FH University of Applied Sciences
TECHNIKUM WIEN

Protocol: MTCG

SWEN1 Project

Mark Youssef

Design Implementation

The Project was split into multiple categories including.

- Database
- Hashing
- Enums
- Essentials
- Logic
- Server
- Parse
- Response
- Templates
- Unit Testing using NUnit.

Everything was tested via a curl script and so the project only works with that one being given; new commands can be added if desired. The “Backend” of the server consists of an HTTP Server that listens to incoming calls triggered by the curl script and executes them accordingly. The data is first being parsed, enabling the program to gather valuable information about the request using its header data, this is being saved into a dictionary that is part of a thread. Each user has their own dictionary with their own data. Through logging in the data gets saved in there and a token is being created, same with registration. Most functions then must go through a Message Handler, which checks for the path in the dictionary, this one sends the data to get parsed, as some of the requests have a JSON attached to them. Afterwards the data gets returned and moved straight to the DB Handler, which takes over all functionality regarding access, fetching, deleting, writing, ... to the Database. Depending on the circumstance, different HTTP codes will be returned alongside other information. If for example data is being fetched, the function will have two return types. One of which being the HTTP Code and the other one being the fetched data within a string, list, array, or any other datatype. The code alongside the data gets then sent to the Response Handler, where the data gets prepared to be sent back to the curl script as some sort of visual response to what was going on in the background.

Battle

The battle was special as it required a lobby to be created, it was a List of Users which was instantiated within the HTTP server so all threads can access it without it being created newly with each new thread. Every time a thread starts, a User is being created with the given information (token) and all the data is fetched and initialized, it then gets put into a lobby. Once two players are in the lobby, one of the threads continues with both players whilst the other one remains waiting until the game is done. The List in the meantime gets cleared and opened for two new threads to do the same. Once the battle is over, it returns a log with all the happenings within the event, as we only want the battle to happen once and not both threads doing it at the same time – we for once lock the thread and twice we set a bool that enables us to only run the battle function once. Both threads then get the log and move on to the Response Handler where it then is being sent to the curl script twice (on purpose!).

Lessons Learned

- Getting to know the usage of PGAdmin 4 and how to properly work with a backend and database by also considering things like SQL Injections and proper and performant SQL statements.
- Getting to know the usage of threads and why locks are so important, as they for me caused major implications regarding race conditions and unwanted behaviour through the code.
- Learning about the importance of a Version Control tool such as GIT and that changes are never truly lost if it's backed up on a remote repository plus its convenience of accessing your project from whichever device you own.
- Never pushing anything up until the last second and then panicking about not being able to finish (totally not me)
- Having a proper database design by using things like references and safeguards such as constraints if the logic within the source code itself is not enough.
- Learning about how to properly use VS as it's not been my primary IDE, it has many unique features and a very powerful intellisense and debugger display.

Unit Testing Decisions

The unit testing was kept rather simple due to the time constraints that went on throughout the entirety of the semester because picking up a part time SE job next to your full-time studies is a terrible idea!

It mostly consistent of edge cases being tested which cannot usually happen through the program as it's got enough safeguards built in, but better be safe than sorry! It did help find some mistakes.

Classes that were tested within the 20 required Unit Tests:

- Battle Logic
- Message Handler
- Parse Data
- Password Hasher
- Response Handler

Battle Logic, most of the tests that happened here were due to the split up of many functionalities in small methods that were easily approachable.

- I tested whether the cards after each round were truly moved to the opposing deck.
- I tested it the other way around too, since there's a lot of playrooms by using triple return types, mistakes could easily occur.
- I tested whether the special abilities buff changed the damage for cards, since again, multiple return types can be messy at times.
- I tested whether everything worked properly when both cards were monsters.
- I tested whether the fighting with abilities worked properly.

Message Handler, minor testing happened there as half of the code consists of one massive function which is rather hard to test. Following tests were executed.

- I tested whether the authorization worked properly by giving the function a predefined curl and seeing if the token was attached to the library with the right username.
- I tested another function that checked whether the user within the link was the same as the one in the token given by a predefined curl.

Parse Data contained multiple unit tests as it was easy to just write a JSON and seeing whether the proper output came out of it, no need to go into further detail as all functions were similarly built and did the same thing with different kinds of JSONs which are attached to the curl script.

Password Hashing, simple testing that is due to the complexity of the hash algorithms used which are defined in a library. It was simple null testing.

Response Handler consisted of quite a few tests but then again – it is the same as with the parsing. It's similar and simple to test as you just must give it a proper string to work with and expect an output to come out of it. Also did simple testing of what would happen if certain codes were not part of the Dictionary that I built with codes and their according reason sentences.

Unique Feature

My unique feature was a password hasher that returned a hashed password in base64 and a salt, both of which were saved in the database as character varying. When the user tries to log in, both are fetched and sent to a validation function that checks whether the given password matches the ones that were just reverted by using the hash and the salt. If so, true is being returned, the login is successful, and the user receives a token. Otherwise, false is being returned and the user is prompted to log in again. Both functions were within their own class called Password Hasher.

Time Tracked

Time spent on the project estimated to be around 75-85 hours, my “tracking” was done via all the commits I made throughout the past couple of weeks. Everything between them, I worked – as I (mostly) commit right after I created another piece of working code, so I don’t lose it.

GIT

https://github.com/mar-jo/MTCG.git