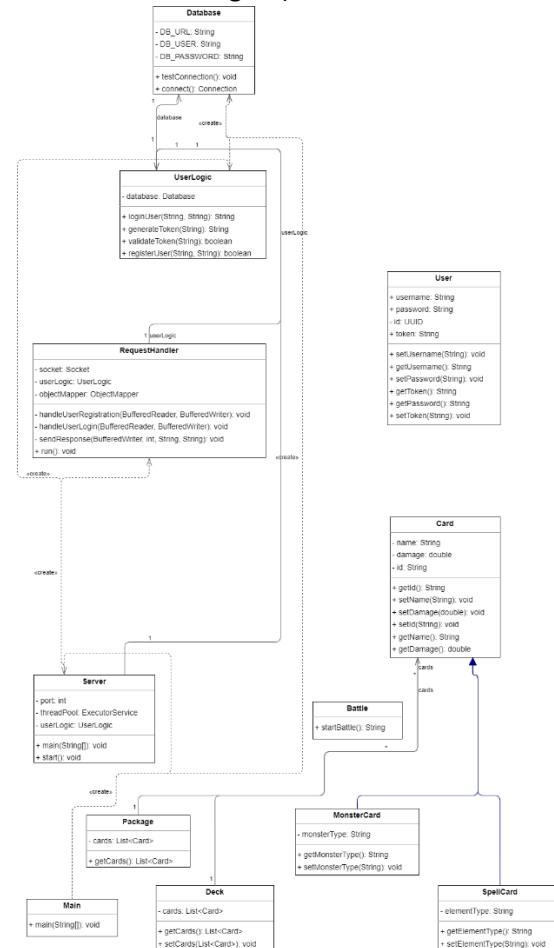
Das UML-Diagramm (Unified Modeling Language) bietet eine visuelle Darstellung der Struktur und Beziehungen der verschiedenen Klassen innerhalb der Anwendung.

Anmerkung: Ausgenommen User & UserLogic, sind die Methoden und Variablen Prototypen, die sich im Laufe der Erarbeitung des Programms noch ändern können.

UML ohne Beziehungen:



UML mit Beziehungen (ich hoffe man kann es gut genug lesen 😊) :



Entscheidungen

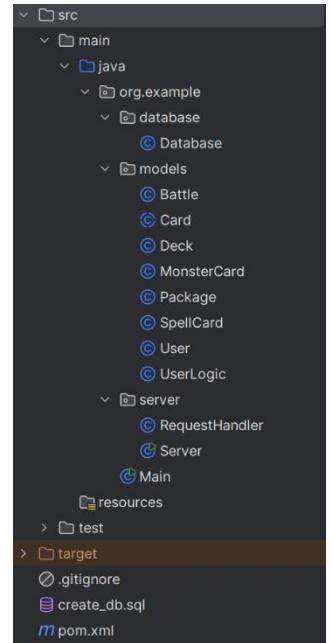
Während der Entwicklung des Projekts, trifft man tausend kleine Entscheidungen, aber den Verlauf der Arbeit stark beeinflussen. Im Folgenden ist eine kurze Übersicht einiger dieser Entscheidungen aufgeführt.

Ordnerstruktur/ Strukturierung des Quellcodes

Entscheidung:

Ich habe mich entschieden, die Pakete klar nach ihren Funktionen zu gliedern, indem ich separate Ordner für Datenbankzugriffe, Modelle und Serverlogik angelegt habe, nur ganz außerhalb befindet sich die Main.java.

Idee: Ich überlege die Files ...Logic.java in einen anderen Ordern zu geben, da mir da der Aufbau nicht ganz gefällt. Kurz vor der Abgabe hab ich mich aber nicht getraut hier Files zu verschieben.



Warum?

Ich habe dieses Konzept bei der ersten Präsentation eines Teiles des Projekts von einem Kollegen gesehen und dann wurde diese noch leicht mithilfe des Lektors modifiziert. Daraufhin, habe ich beschlossen, dass ich diese Struktur sehr inspirierend fand und es ähnlich in meinem Projekt umsetzen möchte.

Thread Pool für Parallelität

Entscheidung:

Ich habe mich für einen Threadpool entschieden statt einem Ansatz wie „Thread-per-Request“.

Warum?

Nach längeren recherchieren und mein Unwissen zu Beginn des Projekts habe ich mich schlussendlich dafür entschieden. Ich hoffe ich habe sie richtig implementiert, da dies meine erste Benutzung mit dieser Art von Threads ist. Mein Projekt kann dadurch mehrere Anfragen parallel verarbeiten, ohne dass es zu Leistungsengpässen kommt. Ob ich dies im Endeffekt brauche, wird sich im Verlauf des Projekts noch zeigen.

Keinen Docker

Entscheidung:

Ich habe im Vergleich zu manchen anderen Kollegen keinen Docker.

Warum?

Hierfür gibt es 2 Gründe:

1. Ich habe noch keinerlei Erfahrung damit gemacht und ich habe mich nicht getraut das gleich beim ersten großen Projekt mit Java einzubauen
2. Ein Kollege beschwert sich seit dem er den Docker hat, dass sein Laptop ständig heiß läuft und die Batterie sich schnell entlädt, das wollte ich nicht riskieren. Könnte aber daran liegen, dass er ihn nicht ausschaltet 😊.

Jackson für JSON

Entscheidung:

Für das Umwandeln von JSON-Daten in Java-Objekte und umgekehrt hast ich die Jackson-Bibliothek verwendet und ins pom.xml eingefügt.

Warum?

Nach kurzer Recherche, war dies eine performante und vor allem einfach umsetzbare Lösung zur Verarbeitung von JSON-Daten.

UUIDs

Entscheidung:

Ich habe mich entschieden, UUIDs (Universally Unique Identifiers) anstelle von Autoinkrement-IDs für die Primärschlüssel in meiner Datenbank zu verwenden.

Warum?

Diese Entscheidung ermöglicht mir eine globale Eindeutigkeit der IDs, wodurch Kollisionen vermieden werden. Der Hauptgrund dafür war aber die Erklärung

InputStreamReader & BufferedReader

Entscheidung:

Eingehende Daten vom Client werden durch InputStreamReader in Kombination mit BufferedReader gelesen.

Warum?

Auch hier wieder durch Recherche. Denn laut Google kann man dadurch die Daten als Reader verarbeiten, und dadurch kann man mit Zeichenfolgen (Strings) arbeiten, anstatt byteweise Daten zu verarbeiten.

HashMaps vor DB

Entscheidung:

In früheren Versionen des Codes habe ich kurzzeitig HashMaps verwendet, um Benutzerdaten temporär im Speicher zu halten, bevor ich die Datenbank aufgesetzt habe.

Warum?

Ich finde das HashMaps schnelle und einfache Datenstrukturen, um Benutzerinformationen zwischenspeichern. Zudem habe ich mit dieser Datenstruktur schon gearbeitet und weiß wie man sie verwendet.

GitHub-Link

Zu Beginn meiner Arbeit mit GitHub hatte ich ein paar Schwierigkeiten daher sind die ersten Commits möglicherweise etwas chaotisch und weniger strukturiert. Im Laufe des Projekts habe ich jedoch viel dazugelernt, und meine ich würde behaupten meine Machenschaften in GitHub hat sich deutlich verbessert. Gegen Ende hin sind die Commits übersichtlicher (meines Erachtens) und folgen einer klareren Struktur.

GitHub-Link: <https://github.com/mimiep/TradingCards>

