Protocol - MonsterCardTradingGame

First name: Michael

Surname: Reichenberger

Key Figure: if22b020

LINK TO GIT

https://github.com/MichaelReichenberger/MonsterCard.git

Design----------

When I plannend the design of the project, I oriented myself towards the layered architecture. I tried to separate and structure the HttpLayer (Server), the BusinessLayer, the DataAccesLayer and the Models.

I have divided the HttpLayer into:

- HttpServer-class
- Router-class
- RequestHandler-class

I have also created a subdirectory called Routes, which contains:

- Route-class
- RouteConfig-class.

There is also a Session subdirectory in the project, which contains the Session class and the SessionHandler class.

The BusinessLayer contains the manager classes:

- UserManager
- TransactionManager
- GameManager
- CardManager

It also contains the **Parser class** and its own very simple **Mapper class.**

The DataAccessLayer is divided into a Repositories directory and a DBAccess class. The Repositories directory contains the repository classes:

- UserRepository,
- TradingsRepository,
- StatsRepository,
- PackageRepository,
- DeckRepository,
- CardsRepository

and the **IRepository interface,** which all repositories implement.

While the **DBAccess class** is responsible for establishing a connection to the database (each repository has a DBAccess object), the repositories are responsible for queries to the database, with each repository (usually) processing queries to a specific table in the database.

The Models directory contains the classes that describe the objects which are passed back and forth between the individual parts of the application. These include:

- User-class
- UserStats-class
- TradingDeal-class
- CardDeck-class
- Card-class.

Mandatory unique Feature----------

For the mandatory unique feature i decided to add a simple **booster to a card after it has won 2 fights within a battle**. If the booster is active, the card instantly wins the next round, even when a special rule, or a spell with a certain element would prevent this.

I also added a **win/lose ratio** to the user stats.

Failures/Lessons learned----------

- Since I spent only a short time for the design at the beginning, I had the problem, that I had to change basic things in my project. This reached from the database schema, the architecture of the application, or even just the structure of the individual classes. These changes caused me to lose a lot of time and I had to make some major changes, especially towards the project deadline. In future, I will plan my software projects better and more precisely in advance to avoid these problems.
- I was also able to slightly improve my time management.
- I have also learned that it is sometimes better to take a little break from a certain topic/problem in order to be able to deal with it all the better afterwards.
- My skills with many different applications and tools, such as Git(GitHub), Docker, VisualStudio, DataGrip, Postman and the Windows Command Line were also constantly improved through the project.
- And last but not least, I learned how to create, execute and use unit tests to validate and test critical parts of the code.

Unit Test Selection----------

For the unit tests, I decided to test the really critical parts of the code, such as the login or the user registration, but also to test code that is very complex, such as the CardFight method in the GameManager.

- By testing the login, I wanted to make sure that an existing user (if using the correct credentials) can log in at any time. I also wanted to make sure that nobody can log in with the wrong credentials.
- I also tested the session manager with unit testing to make sure that the session management works correctly to prevent problems with login or token based security.
- I also decided to test some functions from the managers and the database to ensure the consistency, correctness and integrity of the data.
- The parser class was also tested as it is used in some important cases in the application.

I used mocking for a few unit tests, but this worked rather mediocre, as I had little time to deal with mocking in more detail.

On average, 29/30 of my unit tests were completed successfully

For the **integration tests**, I used the Curl scripts from the Moodle course and adapted them, inserted them into Postman, or executed them directly.

The Curl scripts can also be found in the project folder

Time spent----------

- According to Github, I ran about **45 Git commits**, estimating I worked about 2h per commit, which would make a total time of **90 hours**.
- But since I often worked longer, especially towards the end of the project period, I add another **20** hours on top.
- I spent a total of about **13 hours** on planning the project and, as mentioned above, I had to plan again and again.

This results in around **120-125** hours of work, although I could certainly have been much faster.