



Protokoll zu Monster Trading Cards Game – Final

Von Miriam Unger (if23b205)

2024/2025

Inhaltsangabe

Beschreibung	3
UML-Diagramm	4
Ordnerstruktur & Test	5
Feature	8
GitHub-Link.....	9
Was habe ich draus gelernt?	10
Investierte Zeit (geschätzt)	11

Beschreibung

In diesem Projekt entwickle/entwickelte ich einen REST-basierten HTTP-Server für ein Monster-Trading-Card-Game (MTCG) in Java. Der Server ermöglicht den Nutzern die Registrierung, Anmeldung, Verwaltung von Karten und Decks sowie das Kämpfen gegen andere Spieler.

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. Es werden verschiedene Spielmechaniken wie das Handhaben von Kartenpaketen, das Erstellen von Decks, das Ausführen von Kämpfen und die Berechnung von Benutzerstatistiken (z. B. ELO) unterstützt.

Für diese finale Abgabe werden folgende Funktionen bereitgestellt:

- Registrierung von Benutzern mit einzigartigen Benutzernamen und Passwörtern
- Anmeldung von Nutzern mit Rückgabe eines Authentifizierungstokens
- Kartenmanagement mit Erstellen und Verwalten von Karten und Erwerb von Packages
- Deckverwaltung mit den besten Karten eines Spielers
- Kämpfe zwischen Nutzern
- ELO und Scoreboard-System für die Ergebnisse nach Kämpfen

Der Server speichert alle relevanten Daten in einer Datenbank und ermöglicht es den Nutzern, ihre Spielstatistiken zu verfolgen.

UML-Diagramm

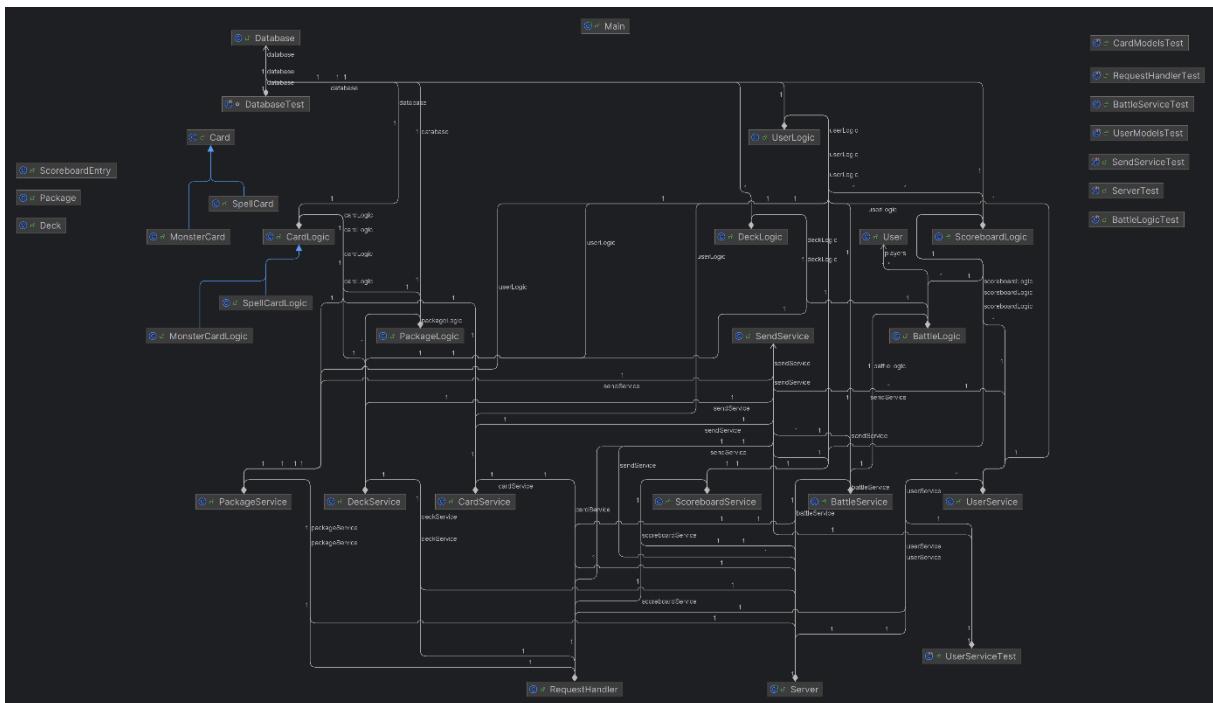
Das UML-Diagramm (Unified Modeling Language) bietet eine visuelle Darstellung der Struktur und Beziehungen der verschiedenen Klassen innerhalb der Anwendung.

