

Protocol

Software Engineering 1 Labor
BIF3-A1/A2 (Holzer Raoul)

MonsterTradingCardsGame

Armin Dervisefendic if23b040

<https://github.com/mide553/MonsterTradingCardsGame.git>

Project Overview

The Monster Trading Cards Game (MCTG) is implemented as a REST-based server application in C# that enables users to:

- Register and login with authentication
- Manage trading cards (acquire packages, build decks)
- Battle other players with special card interactions
- Trade cards with other players
- View scoreboard and player profiles

Implementation Steps

1. Server (Server.cs)

- TCP listener on port 10001
- Request/response handling
- Authentication via tokens
- Game logic implementation

2. Database (Database.cs)

- PostgreSQL database integration using Npgsql
- User data persistence
- Card management
- Deck storage

3. Models

- User.cs: User model with stats and cards
- Card.cs: Card model with type/damage logic
- Trade.cs: Trade request handling

Key Features

User Management

- Registration and login with token generation
- Profile management with ELO rating system
- Coin-based economy (starting with 20 coins)

Card System

- Card acquisition (5 coins per package)
- Deck building (4 cards per deck)
- Special card type interactions:
 - Water spells vs Knights (instant win)
 - Goblins vs Dragons (Goblins can't attack)
 - Wizard vs Ork control
 - Kraken spell immunity
 - FireElf Dragon evasion

Battle System

- Turn-based combat
- Special interaction rules
- ELO rating adjustments
- Card trading between winners/losers

API Endpoints

- POST /register # User registration
- POST /login # User login & token generation
- POST /cards # Add cards
- GET /cards # Get user's cards
- POST /acquire # Buy card package
- POST /deck # Configure deck
- POST /battle # Battle other players
- GET /stack # View card collection
- GET /scoreboard # View player rankings
- GET /profile # View user profile

Database Schema

```
users (  
  username VARCHAR(50) PRIMARY KEY,  
  password VARCHAR(100),  
  token VARCHAR(100),  
  coins INTEGER DEFAULT 20,  
  elo INTEGER DEFAULT 100,  
  games_played INTEGER DEFAULT 0  
)  
  
cards (  
  id SERIAL PRIMARY KEY,  
  name VARCHAR(50),  
  type VARCHAR(50),  
  power INTEGER,  
  is_spell BOOLEAN,  
  damage INTEGER,  
  owner_username VARCHAR(50) REFERENCES users(username)  
)  
  
deck_cards (  
  username VARCHAR(50) REFERENCES users(username),  
  card_id INTEGER REFERENCES cards(id),  
  PRIMARY KEY (username, card_id)  
)
```

How to Run the Monster Trading Cards Game Server

Prerequisites

Before running the server, ensure you have the following installed on your system:

1. **.NET SDK**: Make sure you have the .NET SDK installed.
2. **IDE**: An Integrated Development Environment (IDE) like Visual Studio or Visual Studio Code is recommended for easy code editing and debugging.

Steps to Run the Server

1. Clone the Repository

Clone the project repository to your local machine using the following command:

```
git clone https://github.com/mide553/MonsterTradingCardsGame.git
```

2. Navigate to the Project Directory

Open a terminal or command prompt and navigate to the project directory:

```
cd MonsterTradingCardsGame/MCTGServer
```

3. Server can be run in VS by building the project and running or by a CMD commands:

- dotnet build
- dotnet run

4. Upon successful execution, you should see a message indicating that the server has started and is listening on port 10001:

Output: Server started on port 10001

You can test the server with following commands:

Register two users:

```
curl -X POST http://localhost:10001/register --header "Content-Type: application/json" -d
{"Username\":\"armin\", \"Password\":\"armin\"}
curl -X POST http://localhost:10001/register --header "Content-Type: application/json" -d
{"Username\":\"bob\", \"Password\":\"bob\"}
```

```
[time] Received request: POST /register
[time] Register request received
[time] User registered: armin
[time] Received request: POST /register
[time] Register request received
[time] User registered: bob
```

