MTCG – Projektprotokoll

Design und Struktur

- **MVC Pattern** für klare Trennung der Komponenten
    - Models (User, Card, Deck, etc.)
    - Controller (UserController, CardController, etc.)
    - Repository Layer für Datenbankzugriffe
- **Repository Pattern** für Datenbankzugriffe
- **RESTful API** Design nach OpenAPI Spezifikation

Tabellenübersicht

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(255) UNIQUE,
  password VARCHAR(255),
  coins INTEGER DEFAULT 20,
  elo INTEGER DEFAULT 100,
  wins INTEGER DEFAULT 0,
  losses INTEGER DEFAULT 0,
  name VARCHAR(255),
  bio TEXT,
  image VARCHAR(255)
);

CREATE TABLE packages (
  id SERIAL PRIMARY KEY,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  acquired BOOLEAN DEFAULT FALSE
);

CREATE TABLE cards (
  id UUID PRIMARY KEY,
```

```
    name VARCHAR(255),
    damage FLOAT,
    element_type VARCHAR(50),
    card_type VARCHAR(50),
    package_id INTEGER REFERENCES packages(id)
);

CREATE TABLE user_cards (
    user_id INTEGER REFERENCES users(id),
    card_id UUID REFERENCES cards(id),
    PRIMARY KEY (user_id, card_id)
);

CREATE TABLE decks (
    user_id INTEGER REFERENCES users(id),
    card_id UUID REFERENCES cards(id),
    PRIMARY KEY (user_id, card_id)
);

CREATE TABLE trading_deals (
    id UUID PRIMARY KEY,
    card_to_trade UUID REFERENCES cards(id),
    user_id INTEGER REFERENCES users(id),
    type VARCHAR(50),
    minimum_damage FLOAT
);
```

Lessons Learned

Technische Herausforderungen

1. **HTTP Server Implementation**
    - Schwierigkeiten bei der Thread-Synchronisation

- Korrekte Request/Response-Verarbeitung
- Token-basierte Authentifizierung

2. **Datenbankdesign**
    - Komplexe Beziehungen zwischen Tabellen
    - Performance-Optimierung bei Kartenoperationen
    - Package-Management
3. **Battle System**
    - Implementierung der unique regeln
    - Element-Typ Effektivität
    - Zufällige Kartenauswahl
4. **Battle-System Implementation**
    - Schwierigkeiten bei der Thread-Synchronisation
    - Komplexität der Element-Typ Effektivität
    - applyBooster um ständigen draw zu vermeiden
5. **Testing**
    - Mockito für Unit Tests
    - Repository-Mocking
    - Test-Coverage der kritischen Komponenten

Gelöste Probleme

- Thread-Safety bei Battle-System
- Korrekte Kartenzuweisung nach Package-Kauf
- ELO-Berechnung und Statistik-Tracking

Unit Testing Entscheidungen

- **User Management Tests**
    - Login validation
    - User creation
    - Profile updates
- **Battle System Tests**
    - Winner determination
    - Special card interactions

    - Damage calculation
- **Card Management Tests**
    - Package creation/validation
    - Deck configuration
    - Card acquisition

22 Unit Tests implementiert, die Kernfunktionalität und Randfälle abdecken.

Unique Feature

Erweitertes Battle-System mit:

- applyBooster Methode um ständiges unentschieden zu verhindern
- HTTP Statuscode bei „empty list“ auf 204 gesetzt

Zeitaufwand (90+ Stunden)

- Projektplanung & Design: 25+h
- Datenbankimplementierung: 15h
- Server-Entwicklung: 20h
- Battle-System: 25h
- Testing & Debugging: 15h
- Dokumentation: 5+h

GitHub Repository

https://github.com/if23b152/MTCG.git

Implementierungsstatus

- ✓ Java Implementation
- ✓ Eigener HTTP-Server
- ✓ Thread-Management
- ✓ PostgreSQL Integration
- ✓ SQL-Injection Prevention
- ✓ Unit Tests (20+)
- ✓ REST API Endpoints
- ✓ Battle-Logik

- ✓ Package-Management
- X Trading-System