# Monster Trading Card Game Protocoll

[Monster Trading Card Game Protocoll 1](#_Toc188981406)

[Introduction 1](#_Toc188981407)

[Features: 2](#_Toc188981408)

[Struktur: 2](#_Toc188981409)

[The Endpoints available are: 2](#_Toc188981410)

[Post /users 2](#_Toc188981411)

[Post /sessions 2](#_Toc188981412)

[Post /showpackages 2](#_Toc188981413)

[Post /package 2](#_Toc188981414)

[Post /score 2](#_Toc188981415)

[Post /change 2](#_Toc188981416)

[Post /showdeck 3](#_Toc188981417)

[Post /profile 3](#_Toc188981418)

[Post /editdeck 3](#_Toc188981419)

[Post /queue 3](#_Toc188981420)

[Database 3](#_Toc188981421)

[card 3](#_Toc188981422)

[collection 3](#_Toc188981423)

[Deck 3](#_Toc188981424)

[packcontents 3](#_Toc188981425)

[tcguser: 3](#_Toc188981426)

[Battle logic: 4](#_Toc188981427)

[Unit tests 4](#_Toc188981428)

[Lessons Learned 4](#_Toc188981429)

[TimeTable: 4](#_Toc188981430)

[Conclusion 4](#_Toc188981431)

## Introduction

This document describes a Card game server, in which we can login, create and buy packages, and fight. For this I created my own rest system and http server without a framework, and I use a token system that authorizes a User to do anything. There are multiple HTTP Endpoints, each with a different set of instructions and requirements. The Server also works in a multithreading way, making it very efficient in doing those tasks. There also are 20 or more Unit tests.

## Features:

### Struktur:

The Project has a few main Classes:  
Program.cs: here the Main http server gets initialized and started.

HTTP.cs: This Class is the main connector as the http server. It starts with the connection listener and also does the multithreading. If a connection comes in, it splits the message into header and body, and also gives it to the Handler.

Handler.cs: this is the main logic of where the requests are going to, and what functions do.

DBConnector.cs: this is the function that connects to the server, gets and writes information to it/from it, and also has every supporting function that the Handler may need.

BatteLogic.cs: Here the fighting queue and the fighting logic is being played out. It returns the information(log) of the fight and who won.

#### The Endpoints available are:

* Post /users
* Post /sessions
* Post /createpackage
* Post /showpackages
* Post /package
* Post /score
* Post /change
* Post /showdeck
* Post /profile
* Post /editdeck
* Post /queue

Where each Endpoint has a specific Task to solve.

#### Post /users

Tries to create a User account with the given username and password.

#### Post /sessions

Tries to create a Token for Authentication for the User.

#### Post /showpackages

This shows all packages that are available to buy

#### Post /package

Here a User can buy a Token if he has enough coins to do so.

#### Post /score

Shows the 10 best Players that exist

#### Post /change

Here a user can change either his password or his Username.

#### Post /showdeck

This shows all the cards associated with the User logged in.

#### Post /profile

This shows the profile of the User logged in.

#### Post /editdeck

Here the user can edit his deck for fights.

#### Post /queue

This is the main battle Endpoint, here Players can queue into a fight, and, if enough Fighters are there to fight, a fight will start.

How each of those HTTP requests are made up can be seen in the given curl script.

### Database

The database is a PostgresSQL db with with the Tables:

#### card

here the information about each card is being stored, with a id, name, damage, type, and species. The Type is either normal, fire or water, and is corresponding to efficiency in a fight. The species is a integer that is being used in a monster vs monster fight to check for special conditions.

#### collection

here all cards of a user are being saved. There is a userid, which corresponds to a userid in the user database, a cardid corresponding to the card database, and a count row, how often a user has this card.

#### Deck

This database has all the Fighting cards in it. Each user can have up to 4 Cards in his fighting database, which which he can queue into battle. Only 2 rows with secondary keys, userid and cardid.

#### packcontents

This database has all Packs with their cards in it. this pack has its own id, packcontentid, and secondary keys to packid and cardid, showing which pack has which cards to get from.

Packs

This Pack shows us the Packid, the name of the pack and how much it costs to buy.

#### tcguser:

this is the user that can do everything, he has his own id, a username, a password, coins to buy packs with, and elo, which he can gain or loose by winning or loosing in battle, and a security token, which he needs for authorization purposes.

## Battle logic:

Each user can, at all times, go into battle queue, even without having cards in his battle deck. There it checks if there are enough players for the fight to work, and either send back a 200 ok, if no other player has been found, or a 200 ok – with battle text, if another player has been found. Should no more Cards be in a player’s deck, or he has queued into the battle without Cards, he looses the match, giving the winner 5 Elo points and the looser -3 Points. If there is a Draw, the battle log will be put out, but no Elo will be distributed. As a notch to a little more strategic value, I, instead of making the cards fight randomly, made them fight in the order they were selected into the deck of each user, making you able to try to predict opponents cards.

## Unit tests

There are 20 Unit tests, testing many components of the program. The unit tests were made post hoc, and try to test specific processes within functions, to eliminate unwanted behavior.

## Lessons Learned

One of the more interesting things I thought about were how to connect the problems, and database calls to get what I want. I also underestimated the project, thinking it was not as hard, but given the many database calls and how each call needs to do something quite close to other calls, I should have sat down before programming and make a few basic functions for database calls, and then just combine them instead of writing new ones for each component.

## TimeTable:

Project Setup(git, vs) 3h

HTTP Server 10h

Rest Server and Parsing 3h

Database Connections 25h

Battle logic 5h

Debugging 8h

Unit-Tests 10h

Documentation 2h

Rewriting Project 8h

ALL IN ALL: 74

## Conclusion

It was a much harder project than initially thought, which took a lot longer than thought. It also didn’t help that I jumped immediately into programming instead of making a chart, which would have helped me not write code twice and be better prepared for what happened. I also had a lot of problems with the database, because I wrote in Uppercase letters, which often times made my query’s not work.