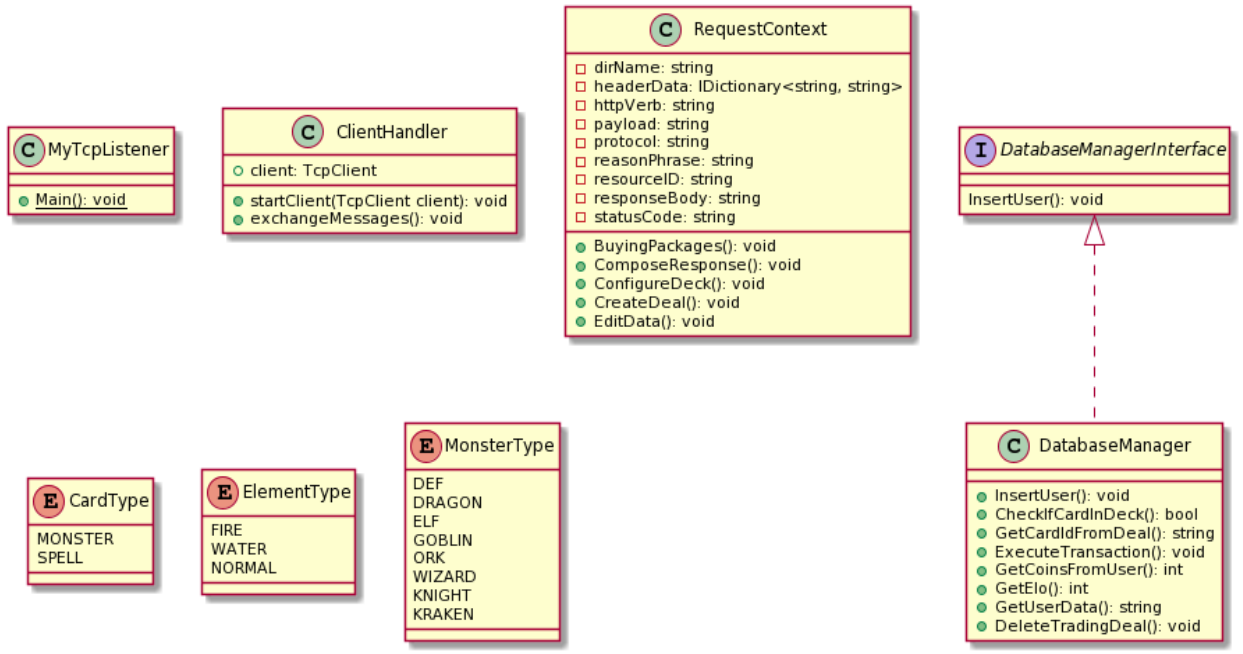
Protokoll Monster Card Trading Game

1. Design
   a. Server and Multi-Threading



In der main-Funktion der MyTcpListener habe ich mit Hilfe der TcpListener Klasse ein Listener aufgebaut der auf Verbindungen auf die IP-Adresse „127.0.0.1“ und Port „10001“ hört. Nachdem eine Verbindung aufgebaut wird, wird der Client der Funktion startClient von der ClientHandler Klasse übergeben. Im startClient wird das Threading durchgeführt. Die Methode exchangeMessages() wird dem Thread übergeben. Die exchangeMessages() liest den Request von Client ein und schickt eine entsprechende Nachricht zurück. Für weitere Implementierung der Card-Klassen habe ich entsprechende enums verwendet: CardType, ElementType, MonsterType.

b. Datenbank Schnittstelle

Die DatabaseManager Klasse ist zuständig für die CRUD Operationen im Bezug mit meiner MCTG Datenbank. Sämtliche Funktionen dieser Klasse werden dann in der RequestContext Klasse aufgerufen wo ich meine Endpoints behandle. Zb. Für den Request ..users/kienboec wird die Funktion GetUserData(kienboec) aufgerufen.

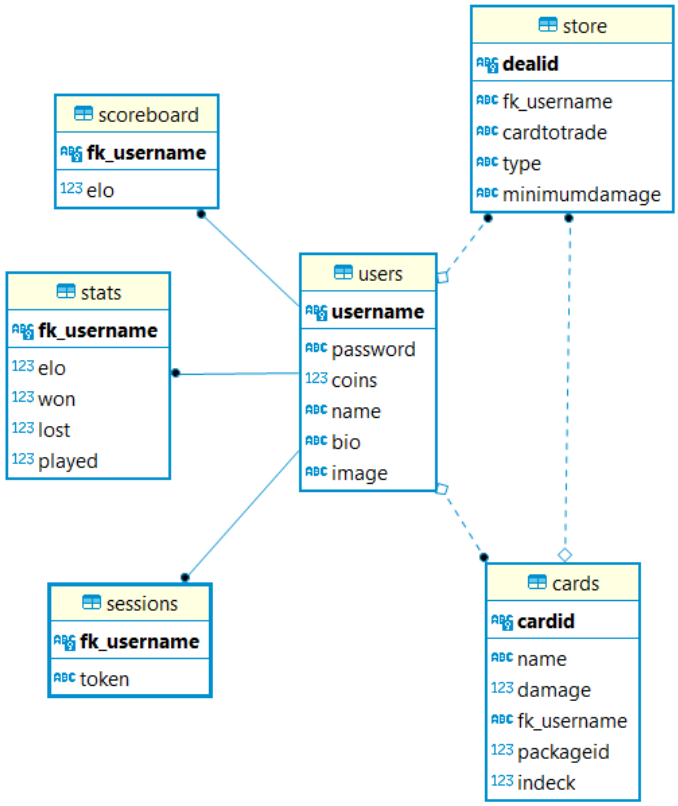
Für die Datenbankausführung habe ich postgre sql gewählt und für die Erstellung der Datenbank habe ich den pgAdmin 4 gewählt.