Login and save tokens:

```
curl -X POST http://localhost:10001/login --header "Content-Type: application/json" -d
{"Username\":\"armin\", \"Password\":\"armin\"} | awk -F' ' '{print $4}' > token1.txt
curl -X POST http://localhost:10001/login --header "Content-Type: application/json" -d
{"Username\":\"bob\", \"Password\":\"bob\"} | awk -F' ' '{print $4}' > token2.txt
```

```
[time] Received request: POST /login
[time] Login request received
[time] User logged in: armin
[time] Received request: POST /login
[time] Login request received
[time] User logged in: bob
```

Add special test cards:

For Armin: WaterSpell vs Knight interaction

```
curl -X POST http://localhost:10001/cards --header "Authorization: Bearer $(cat token1.txt)"
--header "Content-Type: application/json" -d '{"Name\":\"WaterSpell\", \"Type\":\"Water\",
\"Power\":20, \"IsSpell\":true, \"Damage\":50}'
curl -X POST http://localhost:10001/cards --header "Authorization: Bearer $(cat token1.txt)"
--header "Content-Type: application/json" -d '{"Name\":\"Knight\", \"Type\":\"Normal\",
\"Power\":30, \"IsSpell\":false, \"Damage\":35}'
```

For Bob: Dragon vs Goblin interaction

```
curl -X POST http://localhost:10001/cards --header "Authorization: Bearer $(cat token2.txt)"
--header "Content-Type: application/json" -d '{"Name\":"Dragon\","Type\":"Monster\","Power\":25,"IsSpell\":false,"Damage\":70}'
curl -X POST http://localhost:10001/cards --header "Authorization: Bearer $(cat token2.txt)"
--header "Content-Type: application/json" -d '{"Name\":"Goblin\","Type\":"Normal\","Power\":15,"IsSpell\":false,"Damage\":25}'
```

```
[time] Received request: POST /cards
[time] Add card request received
[time] Card added: WaterSpell for user armin
[time] Received request: POST /cards
[time] Add card request received
[time] Card added: Knight for user armin
[time] Received request: POST /cards
[time] Add card request received
[time] Card added: Dragon for user bob
[time] Received request: POST /cards
[time] Add card request received
[time] Card added: Goblin for user bob
```

Get packages for both users:

```
curl -X POST http://localhost:10001/acquire --header "Authorization: Bearer $(cat token1.txt)"
--header "Content-Type: application/json"
curl -X POST http://localhost:10001/acquire --header "Authorization: Bearer $(cat token2.txt)"
--header "Content-Type: application/json"
```

```
[time] Received request: POST /acquire
[time] Acquire package request received
[time] Package acquired by user: armin
[time] Received request: POST /acquire
[time] Acquire package request received
[time] Package acquired by user: bob
```

Check stacks:

```
curl -X GET http://localhost:10001/stack --header "Authorization: Bearer $(cat token1.txt)"
curl -X GET http://localhost:10001/stack --header "Authorization: Bearer $(cat token2.txt)"
```

```
[time] Received request: GET /stack
[time] Stack retrieved for user: armin
[time] Received request: GET /stack
[time] Stack retrieved for user: bob
```

Configure decks with special cards:

```
curl -X POST http://localhost:10001/deck --header "Authorization: Bearer $(cat token1.txt)"
--header "Content-Type: application/json" -d '[
  {"Name":"WaterSpell","Type":"Water","Power":20,"IsSpell":true,"Damage":50},
  {"Name":"Knight","Type":"Normal","Power":30,"IsSpell":false,"Damage":35},
  {"Name":"Card1","Type":"Type1","Power":10,"IsSpell":false,"Damage":10},
  {"Name":"Card2","Type":"Type2","Power":20,"IsSpell":true,"Damage":20}
]'
```

```
curl -X POST http://localhost:10001/deck --header "Authorization: Bearer $(cat token2.txt)"
--header "Content-Type: application/json" -d '[
  {"Name":"Dragon","Type":"Monster","Power":25,"IsSpell":false,"Damage":70},
  {"Name":"Goblin","Type":"Normal","Power":15,"IsSpell":false,"Damage":25},
  {"Name":"Card1","Type":"Type1","Power":10,"IsSpell":false,"Damage":10},
  {"Name":"Card2","Type":"Type2","Power":20,"IsSpell":true,"Damage":20}
]'
```

```
[time] Received request: POST /deck
[time] Manage deck request received
[time] Deck updated for user: armin
[time] Received request: POST /deck
[time] Manage deck request received
[time] Deck updated for user: bob
```


