**Protokoll: Monster Trading Cards Game (MCTG)**

**1. App Design Overview**

Das Monster Trading Cards Game ist eine serverbasierte Anwendung, die Benutzerinteraktionen über Controller und die zugrundeliegende Logik in Services verwaltet. Kernentitäten wie Karten, Decks und Schlachten werden modelliert. Die Anwendung verwendet einen benutzerdefinierten HTTP-Server, der Anfragen von Clients verarbeitet und Antworten generiert.

**2. Hauptdesignentscheidungen**

**2.1 Benutzerdefinierter HTTP-Server**

Die Anwendung implementiert einen eigenen HTTP-Server, der von einer Basisklasse abgeleitet ist. Der Server ist modular aufgebaut und gliedert sich in Anfrage- und Antwortverarbeitung, Routing und Dienstprogramme. Diese Implementierung bietet Flexibilität und vollständige Kontrolle über die Serverfunktionalität.

**2.2 MVC-Architektur**

Die Anwendung folgt einem MVC-ähnlichen Ansatz:

* **Controller:** Verarbeiten HTTP-Anfragen und leiten sie an die entsprechenden Services weiter.
* **Services:** Enthalten die Geschäftslogik, wie z. B. die Verwaltung von Benutzern, Karten und spielbezogenen Vorgängen.
* **Modelle:** Repräsentieren die Kernentitäten wie Benutzer, Karten und Decks.

**2.3 Klassendiagramm**

Das Klassendiagramm beschreibt die Beziehung der wichtigsten Klassen:

* **Main:** Initialisiert die Anwendung und erstellt Kernobjekte wie Karten und Benutzer.
* **Controller und Service Klassen:** Verantwortlich für die Verarbeitung spezifischer HTTP-Anfragen.
* **UserService, BattleArena:** Implementieren die Geschäftslogik für Benutzer- und Spielinteraktionen.
* **Card, MonsterCard, SpellCard:** Modellieren die Entitäten des Spiels.

**3. Lessons Learned**

Während der Entwicklung des Projekts wurden folgende Erkenntnisse gewonnen:

* **Klarheit der Anforderungen:** Die genaue Spezifikation war entscheidend, um komplexe Logiken wie Schlachten und Handelsfunktionen zu implementieren.
* **Testbarkeit:** Die Strukturierung des Codes in klar abgegrenzte Controller und Services erleichterte die Implementierung von Unit-Tests.
* **Fehlerbehebung:** Der Debugging-Prozess für die benutzerdefinierte HTTP-Server-Implementierung war anfangs zeitaufwändig, führte aber zu einem tieferen Verständnis der Protokolle.

**4. REST-API-Endpunkte**

| **HTTP-Methode** | **Endpoint** | **Tag** | **Beschreibung** |
| --- | --- | --- | --- |
| **POST** | **/users** | **users** | **Registriert einen neuen Benutzer mit Benutzername und Passwort.** |
| **GET** | **/users/{username}** | **users** | **Ruft die Daten eines Benutzers basierend auf seinem Benutzernamen ab. Nur Admin oder Benutzer selbst erlaubt.** |
| **PUT** | **/users/{username}** | **users** | **Aktualisiert die Benutzerdaten für den angegebenen Benutzernamen. Nur Admin oder Benutzer selbst erlaubt.** |
| **POST** | **/sessions** | **users** | **Loggt einen Benutzer mit Benutzernamen und Passwort ein.** |
| **POST** | **/packages** | **packages** | **Erstellt ein neues Paket (nur Admin).** |
| **POST** | **/transactions/packages** | **packages** | **Erwirbt ein Kartenpaket mit dem Geld des angegebenen Benutzers.** |
| **GET** | **/cards** | **cards** | **Gibt alle Karten des Benutzers zurück.** |
| **GET** | **/deck** | **cards** | **Zeigt das aktuell konfigurierte Deck des Benutzers an.** |
| **PUT** | **/deck** | **cards** | **Konfiguriert ein Deck mit vier angegebenen Karten.** |
| **GET** | **/stats** | **game** | **Ruft die Statistiken eines einzelnen Benutzers ab.** |
| **GET** | **/scoreboard** | **game** | **Gibt das Scoreboard geordnet nach ELO zurück.** |
| **POST** | **/battles** | **game** | **Tritt einer Lobby bei, um einen Kampf zu starten.** |
| **GET** | **/tradings** | **trading** | **Ruft verfügbare Handelsangebote ab.** |
| **POST** | **/tradings** | **trading** | **Erstellt ein neues Handelsangebot.** |
| **DELETE** | **/tradings/{tradingdealid}** | **trading** | **Löscht ein bestehendes Handelsangebot.** |
| **POST** | **/tradings/{tradingdealid}** | **trading** | **Führt einen Handel für ein Angebot mit einer angegebenen Karte aus.** |

