Battleground

In this assignment we will be working with GraphQL to create a layer that supports a website for battling Pokémons. The data sources the GraphQL layer will use is an API exposing information about Pokémons and a database preserving data about players, battles and attacks.

Template

The assignment comes with a template which includes:

- **Battleground.Api**
    - Program.cs (*all GraphQL related boilerplate is included*)
    - **Schema/**
        - **Queries/**
        - **Mutations/**
        - **Types/**
        - **InputTypes/**
        - BattlegroundSchema.cs
- **Battleground.Models**
    - **Exceptions/** - contains all relevant exceptions that should be used when you want to throw a descriptive exception and handle it correctly
- **Battleground.Services**
    - **Implementations/**
        - BattleService.cs
        - InventoryService.cs
        - PlayerService.cs
        - PokemonService.cs
    - **Interfaces/**
        - IBattleService.cs
        - IInventoryService.cs
        - IPlayerService.cs
        - IPokemonService.cs
- **Battleground.Repositories**
    - **Entities/**
    - BattlegroundDbContext.cs
- Battleground.sln

Data source

There are two data sources: **a web service** and **a PostgreSQL database**. The web service will be accessible via this url: https://pokemon-proxy-api.herokuapp.com/ and there are two exposed endpoints: /pokemons and /pokemons/{pokemon-name}. This web service exposes all data related to information about Pokémons. It is recommended to use **HttpClient** (https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=net-6.0) to implement that functionality. The rest of the data is associated with the PostgreSQL database. It is your job to navigate to https://www.elephantsql.com/ and create a new database, set up the schema (See database schema section) and provide the data. **You must follow the schema setup as provided in the database schema section.**

Rules (10%)

- Applies to all
  - A query or mutation which accepts an id as a field argument must check whether the resource with the provided id exists
- Battle
  - A player can only partake in one active (*not started or started*) battle at a time
  - A battle may only consist of two players along with two pokemons
  - A participating player in a battle must own the pokemon he presents
  - A battling pokemon must be valid
  - A battling player must be valid
- Inventory
  - A player can only have one of each Pokémon, and therefore no duplicates are allowed in the inventory
  - Each pokemon can only appear in one inventory at a time
  - A player can only add a valid Pokémon to his inventory
  - A player's inventory has a capacity of 10 Pokémons
- Attack
  - An attacking Pokemon must exist
  - An attacking Pokemon must be linked to a battle

Battle

A battle is between two Pokémons which have different traits to suit them for battle. The battle is turn-based, where each player should attack its opponent. This happens until one Pokémon defeats the other Pokémon and is crowned the winner of the battle. There are two rules for each turn:

- Each attack has a 30% chance of being a critical attack and if the attack is a critical attack it will become 40% more powerful than the base attack
- Each attack has a 15% chance of not hitting the opponent - if that is the case no damage is made to the opponent during that turn

Assignment description

Below is a description of the functionality the **GraphQL API** should contain. For all reference on how the schema should be set up, look at the **GraphQL schema** section below.

- Queries **(30%)**
  All queries should exclude resources which are marked as deleted
  - **(5%) allPokemons** - Should return a list of all pokémons
  - **(5%) allBattles** - *Should return a collection of all battles. Contains a field argument called **status** which is of type **BattleStatus** (enum) and should be used to filter the data based on the status of the battle*
  - **(5%) allPlayers** - *Should return a collection of all players*
  - **(5%) pokemon** - *Should return a specific pokémon by id*
  - **(5%) battle** - *Should return a specific battle by id*
  - **(5%) player** - *Should return a specific player by id*

- Mutations **(35%)**
  *All field arguments to mutations should be defined as input types, see **Schema** section*
  - **(5%) addBattle** - *Create a battle between two players pokémons and returns the newly created battle*
  - **(5%) attack** - *Attacks a pokemon within a battle and returns the result*
  - **(5%) addPlayer** - *Create a player and return the newly created player matching the Player type*
  - **(5%) removePlayer** - *Marks a player as deleted and returns either true or an error if something happened*
  - **(5%) addPokemonToInventory** - *Add a pokémon to an inventory of a specific player and returns either true or an error if something happened. A player can only have one of each type - therefore no duplicates allowed in the inventory*
  - **(5%) removePokemonFromInventory** - *Removes a pokémon from an inventory of a specific player and returns either true or an error if something happened*
- **Bonus (5%)**
  - Eliminate the N+1 problem by using DataLoader (https://www.nuget.org/packages/GraphQL.DataLoader) for connected data. All boilerplate setup and dependencies are already configured within Program.cs
  - You can follow this guideline: https://graphql-dotnet.github.io/docs/guides/dataloader