Start a battle:

```
curl -X POST http://localhost:10001/battle --header "Authorization: Bearer $(cat token1.txt)"  
--header "Content-Type: application/json" -d '{"bob"}"
```

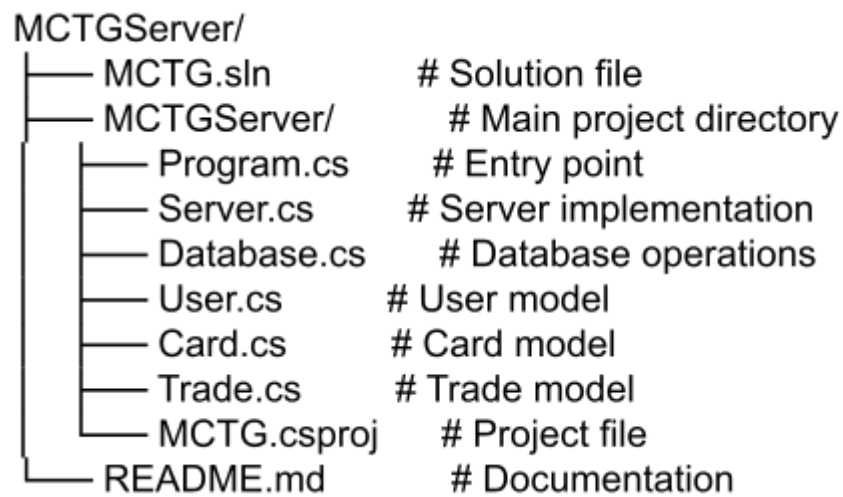
```
[time] Received request: POST /battle  
[time] Battle request received  
[time] Battle started: armin vs bob  
[time] Battle completed: bob wins! (16 rounds)
```

Check results:

```
curl -X GET http://localhost:10001/scoreboard  
curl -X GET http://localhost:10001/profile --header "Authorization: Bearer $(cat token1.txt)"  
curl -X GET http://localhost:10001/profile --header "Authorization: Bearer $(cat token2.txt)"
```

```
[time] Received request: GET /scoreboard  
[time] Scoreboard retrieved  
[time] Received request: GET /profile  
[time] Profile retrieved for user: armin  
[time] Received request: GET /profile  
[time] Profile retrieved for user: bob
```

Database Schema



```
graph TD; MCTGServer["MCTGServer/"] --- MCTGSln["MCTG.sln # Solution file"]; MCTGServer --- MCTGServerDir["MCTGServer/ # Main project directory"]; MCTGServerDir --- ProgramCs["Program.cs # Entry point"]; MCTGServerDir --- ServerCs["Server.cs # Server implementation"]; MCTGServerDir --- DatabaseCs["Database.cs # Database operations"]; MCTGServerDir --- UserCs["User.cs # User model"]; MCTGServerDir --- CardCs["Card.cs # Card model"]; MCTGServerDir --- TradeCs["Trade.cs # Trade model"]; MCTGServerDir --- MCTGcsproj["MCTG.csproj # Project file"]; MCTGServerDir --- READMEmd["README.md # Documentation"];
```

MCTGServer/
— MCTG.sln # Solution file
— MCTGServer/ # Main project directory
 — Program.cs # Entry point
 — Server.cs # Server implementation
 — Database.cs # Database operations
 — User.cs # User model
 — Card.cs # Card model
 — Trade.cs # Trade model
 — MCTG.csproj # Project file
— README.md # Documentation