UML ohne Beziehungen:

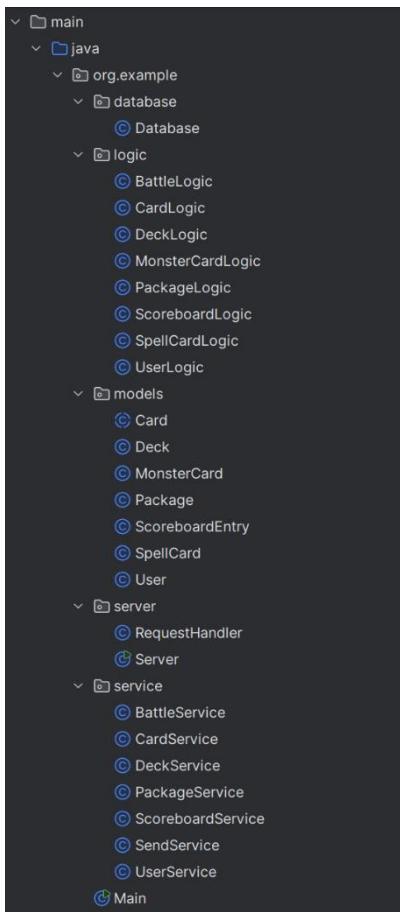


Informationen: Den RequestHandler habe ich noch weiter aufgeteilt in die einzelnen Services.

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

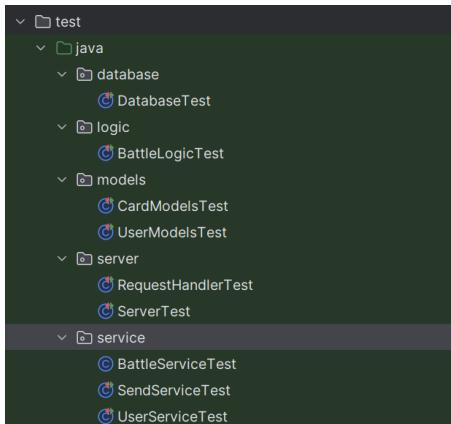


Ordnerstruktur & Test



Falls Sie noch von meiner Präsentation wissen, hatte ich einen relativ langen RequestHandler. Diese habe ich aber dann aufgeteilt in kleine Dateien, die jetzt als Service bekannt sind.

Testordner:



Liste der Tests: Tests passed

Wichtige Anmerkung: Ich hätte wirklich nicht gedacht das Tests schreiben so schwierig ist, aber als es dann plötzlich notwendig war für sinnvolle Test zu mocken, habe ich wirklich meine Schwierigkeiten gehabt. Ich hoffe dennoch, dass meine Tests passen 😊.

Zudem bin ich der Meinung das das CURL auch schon ein super Test. Des weiteren, wenn man das gut kann hat man dann 10000 Test, was vermutlich das sehr Realitätsnah wäre, bzw. man hätte die Test schon vor der Implementierung wie wir es in TDD gelernt haben.

- testConnect_Success
 - o Beschreibung: Testet ob connect() Methode eine erfolgreiche Verbindung zu DB herstellt.
 - o Warum: Überprüfung ob DB korrekt ist.
- testConnect-Failure
 - o Beschreibung: Simuliert Szenario sodass connect() fehlschlägt.
 - o Warum: Überprüfung ob System mit Fehlschlägen richtig umgeht.
- testTestConnection
 - o Beschreibung: Überprüft ob testConnection() Methode keine Ausnahme wirft
 - o Warum: Verbindung testen und schauen ob Datenbank betriebsbereit ist.
- testAddPlayer
 - o Beschreibung: Schauen ob Spieler korrekt zu BattleLogic hinzugefügt wird
- testRoundDamage
 - o Beschreibung: Überprüfung der Schadensberechnung zwischen 2 Karten.
- testWizzardVsOrk
 - o Beschreibung: Überprüfung Sonderkategorie bei Kampf.
- testKnightVsWaterSpell
 - o Beschreibung: Überprüfung Sonderkategorie bei Kampf.
- testKrakenVsSpells
 - o Beschreibung: Überprüfung Sonderkategorie bei Kampf.
- testFireElfVsDragon
 - o Beschreibung: Überprüfung Sonderkategorie bei Kampf.
- testWaterVsFire
 - o Beschreibung: Überprüfung Sonderkategorie bei Kampf.
- testNormalVsWater
 - o Beschreibung: Überprüfung Sonderkategorie bei Kampf.

