ALOM - TP 8 - High-Availability

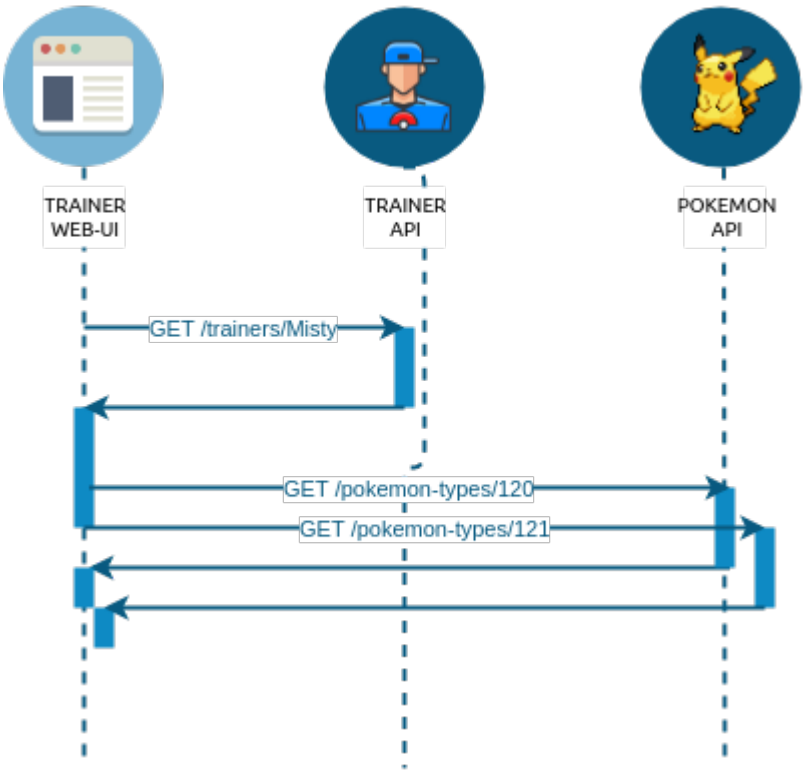
Table of Contents

1. Présentation et objectifs

Le but est de continuer le développement de notre architecture "à la microservice".

Nous allons aujourd'hui rendre les appels entre le `game-ui` et les autres micro-services plus performants en utilisant le parallélisme, tel que présenté dans le cours.

Nous allons aussi ajouter du cache pour améliorer les performances de nos services !



Pendant ce TP, nous faisons évoluer notre IHM `game-ui` ! Nous allons également commencer les développements du micro-service de combat !

Ce TP est moins guidé que d'habitude. Nous avons déjà toutes les bases nécessaires pour travailler de manière autonome.

1.1. Pré-requis

Les pré-requis à ce TP sont :

- Avoir terminé la partie *Pour aller plus loin* du TP 5 GUI

2. game-ui

2.1. Envoi de mails asynchrones

Développez dans `game-ui` un envoi de mail aux nouveaux trainers s'inscrivant dans l'application.

```
interface MailService {

    void sendWelcomeEmail(Trainer t);

}
```

```
class MailServiceImpl implements MailService {
```

```
    // TODO
}
```

Les envois de mails doivent :

- être asynchrones avec l'annotation `@Async`.

