Monster Card Game - Development Protocol

1.Technical Steps and Design Choices

Overview

This protocol documents the development of a REST-based server for a **Monster Trading Card Game (MCTG)** using Java. The goal of the project is to implement core functionalities, such as user registration, login, card management, and trading operations, all within a REST API structure. The server has been designed using Java's built-in "HttpServer" class without relying on external HTTP frameworks (such as Spring), and it follows token-based authentication for security.

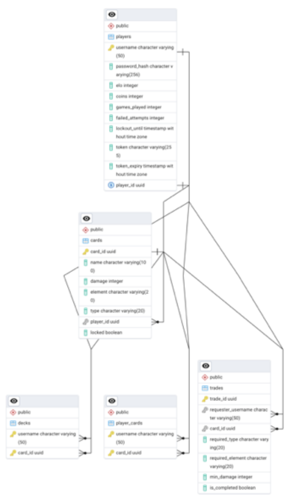
Design Choices:

The project follows an object-oriented design, with key classes representing players, cards, and battles. This structure promotes modularity, making it easier to add features or adjust existing functionality. Decks are designed to hold exactly four cards to maintain balance, and validation rules prevent incomplete or oversized decks from being used in battles. The battle system implements turn-based mechanics, where each card's attributes—such as damage and elemental type—determine the outcome. Decks are locked during battles to ensure fairness, preventing last-minute modifications. All data is stored is a Postgres database.

Packages and Classes:

- cards: Card, MonsterCard, SpellCard
- db: DBConnection
- game: Battle, Booster
- player: Player, UserController, UserRepository, UserService
- requestHandler: Request, RequestHandler
- scripts: test_mctg.sh
- server: HttpMethod, MCTGServer
- trading: Trade, TradeController, TradeRepository

Database Schema:


public
players
username character varying (50)
password_hash character varying(256)
elo integer
coins integer
games_played integer
failed_attempts integer
lockout_until timestamp without time zone
token character varying(255)
token_expiry timestamp without time zone
player_id uuid
public
cards
card_id uuid
name character varying(100)
damage integer
element character varying(20)
type character varying(20)
player_id uuid
locked boolean
public
decks
username character varying (50)
card_id uuid
public
player_cards
username character varying (50)
card_id uuid
public
trades
trade_id uuid
requester_username character varying(50)
card_id uuid
required_type character varying(20)
required_element character varying(20)
min_damage integer
is_completed boolean

Failures and Solutions:
Several issues were encountered during development, primarily around deck validation, user updates, and token management. One major issue involved receiving a 400 Bad Request - Missing token error during user profile updates. This was resolved by explicitly passing the authorization token in the request body. Another frequent issue involved duplicate cards appearing in decks, causing battle errors. This was addressed by enforcing a strict check to ensure each deck contains exactly four unique cards. Additionally, inconsistencies in the update process for usernames and passwords were resolved by including both the current and new usernames in the payload.

2. Unit Tests

Implemented Unit Tests:
The tests focus on core game mechanics and user management. Ensuring the deck validation, battle system, and player management work correctly is crucial to maintaining the integrity of the game. The unit tests simulate real-world scenarios, such as invalid deck sizes and incorrect battle setups, to prevent runtime errors.

- BattleTest
- BoosterTest
- CardTest
- PlayerTest

Critical Code and Testing Breakdown:
The BattleTest class verifies that battles execute correctly, including validation for deck size and damage calculation. It ensures that invalid decks result in errors before battles start. The BoosterTest ensures that the new feature is applied properly. The PlayerTest class focuses on deck construction, ELO adjustments, and package acquisitions, guaranteeing that player resources are handled properly. The CardTest class checks for uniqueness by validating the UUID generation for each card. This prevents players from acquiring duplicate cards through potential loopholes.

3. Time Tracking:
Development was divided into several key phases. The initial project setup took approximately 6 hours, during which the foundational classes were created. Implementing card and deck functionality required an additional 12 hours, with another 6 hours dedicated to developing the player class. The battle system, being the most complex part of the project, required 12 hours to implement and refine. Additional time was spent on database integration, testing, and debugging, totaling another thirteen hours. Token management, security, and documentation required approximately 11 hours combined.

4. Key Design Decisions
Card Uniqueness and Identity:
A unique identifier (UUID) is assigned to each card upon creation, ensuring that no two cards

share the same identity. This design decision prevents duplication errors and guarantees that each card remains traceable throughout the game. UUID-based identification was chosen over traditional incrementing IDs to avoid collisions during card trading or future expansions that may involve distributed databases.
The uniqueness of cards also plays a role in deck construction, as the system prevents players from adding the same card multiple times to a single deck. This enforces variety and promotes diverse deck strategies, pushing players to explore different card combinations rather than relying on the repetition of overpowered cards.

Simplified Token and Session Management:
A streamlined token-based authentication system was chosen for user verification and session management. This decision balances simplicity and security by enabling players to manage their sessions without complex session IDs or server-side tracking. Tokens are generated upon login and must be passed during any sensitive operation, such as updating profiles or acquiring booster packs. The decision to include the token in the request body during updates ensures that no operation can bypass authentication. This reduces the risk of unauthorized modifications to player profiles and prevents tampering with critical game data.

5. Lessons Learned:
One major takeaway from the project was the importance of token management during API interactions. Early errors resulted from missing or invalid tokens, highlighting the need to consistently pass authorization headers. Addressing this required modifying both the backend logic and the curl scripts to ensure tokens were correctly handled during user updates. Additionally, the project reinforced the value of automated testing. By writing unit tests early, errors in deck validation and battle logic were detected before deployment. This saved significant time during the later stages of development.

Link to Github: https://github.com/TD717/MCTG_Project