**5. Implementierung der BattleLogik**

**5.1 Grundlegender Ablauf**

* Karten werden pro Runde zufällig aus den Decks beider Spieler ausgewählt.
* Die Runde endet, wenn eine Karte die andere besiegt (basierend auf Schaden und Element).
* Besiegte Karten werden dem Siegerdeck hinzugefügt.
* Bei einem Unentschieden bleiben die Karten unverändert.

**5.2 Besonderheiten**

* **Goblins vs. Drachen:** Goblins greifen keine Drachen an.
* **Zauberer vs. Orks:** Zauberer kontrollieren Orks und verhindern Schaden.
* **Ritter vs. Wasserzauber:** Ritter ertrinken durch Wasserzauber.
* **Kraken:** Immun gegen Zauber.
* **Feuerelfen vs. Drachen:** Feuerelfen weichen Drachenangriffen aus.

**5.3 Score-System**

* **Startwert:** 100 Score.
* **Gewinn:** +3 Punkte.
* **Niederlage:** -5 Punkte.
* **Unentschieden:** Keine Änderung.
* Erweiterte Berechnungen für differenzierte Bewertungen sind in der Implementierung möglich.

**6. Tradings**

* Benutzer können Karten aus ihrem Stapel in den Handelsmarkt einstellen.
* Handelsanforderungen beinhalten Typen (Zauber oder Monster) und Mindestschaden.
* Akzeptierte Handelsangebote führen zum direkten Tausch der Karten.

**7. Einzigartiges Feature**

Ein "Critical-Hit "-Feature wurde hinzugefügt, bei dem jede Karte eine Chance von 20 % hat, den Schaden verdoppeln zu können:

* Karten können einen Schadensbonus von 100 % erhalten.
* Das Power-Up hat bei jeder Karte eine 20 % Chance zufällig angewendet zu werden.

**8. Datenpersistenz**

Die Anwendung verwendet eine PostgreSQL-Datenbank:

* Tabellen: users, cards, trades, battles und Verbindungstabellen (z.b.package\_cards).
* Benutzer-Token und Sitzungsdaten werden sicher gespeichert, um Aktionen zu authentifizieren.

**9. Tests**

**9.1 Unit-Tests**

* **BattleLogik:** Verifizierung der korrekten Schadensberechnung und Sicherstellung der korrekten Implementierung der Spezialfälle.

**9.2 Integrationstests**

Ein CURL-Skript testet die REST-API mit definierten Anforderungen und überprüft alle wichtigen Funktionen:

* Benutzerregistrierung und -anmeldung.
* Erstellung und Abruf von Karten.
* Durchführung von Kämpfen.

**10. Zeitmanagement**

| **Aufgabe** | **Zeitaufwand** |
| --- | --- |
| Implementierung des HTTP-Servers | 10 Stunden |
| API-Design und -Implementierung | 15 Stunden |
| Schlachtlogik | 12 Stunden |
| Tests | 8 Stunden |
| Dokumentation | 5 Stunden |
| **Gesamtzeit** | **50 Stunden** |

**11. GIT-Repository**

Der Quellcode der Anwendung ist in folgendem Repository verfügbar: <https://github.com/JoelK27/Monster_Card_Game>