Das ER Modell wurde mit Hilfe von Dbeaver erstellt.

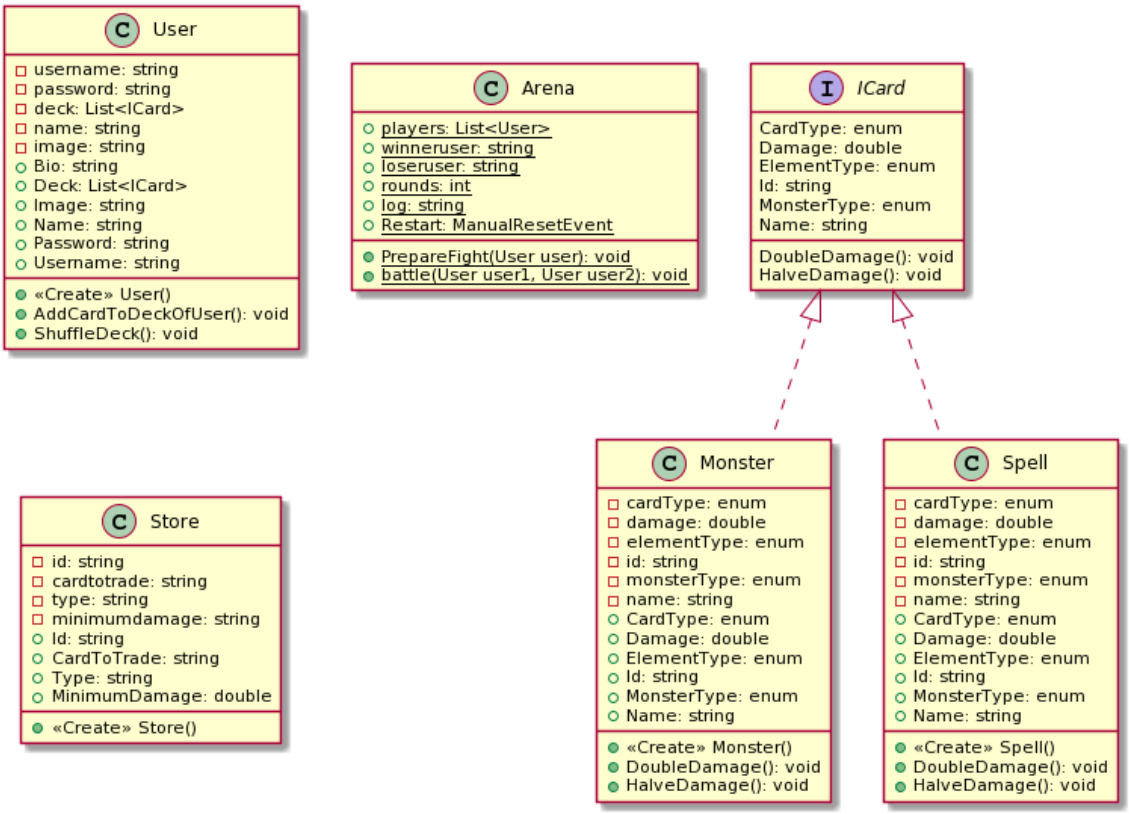
Foreign Keys:

Die foreign keys von den Tabellen scoreboard, stats, sessions, cards und store: fk_username referenziert den primary key (username) der Tabelle users.

Die Tabelle store hat noch ein foreign key (cardtotrade) der den primary key (cardid) aus der Tabelle cards referenziert.



c. Spiel Logik



Wenn ein User ein battle Request ausführt wird sein Deck befüllt mit den Karten von der Datenbank (die Karten die in dem Deck sind, sind mit indeck = 1 markiert, alle anderen mit indeck = 0). Das CardType und MonsterType wird berücksichtigt indem man 2 string-Bearbeitungsfunktionen auf das cards name durchführt. Für das Deck des User habe ich mich für eine Liste entschieden, da man auf beliebe Positionen Karten hinzufügen/entfernen kann. Im Endpoint Behandlung wird dann die statische Funktion PrepareFight(User user) aufgerufen, wo der User in der statischen Liste player hinzugefügt wird. Falls nur ein User in der Liste ist blockiere in den entsprechenden thread mit der Funktion WaitOne() von der ManualResetEvent Klasse.

Falls es 2 users in der players Liste gibt, wird die statische Funktion battle(user1, user2) ausgeführt. Die battle Logik wurde gemäß Angabe implementiert. Bei einem Vergleich zwischen FireElves und Dragons habe ich nur überprüft ob damageFireElf > damageDragon, ansonsten ist ein draw. Während des battles wird das Ganze in einem statischen Log gespeichert. Am Ende des battles gebe ich frei alle threads mit der Funktion Set() der Klasse ManualResetEvent die auch statisch in meiner statischen Arena Klasse deklariert ist.

2. Failures and Select Solutions

Da meine Datenbank nicht von Anfang an vollständig entworfen wurde, musste ich immer wieder das Programm ändern.

Ein weiteres Failure war das ich nicht von Anfang an test driven entworfen habe, sodass ich es nicht mehr geschafft habe die Datenbank zu mocken für meine Tests für die Datenbank.

3. Time tracking

Leider habe ich nicht von Anfang die genaue Zeit betrachtet, kann aber mit einer Fehlerspanne von +-5 Stunden folgenden Arbeitszeiten bekanntgeben:

REST-HTTP-Server ≈ 30 Stunden

Monster Card Trading Game ≈ 80 Stunden

4. GIT Link

https://github.com/watchsomegoals/SWE1-MCTG