MTCG: Intermediate Protokoll

Rentenberger Lorenz

27. Januar 2025

1 Projektübersicht

Das Monster Trading Card Game (MTCG) ist eine REST-API Implementierung eines Kartenspiels. Die Anwendung ermöglicht Benutzern, sich zu registrieren, Karten zu sammeln, Decks zusammenzustellen und gegen andere Spieler anzutreten.

1.1 GIT

Link zum Github-Repository: `https://github.com/LRenTi/BIF3_SWEN1`

2 Unique Features

Winrate wird berechnet und angezeigt im Scoreboard. Die Winrate eines Benutzers wird im Scoreboard angezeigt und gibt das Verhältnis der gewonnenen zu den verlorenen Spielen an. Sie wird berechnet, indem die Anzahl der gewonnenen Spiele durch die Anzahl der verlorenen Spiele geteilt wird:

$$\text{Winrate} = \frac{\text{Wins}}{\text{Losses}}$$

Diese Kennzahl bietet einen schnellen Überblick über die Leistung eines Benutzers im Vergleich zu anderen Spielern. Eine höhere Winrate deutet auf eine bessere Erfolgsquote in den Kämpfen hin.

3 Architektur und Design

Die Architektur der Anwendung basiert auf einem mehrschichtigen Ansatz. Die API-Schicht (MTCG.Api) übernimmt die Bearbeitung von HTTP-Requests und das Routing der Endpunkte. Die Core-Schicht (MTCG.Core) enthält die Geschäftslogik und alle zentralen Entitäten des Spiels. Die Data-Schicht (MTCG.Data) enthält die Datenbank und die Repositorys. Diese Trennung ermöglicht eine klare Struktur und erleichtert zukünftige Erweiterungen.

4 Battle-Logik im BattleHandler

Der `BattleHandler` ist verantwortlich für die Durchführung von Kämpfen zwischen zwei Benutzern in der Anwendung. Die Hauptkomponenten der Battle-Logik sind wie folgt:

- **StartBattle:** Diese Methode initiiert einen Kampf. Wenn ein Benutzer einen Kampf starten möchte, wird überprüft, ob ein anderer Benutzer bereits in der Warteschlange ist. Falls nicht, wird der Benutzer in die Warteschlange gesetzt. Andernfalls wird der Kampf zwischen dem wartenden Benutzer und dem neuen Herausforderer gestartet.
- **DoBattle:** Diese Methode führt den eigentlichen Kampf durch. Der Kampf besteht aus maximal 100 Runden, in denen Karten der beiden Benutzer gegeneinander antreten. Die Karten werden zufällig ausgewählt, und der Schaden wird basierend auf den Karteneigenschaften und Elementen berechnet.
- **Spezielle Interaktionen:** Bestimmte Karteninteraktionen haben spezielle Regeln, z.B.:
  - `Goblin` verliert immer gegen `Dragon`.
  - `Wizard` gewinnt immer gegen `Ork`.
  - `Knight` verliert gegen `Water Spell`.

- `Kraken` ist immun gegen Zaubersprüche.
- `Fire Elves` können `Dragon` ausweichen.

- **Kampfergebnis:** Der Benutzer, der die meisten Runden gewinnt, wird zum Sieger erklärt. Der Sieger erhält eine Erhöhung seines Elo-Ratings und seiner Siegstatistik, während der Verlierer eine Verringerung seines Elo-Ratings erfährt.

Diese Logik stellt sicher, dass die Kämpfe dynamisch und strategisch sind, indem sie sowohl auf Zufall als auch auf spezifische Karteninteraktionen setzen.

5 API Endpunkte

Methode	**Beschreibung**
GET /cards	Ruft alle verfügbaren Karten ab.
POST /cards	Fügt eine neue Karte hinzu.
GET /cards/me	Ruft die eigenen Karten ab.
POST /deck	Fügt eine Karte zu einem Deck hinzu.
DELETE /deck	Entfernt eine Karte aus einem Deck.
GET /deck	Ruft das Deck eines Benutzers ab.
PUT /deck	Definiert das Deck eines Benutzers.
GET /battles	Startet einen Kampf zwischen zwei Benutzern.
GET /scoreboard	Ruft das Scoreboard mit den Benutzerstatistiken ab.
POST /market	Erstellt ein neues Angebot auf dem Markt.
GET /market	Ruft alle Marktangebote ab.
PUT /market/{offerId}	Aktualisiert ein Marktangebot mit der angegebenen ID.
POST /purchase/{packageId}	Kauft ein Kartenpaket mit der angegebenen ID.
GET /package	Ruft alle verfügbaren Kartenpakete ab.
POST /package	Erstellt ein neues Kartenpaket.
POST /sessions	Erstellt eine neue Benutzersitzung (Login).
GET /users/{username}	Ruft die Informationen eines bestimmten Benutzers ab.
POST /users	Erstellt einen neuen Benutzer.