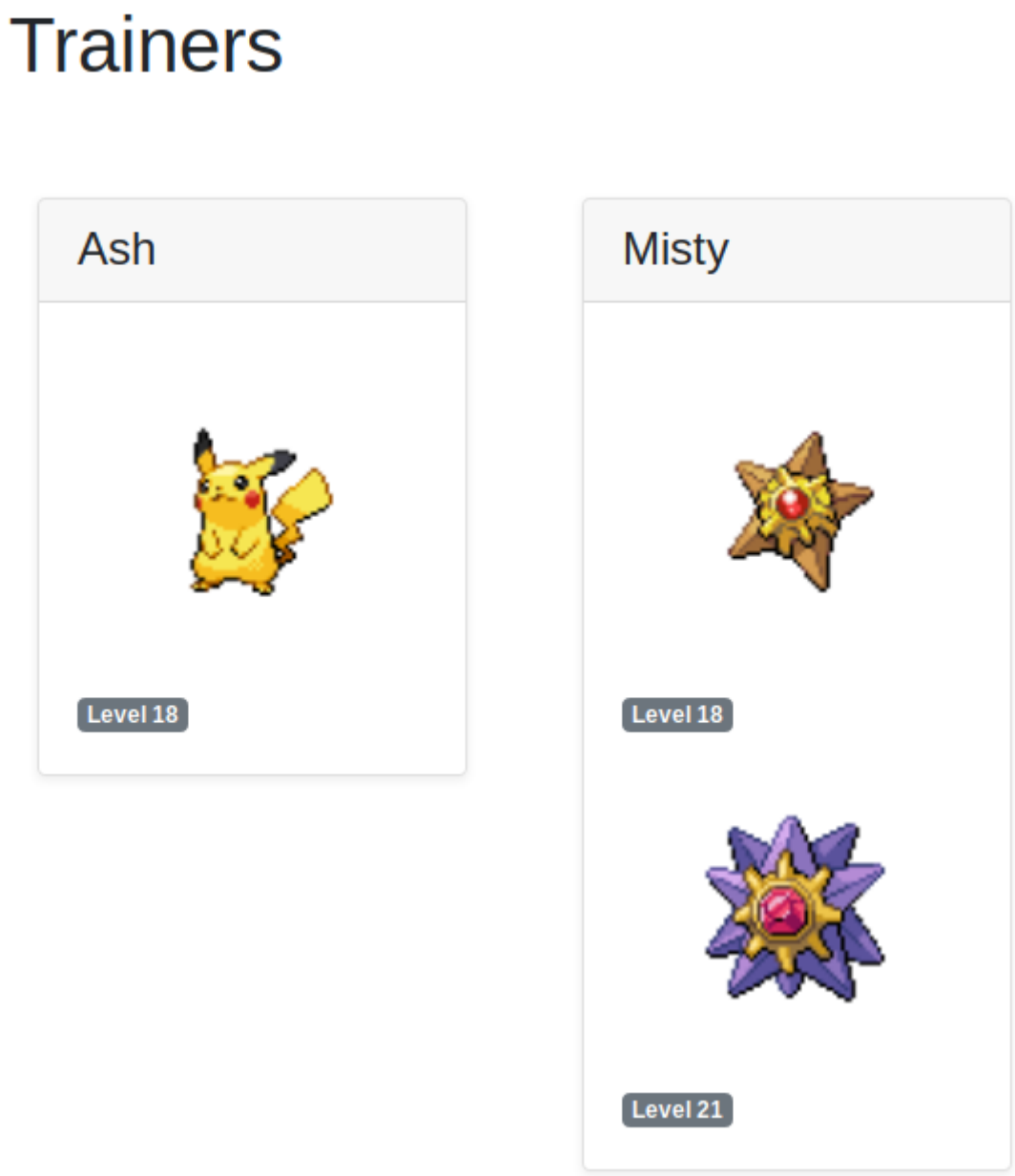
Nous n'allons pas nous brancher sur un réel serveur de mail. Votre `MailService` fera simplement un `System.out.println` pour simuler l'envoi pour l'instant. Nous développerons un micro-service d'envoi de mails dans un futur TP !

2.2. Page des trainers

Ajoutez dans votre IHM l'affichage de la liste des dresseurs de Pokemons, ainsi que leur équipe.

Cette partie était déjà proposée dans le TP 5

Cette liste pourra prendre la forme suivante :



2.3. Mise en place de cache

Ajoutez une gestion de cache sur le service qui récupère la liste des types de pokemon ainsi que la liste des dresseurs.

Le cache des dresseurs doit avoir une durée de vie assez courte (1 minute), parce qu'un dresseur peut faire évoluer son équipe !

Testez unitairement le bon fonctionnement de votre cache.

Voici pour vous aider un test unitaire que j'ai implémenté pour valider la bonne configuration de mon cache :

PokemonTypeServiceImplTest.java

```java
package com.miage.alom.tp.game_ui.config;

import com.miage.alom.tp.game_ui.pokemonTypes.PokemonType;
import com.miage.alom.tp.game_ui.pokemonTypes.PokemonTypeApiRepository;
import com.miage.alom.tp.game_ui.pokemonTypes.PokemonTypeService;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.cache.CacheManager;
import org.springframework.web.client.RestTemplate;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

@SpringBootTest ①
class PokemonTypeServiceImplTest {

    @Autowired
    PokemonTypeService pokemonTypeService;

    @MockBean ②
    PokemonTypeApiRepository apiRepository;

    @Autowired
    CacheManager cacheManager;

    @BeforeEach
    void setUp() {
        var pikachu = new PokemonType(25, "Pikachu");
        when(apiRepository.getPokemonType(25)).thenReturn(List.of(pikachu));
    }

    @Test
    void getPokemonType_shouldUseCache() {
        // first call, should call the mock
        pokemonTypeService.getPokemonType(25);

        // apiRepository should have been called once
        verify(apiRepository).getPokemonType(25);

        // second call, should use cache
        pokemonTypeService.getPokemonType(25);

        // apiRepository should not be called anymore because result is in cache !
③
```

```
49          verifyNoMoreInteractions(apiRepository);
50
51          // one result should be in cache !
52          var cachedValue = cacheManager.getCache("pokemon-types").get(25).get();
53          assertNotNull(cachedValue);
54          assertEquals(PokemonType.class, cachedValue.getClass());
55          assertEquals("Pikachu", ((PokemonType)cachedValue).name());
56      }
57  }
```

① Nous exécutons un test d'intégration qui démarre spring-boot

② Dans le test, nous remplaçons notre ApiRepository par un mock, qui nous permettra de vérifier s'il a été appelé

③ Nous validons que le cache est bien utilisé

2.4. Validation de vos développements

Pour vous amuser, vous pouvez tester vos développements avec une ou plusieurs de vos API éteintes pour voir ce qu'il se passe.