Warum: Schauen ob die BattleLogic bei den Ausnahmen der Karten, die richtigen Damages ausrechnet.

- testCardCreation
 - o Beschreibung: Überprüfung der Erstellung einer Karte
 - o Warum: Schauen ob korrekt instanziert wurde.
- testSpellCardInheritance
 - o Beschreibung: Überprüfung der Erstellung einer Karte.
 - o Warum: Schauen ob die Vererbung funktioniert.
- testDifferentCardInstance
 - o Beschreibung: Vergleich von 2 verschiedenen Instanzen.
 - o Warum: Schauen ob jede Karte eindeutig und separat verwaltet wird.
- testCardTypeConsistency
 - o Beschreibung: Konsistenz von Spell und Monster prüfen.
 - o Warum: Um zu schauen ob Logik zur Unterscheidung fehlerfrei arbeitet.

- testDifferentCardTypes
 - o Beschreibung: Vergleich von 2 verschiedenen Instanzen.
 - o Warum: Schauen ob jede Karte eindeutig und einzigartig ist.

- testUserConstructorAndGetters
 - o Beschreibung: Testet Funktion von Konstruktor.
 - o Warum: Schauen ob die Objekte korrekt initialisiert wurden.

- testSetters
 - o Beschreibung: Testet Setter.
 - o Warum: Schauen ob die Objekte korrekt initialisiert wurden und geändert werden können.

- testEmptyUser
 - o Beschreibung: Testet Funktion von leerem Konstruktor.
 - o Warum: Schauen ob die Objekte korrekt initialisiert wurden.

- testSetCoins
 - o Beschreibung: Setzen der Attribute.
 - o Warum: Schauen ob die Attribute korrekt initialisiert wurden.

- testDefaultUser
 - o Beschreibung: Testet Funktion von leerem Feldern beim Erstellen.
 - o Warum: Schauen ob alles passt

- testMethodNotAllowed
 - o Beschreibung: Simuliert eine HTTP-Anfrage mit einer Methode, die nicht erlaubt ist
 - o Warum: Überprüft, ob der Server korrekt mit dem HTTP-Statuscode antwortet

- testServerStartsCorrectly
 - o Beschreibung: Überprüft, ob der Server erfolgreich startet
 - o Warum: Sicherstellen, dass Serveroperationen mit Binden an Port und Starten von run() auch funktioniert.

- testSendResponse
 - o Beschreibung: Überprüft, ob sendResponse() die Korrekte http Antwort liefert.
 - o Warum: siehe Beschreibung

- testHandleUserRegistration
 - o Beschreibung: Überprüft handleUserRegistration () .
 - o Warum: Schauen ob Registrierungsanfrage korrekt verarbeitet wird. Wird in DB geschrieben.

Feature

Ich hoffe das das als Feature gilt, aber ich hab mir gedacht, ich möchte nicht irgendeinen Codeteil ändern damit die Spielregeln ein bisschen anders sind, ich hab mich deswegen für ASCII Art im Programm entschieden.

Beispiele:

Beim Starten von Battle:

```
{"message":"One Player."}  
0  
/|\br/> / \  
HTTP/1.1 201 OK  
Content-Type: application/json  
Content-Length: 73  
  
{"message":"Two Player."}  
0 0  
/|\ /|\br/> / \ / \
```

Beim BattleLog:

```
It's a draw! No cards are exchanged.  
+-----+  
| |  
| ♠ ♣ |  
| DRAW |  
| ♥ ♦ |  
+-----+
```

```
kienboec's card wins!  
+-----+  
| |  
| ♠ ♣ |  
| CHANGE |  
| ♥ ♦ |  
+-----+
```

Battle finished after 13 rounds. Winner: altenhof

```
-----  
| |  
| 🏆 |  
| -----|
```

```
-----  
| ROUND 13 |  
| -----|
```

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>



Was habe ich draus gelernt?