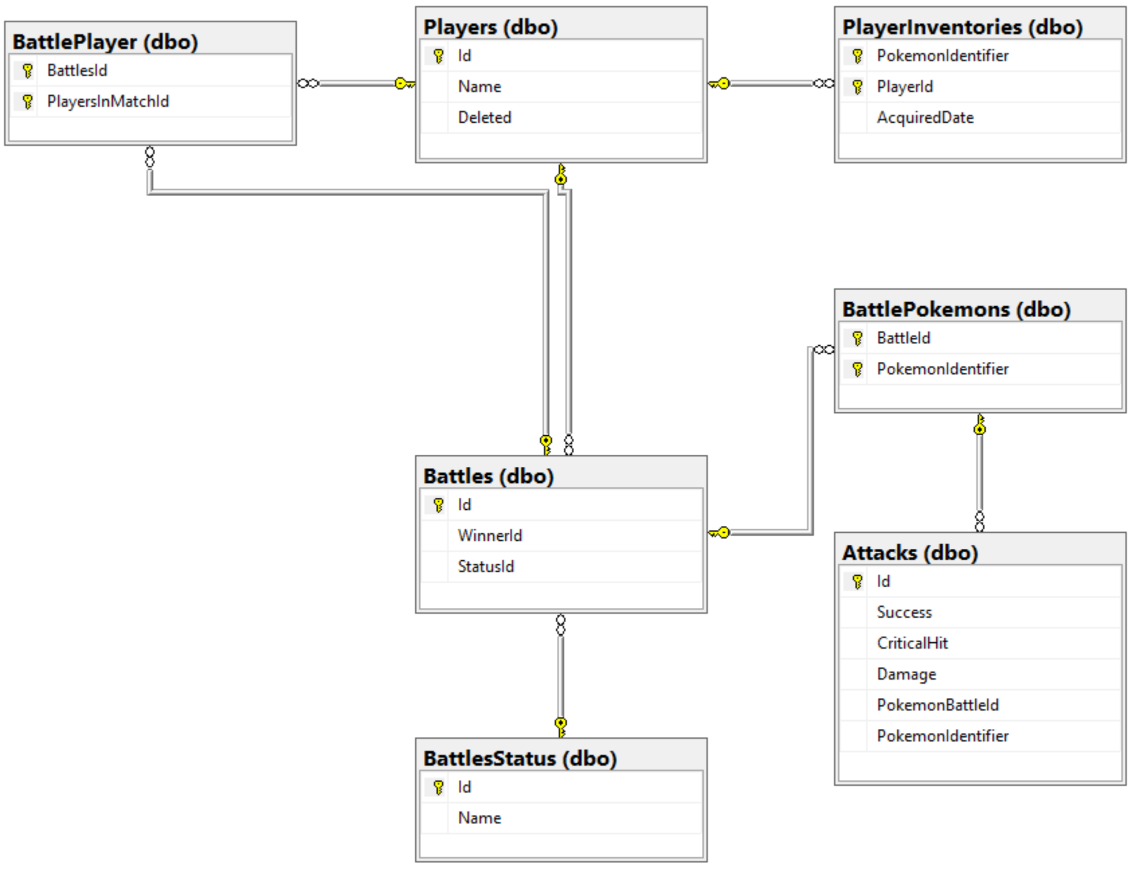
GraphQL schema (15%)

- Queries
  - pokemon(id: String): PokemonType
  - allPokemons: [PokemonType!]!
  - player(id: Int): PlayerType
  - allPlayers: [PlayerType!]!
  - battle(id: Int): BattleType
  - allBattles(status: BattleStatus): [BattleType!]!

- Mutations
  - addBattle(input: BattleInput!): BattleType!
  - attack(input: AttackInputType!): AttackType!
  - addPlayer(input: PlayerInput!): Boolean
  - removePlayer(id: Int!): Boolean
  - addPokemonToInventory(input: InventoryInput!): Boolean
  - removePokemonFromInventory(input: InventoryInput!): Boolean
- Types
  - PokemonType
    - name: string*
    - baseAttack: int*
    - healthPoints: int*
    - weight: int*
    - owners: An array of PlayerType (*where the array cannot be null nor the items within the array*)
  - PlayerType
    - id: ID*
    - name: string*
    - inventory: An array of PokemonType (*where the array cannot be null nor the items within the array*)
  - BattleType
    - id: ID*
    - status: BattleStatus*
    - winner: PlayerType
    - battlePokemons: An array of PokemonType (*where the array cannot be null nor the items within the array*)
    - playersInMatch: An array of PlayerType (*where the array cannot be null nor the items within the array*)
    - attacks: An array of AttackType (*where the array cannot be null nor the items within the array*)
  - AttackType
    - damageDealt: int*
    - criticalHit: bool*
    - successfulHit: bool*
    - attackedBy: PokemonType

- Input types
  - BattleInputType
    - playerIds: An array of integers (*where the array cannot be null nor the items within the array*)
    - pokemonIds: An array of integers (*where the array cannot be null nor the items within the array*)
  - AttackInputType
    - attacker: string*
    - battleId: int*
  - PlayerInputType
    - name: string*
  - InventoryInputType
    - playerId: int*
    - pokemonIdentifier: string*
- Enum types
  - BattleStatus
    - NOT_STARTED
    - STARTED
    - FINISHED

Database schema (10%)



Submission

Submit a single compressed file (***.zip, *.rar**) in **Canvas**. If you are working in groups, please remember to add a comment with the name of the group members (*excluding the one who is submitting the file*).