Pokémon Battle

Welcome to the exciting world of fintech! We are an early-stage startup dedicated to creating seamless financial products specifically designed for Non-Resident Indians (NRIs). As we continue to innovate and expand, we are on the lookout for talented software engineers to join our dynamic team.

For your task, you will be building a Pokémon Battle Simulator. Get ready to showcase your skills in this fun challenge! This assignment is tailored for backend development, with a preference for Python and GoLang.

Dataset:

Please download the dataset from here. We will be using it for the assignment. You can find the description of the dataset on the url .

Battle Simulator Logic:

- Load the dataset from the CSV file.
- Implement a function to accept two Pokémon names as input.
    - Normalize the input by converting it to lowercase.
    - Check for spelling mistakes:
        - If there is a one-word spelling mistake, accept the input.
        - If there are more than one-word mistakes, throw an error.
- Parse the dataset to extract these fields for both Pokémon:

    - There are going to be 2 rounds:
        - Pokémon A will attack Pokémon B.
        - Pokémon B will attack Pokémon A.
    - For round 1, where Pokémon A will attack Pokémon B, extract the following for A & B (vice versa for round 2):
        - For Pokémon A:
            - `type1`: primary attack type of Pokémon, e.g., grass.
            - `type2`: secondary attack type of Pokémon, e.g., poison.
            - `attack`: attack value of Pokémon.
        - For Pokémon B:
            - Extract the `against_?` field for Pokémon B, where `?` represents the `type1` and `type2` of Pokémon A. For example, if Pokémon A's `type1` is grass, then extract `against_grass` for Pokémon B; if `type2` is poison, extract `against_poison` for Pokémon B.

- Calculate the damage percent given the value of both rounds:
    - For round 1, where Pokémon A will attack Pokémon B, the damage given by Pokémon A to B will be calculated by:

```
(attack(Pokémon A) / 200) * 100 - (((against_<type1> / 4)
* 100) + ((against_<type2> / 4) * 100))
```

    - Vice versa for round 2 where Pokémon B will attack Pokémon A.
    - If the damage given by Pokémon A is greater, then Pokémon A wins. If the damage given by Pokémon B is greater, then Pokémon B wins. If the score is the same, then it's a draw.

API Contract :

This is to be decided by you. You are free to use any request-response schema as long as it is an accepted standard. Here are our requirements:

We require 3 APIs:

- **API 1** will be a listing API with pagination.
- **API 2** will be the battle API, which will take Pokémon A name and Pokémon B name and will return a battle ID (in UUID4 format). The battle is going to be asynchronous, meaning when we hit the API, we should get a battle ID in response and the battle will start in the background. We can get the status of the battle using API 3.
- **API 3** will take the battle ID and will return the result of the battle in the following format:
  - If the battle is in progress, then return `BATTLE_INPROGRESS` as status with `null` as result.
  - If the battle has failed due to any exception, then return `BATTLE_FAILED` as status with `null` as result.
  - If the battle has been completed, then return `BATTLE_COMPLETED` as status and an object as result:

```
{
    "status": "BATTLE_INPROGRESS/BATTLE_COMPLETED/BATTLE_FAILED",
    "result": {
        "winnerName": "<Pokémon name which won>",
        "wonByMargin": "<Damage given by winning Pokémon>"
    }
}
```

Expectations :

- You are expected to write clean code using OOP principles . OOPs for python and for golang , use the interface and struct with proper embedding , Refer to this for golang LLD orchestration.
- Try using some design patterns to solve the problem.
- If stuck somewhere from a requirements perspective, make an assumption and move on.
- Use whatever framework you like.
- Deploy it wherever you like.
- We will be more focused on low-level design and the working of the code.
- TDD (Test-Driven Development) is compulsory; the expected code coverage is 80%.
- A UML class diagram is compulsory. Please attach it in the repository