Dokumentation zum App-Design des Monster Trading Card Games

Übersicht

Diese Dokumentation beschreibt das Design, die Kernkonzepte und die Lessons Learned während der Entwicklung eines Monster Trading Card Games. Ziel ist es, einen Überblick über die Architektur, die getroffenen Entscheidungen und die einzigartigen Funktionen der App zu geben.

Architektur der App

Die Architektur basiert auf einer klaren Trennung von Verantwortlichkeiten:

Services

- **Beschreibung**: Die Services sind das Herzstück der Geschäftslogik. Hier werden komplexe Operationen ausgeführt, die mehrere Datenquellen oder Geschäftsregeln betreffen.
- **Beispiele**:
    - **AuthenticationService**: Zuständig für das Authentifizieren und Registrieren von Spielern und Admins
    - **TradingService**: Behandelt die Trading-Angebote und die Annahmen von Karten

Repositories

- **Beschreibung**: Repositories sind für den Zugriff auf die Datenbank oder andere persistente Speicher zuständig. Sie abstrahieren die Datenquelle, um eine einfache Wiederverwendbarkeit und Testbarkeit zu gewährleisten.
- **Beispiele**:
    - **UserRepository**: Verwaltet Spielerprofile und -statistiken.
    - **CardRepository**: Speichert und verwaltet die Kartendaten.

Models

- **Beschreibung**: Die Models stellen die Entitäten der Applikation dar.
- **Beispiele**:
    - **User**: Stellt einen Benutzer dar und besteht aus weiteren anderen Model z.B. UserCredentials oder UserStats
    - **Deck**: Stellt eine Sammlung von Karten dar, mit der ein Spieler gegen andere Spieler antreten kann

Handler

- **Beschreibung**: Um das Hinzufügen von neuen API-Endpoints zu ermöglichen und die die Abhandlung von Requests und deren URL effektiver und schneller zu machen, wurde die Handle-Klasse mit der Handle-Methode implementiert, die je nach Route andere HandleMethoden der erbenden Klassen aufruft. Mögliche Routen sind:
    - **Final Routes**: Das sind Routes, die mit einer bestimmten Pfadsegment enden und keine weitere unterliegenden Pfadsegmente hat, z.B. ein POST auf /api/packages

- **Unfinal Routes:** Das sind Routes, die ein unterliegendes Pfadsegment benötigen und demnach auch einen eigenen Handler z.B. /api
- **Variable Routes:** Das sind Routes, die innerhalb des Pfades ein variables Pfadsegment hat, z.B. /api/users/{username}

Lessons Learned

Ich habe mich bewusst dagegen entschieden, die im Moodle-Kurs bereitgestellte Version (abgesehen vom HttpServer) zu verwenden, um selbst auf Schwierigkeiten und Verbesserungspotenziale zu stoßen. In meiner Version gibt es keine Möglichkeit, innerhalb des Servers bestimmte „Datensätze“ für eine bestimmte Zeit zu sperren (z. B. durch Mutexes oder ähnliche Mechanismen). In der Moodle-Version scheint dies durch die Implementierung von **IAtom** gelöst zu sein. Mir ist bewusst, dass meine Lösung nicht optimal ist. Allerdings habe ich durch das eigenständige Experimentieren und Lösen von Problemen mehr gelernt.
Weitere mögliche Verbesserungen wären eine vollständige Transaktionssicherheit sowie die Reduktion der Methoden in den Repositories auf die wesentlichen CRUD-Funktionalitäten.

Entscheidungen zu Unit-Tests

- **Was wird getestet?**
  - Datenzugriffslogik in den Repositories.

Unique Feature

Das Unique Feature ist die Möglichkeit, ein Trading Offer mit einem **Automatic Accept** zu kennzeichnen. Beim Erstellen eines Offers wird sofort geprüft, ob es bereits ein anderes Offer gibt, dessen Anforderungen von der angebotenen Karte erfüllt werden und ob die Karte den eigenen Anforderungen entspricht. Falls dies der Fall ist, wird kein neues Trading Offer erstellt, sondern das bereits bestehende Angebot direkt akzeptiert.

GitHub-Repository:

[paulisfaul/MTCG: Monster Trading Cards Game - SWEN]

Ich hatte Probleme beim Committen und Pushen, ich musste daher die Dateien manuell hochladen.