Implementation Decisions

Implemented Card Interface where Monstercard and Spellcard are inherited from to later store Monstercard and Spellcard in a list of the Card Interface.

Implemented Battle with 2 players with a ConcurrentQueue, to store a user's battle request until a second battle request is received. Then use the battle function in the battle handler and send both clients the same responses.

Created tables:

- Battle History: store each occurred battle
- Cards: store all existing cards in the game
- Deck: store user's deck
- Packages: store packages
- Sessions: store sessions
- Stack: store user's card
- Stats: store stats and scoreboard
- Tradings: store trade deals
- Users: store users

Created enums (types):

- Card type
  - Monster, Spell
- Element
  - Water, Fire, Normal
- Monster
  - Null, Goblin, Dragon, Wizard, Ork, Knight, Kraken, Elf
- Trading status
  - Open, done, deleted

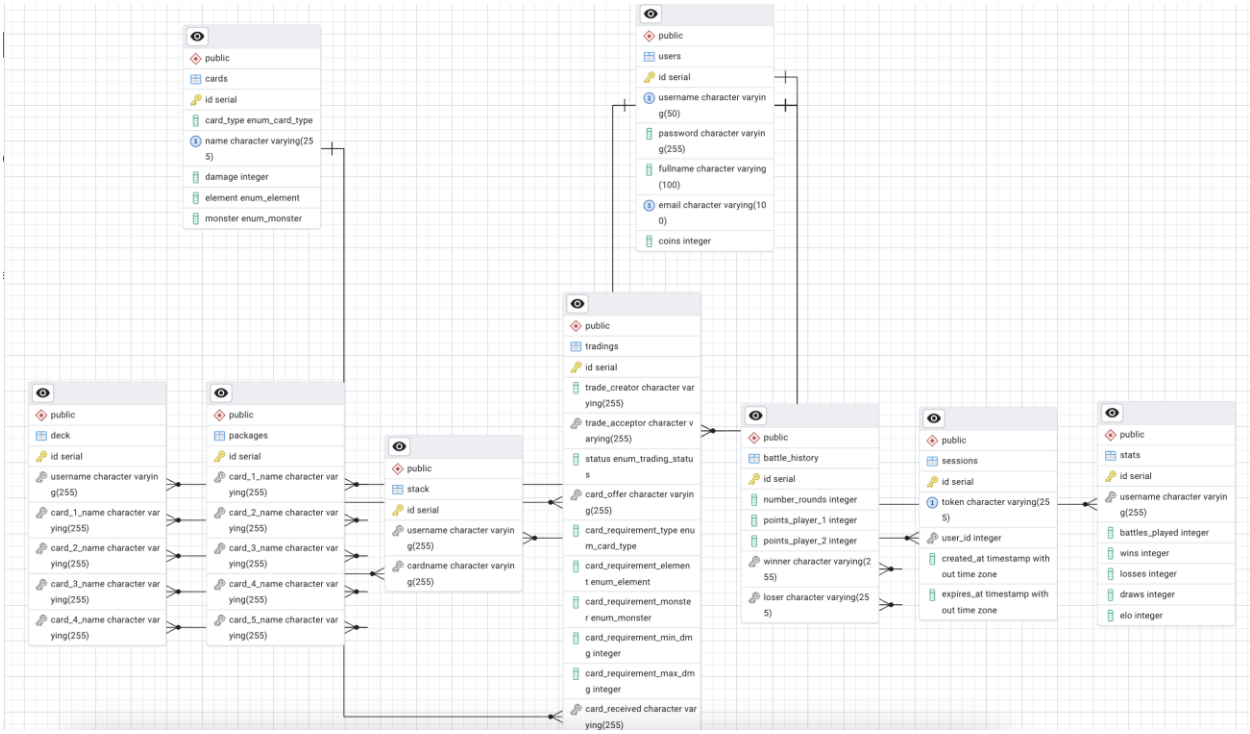
Created a separate database for unit tests.

Project Structure

- Handler
  - Contains handlers that define all routes and format responses
    - BattleHandler

        - CardHandler
        - SessionHandler
        - TradingHandler
        - UserHandler
- Interfaces
    - Interface for cards and handlers
- Misc
    - Contains enums and helper functions
- Models
    - Contains all essential classes, further divided in:
        - Battle
        - Card
        - Stats
        - Tradings
        - Users
- Network
    - Contains classes for request handling and database connection
- Repositories
    - Contains all database query functions for database transactions
- Specs
    - Contains unit tests for
        - User
        - Package
        - Deck
        - Battle

ERD Table Diagram



Implemented Features

Users Route

- POST users
  - Register user
- GET users/username
  - Display user information
- PUT users/username
  - Update user information
    - Username, Password, Fullname, Email can be updated individually

Sessions Route

- POST sessions
  - Login user

Packages Route

- POST packages
  - Create package

- POST transactions/packages
    - Buy package

Cards / Stack Route

- GET cards
    - Display all of user's owned cards

Deck Route

- GET deck
    - Display user's configured deck
- PUT deck
    - Configure deck

Battle Route

- POST battle
    - Requires two battle requests of different users

Stats Route

- GET stats
    - Display user's battle stats

Scoreboard Route

- GET scoreboard
    - Display the stats of all users

Tradings Route

- GET tradings
    - Display all open trade deals
- POST tradings
    - Create a trade deal
- POST tradings/id
    - Accept a trade deal
- DELETE tradings/id
    - Delete a trade deal

Lessons learned

- Unit Testing with NUnit
    - Testing not only verifies correctness but also reveals edge cases, such as invalid user actions or unexpected inputs
- Deeper Understanding of RESTful API implementation
    - Understanding low-level HTTP concepts, request parsing, response formatting, managing headers
- Database Integration with PostgreSQL and C#

Unit Testing Decisions

Unit Tests use their own testing environment database where the data created during the unit tests get deleted in the teardown.

- User
    - Creating users with taken usernames and taken email addresses
    - Get, Update, Login
    - Tests with different user input and checking for edge cases
- Package
    - Creating packages with non-existent cards
    - Buy packages
- Deck
    - Create deck with non-existent cards and missing input
- Battle
    - Testing the battle function between two cards and cover many edge-cases
    - Running the full battle function between two users and check their stat afterwards

Unique Feature

The Unique Feature implements a battle rank to a user that is displayed on a user's stat page and on the scoreboard. The rank displays a user's skill and their Elo. The ranks are divided in different Elo intervals.

- Bronze: <110
- Silver: 110-125
- Gold: 125-140
- Platinum: 140-155

- Diamond: 155-170
- Master: >170

Tracked Time

Task	Hours Spent	Notes
Initial Setup	1 hour	Setting up environment and dependencies.
Model Design	1 hour	UML Class Design
Model Implementation	5 hours	Class and project structure implementation
Routes and Handler Implementation	10 hours	Coding and testing API endpoints
Repository Implementation	8 hours	Implementation of all DB transactions and queries
Database Implementation	5 hours	Designing table structure and creating database, tables, enums
Unit Tests	2 hours	Writing unit tests for user, package, deck and battle
Integration Tests	2 hours	Testing with curl and postman
Documentation	2 hours	Preparing final protocol document
Total	36 hours	

Integration Testing

Postman collection for all endpoints in “MTCG Project.postman_collection.json”.

Link to GIT

https://github.com/kenno10101/MTCG_SWEN1