Tabelle 1: API-Endpunkte

6 Begründung der Testfälle

Die Implementierung von Unit-Tests in der Softwareentwicklung ist entscheidend, um die Qualität und Zuverlässigkeit des Codes sicherzustellen. In diesem Projekt wurden die Tests aus mehreren Gründen eingesetzt:

- **Verifikation der Funktionalität:** Die Tests überprüfen, ob die grundlegenden Funktionen der Anwendung, wie die Erstellung und Authentifizierung von Benutzern, korrekt implementiert sind. Dies stellt sicher, dass die Kernfunktionen wie erwartet arbeiten.
- **Sicherstellung der Geschäftslogik:** Die Tests validieren die Geschäftslogik, insbesondere die Kampfmechanismen und die Berechnung von Spielstatistiken wie Elo-Ratings und Winrates. Dies ist wichtig, um die Integrität der Spielregeln zu gewährleisten.
- **Erkennung von Regressionen:** Durch das regelmäßige Ausführen der Tests kann sichergestellt werden, dass neue Änderungen oder Erweiterungen im Code keine bestehenden Funktionen beeinträchtigen. Dies hilft, unerwünschte Seiteneffekte zu vermeiden.
- **Dokumentation des Verhaltens:** Die Tests dienen als lebendige Dokumentation des erwarteten Verhaltens der Anwendung. Sie bieten eine klare Spezifikation dessen, was der Code leisten soll, und können als Referenz für zukünftige Entwicklungen genutzt werden.
- **Erhöhung der Codequalität:** Durch das Schreiben von Tests wird der Entwickler dazu angeregt, den Code modularer und wartbarer zu gestalten. Dies führt zu einer besseren Struktur und Lesbarkeit des Codes.

Insgesamt tragen die Tests dazu bei, die Zuverlässigkeit und Stabilität der Anwendung zu erhöhen, indem sie sicherstellen, dass alle Komponenten wie vorgesehen funktionieren und dass Änderungen im Code keine negativen Auswirkungen haben.

7 Lessons Learned

Im Verlauf dieses Projekts habe ich wertvolle Erfahrungen und Kenntnisse in verschiedenen Bereichen der Softwareentwicklung gesammelt:

- **Verwendung eines HTTP-Servers:** Ich habe gelernt, wie man einen HTTP-Server einrichtet und konfiguriert, um Anfragen zu empfangen und zu verarbeiten. Dies beinhaltete das Verständnis der HTTP-Protokolle und der effizienten Verwaltung von Anfragen und Antworten.
- **Erstellung der API und der Endpunkte:** Die Entwicklung einer RESTful API war eine zentrale Komponente des Projekts. Ich habe gelernt, wie man Endpunkte definiert, um verschiedene Funktionen der Anwendung bereitzustellen, und wie man diese Endpunkte testet und dokumentiert.
- **Verwendung von PostgreSQL:** Die Integration von PostgreSQL als Datenbanklösung hat mir Einblicke in die Verwaltung und Abfrage von Datenbanken gegeben. Ich habe gelernt, wie man Datenbanktabellen erstellt, Daten speichert und abruft sowie wie man SQL-Abfragen optimiert.

Diese Erfahrungen haben mein Verständnis für die Entwicklung moderner Webanwendungen vertieft und mir geholfen, meine Fähigkeiten in der Backend-Entwicklung zu erweitern.

8 Zeitaufwand

Im Rahmen dieses Projekts habe ich die folgenden Zeitaufwände verzeichnet:

- **Projektentwicklung:** Die Entwicklung des gesamten Projekts hat insgesamt etwa 50 Stunden in Anspruch genommen. Dies umfasste die Planung, Implementierung und das Testen der Anwendung.
- **Dokumentation:** Für die Erstellung der Projektdokumentation habe ich etwa 2 Stunden benötigt. Dies beinhaltete das Schreiben und Formatieren der Dokumentation in LaTeX.
- **Curl-Skript:** Die Erstellung und das Testen des Curl-Skripts zur Automatisierung von API-Anfragen haben weitere 1,5 Stunden in Anspruch genommen.

Somit beträgt der Gesamtzeitaufwand etwa 53,5 Stunden.