Teste Pokémon

Requisitos Atualizados do Teste de Integração com a API de Pokémon

Objetivo

Avaliar a capacidade do desenvolvedor de integrar uma aplicação com a API de Pokémon, permitindo a seleção de dois Pokémons, identificação de seus tipos e comparação para determinar o vencedor em uma batalha.

Detalhamento dos Requisitos

1. **Criar um Endpoint**:
   - O desenvolvedor deve criar um endpoint `/battle` que aceite requisições POST com um JSON contendo os IDs de dois Pokémons. Por exemplo:

```
{
  "pokemon1": 1,
  "pokemon2": 2
}

```

2. **Fazer Chamada à API**:
   - O endpoint deve fazer uma chamada GET para a URL:
     - Para obter detalhes do Pokémon 1: `https://pokeapi.co/api/v2/pokemon/{pokemon1_id}`
     - Para obter detalhes do Pokémon 2: `https://pokeapi.co/api/v2/pokemon/{pokemon2_id}`
   - O desenvolvedor deve garantir que, se a chamada falhar, a aplicação retorne um erro apropriado.
3. **Obter Detalhes do Pokémon**:
   - Após receber a resposta da API, deve-se extrair a propriedade `types` de cada Pokémon, que é uma lista contendo a estrutura:

```
"types": [
  {
    "slot": 1,
    "type": {
      "name": "electric",
      "url": "https://pokeapi.co/api/v2/type/13/"
    }
  }
]

```

- A partir dessa estrutura, o desenvolvedor deve extrair o nome do tipo (por exemplo, `electric`).

4. **Regra de Vantagens**:
   - O desenvolvedor deve implementar a seguinte tabela de vantagens entre tipos:

Type	Strong Against	Weak Against
Fire	Grass, Bug, Ice	Water, Rock, Dragon
Water	Fire, Ground, Rock	Electric, Grass
Grass	Water, Ground, Rock	Fire, Flying, Bug
Electric	Water, Flying	Ground
Ground	Fire, Electric, Rock	Water, Grass
Flying	Grass, Fighting, Bug	Electric, Rock
Fighting	Normal, Rock, Ice	Fairy, Flying
Fairy	Fighting, Dragon, Dark	Steel, Fairy
Dark	Psychic, Ghost	Fairy
Psychic	Fighting, Poison	Bug, Ghost

- Se um tipo não tiver vantagem sobre o outro, deve-se declarar empate.

5. **Retornar Resultados**:
   - O endpoint deve retornar um JSON contendo:
     - Os nomes dos Pokémons selecionados.
     - Os resultados da comparação de tipos, por exemplo:

```
{
  "pokemon1": "pikachu",
  "pokemon2": "squirtle",
  "results": [
    "Electric is strong against Water. Pikachu wins!"
  ]
}

```

6. **Testes**:
   - O desenvolvedor deve implementar testes automatizados para verificar as seguintes funcionalidades:

    - A chamada à API retorna os detalhes corretos dos Pokémons.
    - A lógica de comparação de tipos funciona corretamente, incluindo empates.
    - O endpoint `/battle` responde adequadamente a entradas válidas e inválidas.
    - Testes de integração para garantir que a aplicação funcione como um todo.
7. **Documentação**:
    - O desenvolvedor deve criar um arquivo `README.md` que inclua:
        - Descrição do projeto e seu objetivo.
        - Instruções para instalação e execução do servidor Flask.
        - Instruções para testar o endpoint `/battle` usando `curl` ou ferramentas como `Postman`.
        - Exemplos de requisições e respostas do endpoint.
        - Detalhes sobre como executar os testes automatizados.
8. **Código no GitHub**:
    - O código deve ser hospedado em um repositório público no GitHub. O repositório deve incluir o `README.md` e a estrutura do projeto.

Análise do Código

Os seguintes aspectos devem ser analisados na implementação:

- **Estrutura do Código**: O código deve ser bem organizado, com funções claras e nomeadas de forma descritiva.
- **Tratamento de Erros**: Deve haver tratamento adequado de erros para chamadas à API e entrada do usuário.
- **Documentação**: O código deve ter comentários explicativos e um README.md que descreva como executar o projeto.
- **Testes**: Testar diferentes combinações de Pokémons para garantir que a lógica de comparação esteja correta.

Boas Práticas

- **Clareza e Simplicidade**: Escreva código que seja fácil de ler e entender.
- **Modularidade**: Separe a lógica em funções distintas para facilitar a manutenção e a reutilização do código.
- **Validação de Entrada**: Verifique se os IDs dos Pokémons são válidos antes de fazer chamadas à API.

- **Utilização de Design Patterns e Pricípios serão muito bem vindos:** Caso o candidato identifique a necessidade de um Patterns ou a utilização de um pricípio colocar no código