Da gibt es gleich mehrere Dinge:

- Also am Anfang hab ich mir mit Java und allem sehr schwer getan, vor allem war ich sehr überfordert von der Angabe weil, es Dinge waren die ich nicht mal mit einer Programmiersprache die ich kann machen konnte, aber im Laufe des Projekt und in den Stunden wurde es immer besser. Am Ende konnte ich schon manche Probleme ohne nur nachzudenken einfach beheben weil ich sofort wusste was das bedeutet
- Das was man glaubt was am meisten Zeit verbraucht, geht relativ schnell, das was man glaubt das schnell geht braucht ewig. Dies betrifft die BattleLogic und das Change User Stats. Das eine ging ur schnell aber beim anderen hab ich gekämpft, weil man ur viel von dem was man schon hatte ändern muss.
- Ich bin definitiv kein Fan von Tests. Damals in TDD habe ich es geliebt die Tests zu schreiben. Aber jetzt habe ich einfach nur gelitten, da wir die meisten Dinge noch nicht kannten und es sehr schwer war sinnvolle Tests zu schreiben mit den mitteln die wir haben.
- Von Refactoring kann man definitiv nie genug bekommen. Ich habe schon sehr viel geändert und verbessert, aber je mehr Zeit ich damit verwende desto mehr fällt mir auf was ich noch refactoren könnte.
- Am Anfang habe ich das CURL nicht gemocht, ich habe damit nämlich noch nie damit gearbeitet, aber je mehr man es verwendet desto mehr hab ich es lieben gelernt weil mal damit sehr gut die nächsten Sachen implementieren konnte.
- Auch IntelliJ habe ich zu lieben gelernt, obwohl ich manchmal kein Fan seiner Fehlermeldungen war, und ich manchmal Stunden gebraucht habe manche Fehler zu Suchen kenne ich mich jetzt super aus und weiß wie man es optimal einsetzt um schnell voran zu kommen.
- Es ist leichter viele Tage hintereinander zu Arbeiten, als immer wieder finde ich. Denn wenn man alles auf einmal macht, dann braucht man sich nicht immer neu reinlesen.

Investierte Zeit (geschätzt)

Vor der Intermediate Abgabe habe ich leider nicht protokolliert wie viel Zeit ich investiert habe, aber dafür dann danach. Ich muss aber leider zugeben das es immer wieder lange Lücken aufgrund von vielen Prüfungen in der FH gibt. Dafür möchte ich mich entschuldigen 😞.

Dies sind rein geschätzte Zeiten: (die tatsächlichen Zeiten können stark davon abweichen)

November 2024

Montag	Dienstag	Mittwoch	Donnerstag	Freitag	Samstag	Sonntag
28	29	30	31	1	2	3
				4 h	2 h	
4	5	6	7	8	9	10
11	12	13	14	15	16	17
2 h				3 h	1 h	
18	19	20	21	22	23	24
4 h	3 h					
25	26	27	28	29	30	1
3 h				4 h		

iKalender.org

Dezember 2024

Montag	Dienstag	Mittwoch	Donnerstag	Freitag	Samstag	Sonntag
25	26	27	28	29	30	1
				2 h		2 h
2	3	4	5	6	7	8
3 h			2 h	6 h		
9	10	11	12	13	14	15
4 h				3 h	8 h	
16	17	18	19	20	21	22
6 h				1 h	4 h	7 h
23	24	25	26	27	28	29
30	31	1	2	3	4	5

iKalender.org

Januar 2025

Montag	Dienstag	Mittwoch	Donnerstag	Freitag	Samstag	Sonntag
30	31	1	2	3	4	5
				6 h	7 h	7 h
6	7	8	9	10	11	12
9 h	8 h	10 h				
13	14	15	16	Präsi		
20	21	22	23	24	25	26
27	28	29	30	31	1	2

iKalender.org