3. battle-api

Prenez un peu de temps pour finaliser les autres TP avant d'entamer cette partie !

Nous commençons dans ce TP le développement du service de combats, que nous continuerons sur les prochaines semaines !

3.1. Projet GitLab

Cliquez sur le lien suivant pour initialiser votre projet sur GitLab : GitLab Classroom

3.2. Stats des Pokemons

Les types de Pokemon ont des statistiques de base :

- vitesse
- attaque
- défense
- hp

Chaque Pokemon, en fonction de son niveau, aura des statistiques qui s'appuient sur ces statistiques de base. Pour les statistiques de vitesse, d'attaque et de défense, la statistique du pokemon est :

```
stat=5+(baseStat * (niveau) / 50)
```

Les points de vie du Pokemon sont calculés avec cette formule :

`stat=10+niveau+(baseStat * (niveau) / 50)`

Un pokemon de niveau 50 a les stats de base + 5, et un nombre de points de vie égal aux stats de base + 60. Un pokemon de niveau 100 a les stats de base * 2 + 5, et un nombre de points de vie égale à la stat de base * 2 + 110

Toutes les valeurs sont arrondies au nombre inférieur.

Pour donner un exemple concret :

Pikachu a les stats de base suivantes :

Table 1. Les stats de base de Pikachu

attack	55
defense	40
speed	90
hp	35

Un pikachu de niveau 5 a les stats suivantes :

Table 2. Quelques niveaux de pikachu

pikachu	**niveau 6**	**niveau 18**	**niveau 50**	**niveau 100**
attack	11	24	60	115
defense	9	19	45	85
speed	15	37	95	185
hp	20	40	95	180

3.3. Attaque et défense

Lors d'un combat, quand un pokémon en attaque un autre, il lui inflige des dégâts qui sont retirés des points de vie du pokemon attaqué.

La formule pour calculer les dégâts infligés par une attaque est :

La formule des dégats, avec `n` *le niveau du pokemon attaquant,* `a` *sa statistique d'attaque, et* `d` *la statistique de défense du pokemon adverse.*

`( ( (2*n)/5 + 2 * a / d ) + 2 )`

3.4. Règles du combat

Le combat se déroule en tour par tour.

Lors d'un tour, chaque dresseur de pokemon peut donner un ordre à son pokemon (attaquer), ou utiliser un objet (potion, etc.).

C’est le dresseur dont la stat de vitesse du pokemon est la plus élevée qui commence. Suivi de l’autre dresseur.

Si pendant un tour la vie de l’un des deux Pokemons tombe à 0, il est KO. C’est le pokemon suivant du dresseur qui prend la suite, et un nouveau tour commence.

3.5. Utilisation de l’API

Dans un premier temps, notre API de combat devra exposer les routes suivantes :

- `POST /battles` : Prend 2 paramètres (noms des 2 dresseurs en paramètres). Crée une instance de combat et retourne l’objet Battle permettant de l’identifier.
- `GET /battles` : liste les combats en cours
- `GET /battles/{uuid}` : Récupère l’état d’un combat en cours
- `POST /battles/{uuid}/{trainerName}/attack` : Permet à un dresseur de donner un ordre d’attaque pendant le combat. Retourne l’état du combat.
  1. Si le trainer attaque quand ce n’est pas son tour, renvoie une erreur `400 BAD REQUEST`

Le combat prend la forme suivante :

Le combat au format JSON

```
{
    "uuid": "781c2cc7-1681-4c6a-a94f-0445a0629453",
    "trainer": {
        "name": "Ash",
        "team": [
            {
                "id": 1,
                "type": {
                    "id": 25,
                    "name": "Pikachu",
                    "sprites": {
                        "back_default":
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/back/25.p
ng",
                        "front_default":
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/25.png"
                    }
                },
                "maxHp": 40,
                "attack": 24,
                "defense": 19,
                "speed": 37,
                "level": 18,
                "hp": 40,
                "ko": false
            }
        ],
```

```
        "nextTurn": true
    },
    "opponent": {
        "name": "Misty",
        "team": [
            {
                "id": 2,
                "type": {
                    "id": 120,
                    "name": "Staryu",
                    "sprites": {
                        "back_default":
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/back/120.
png",
                        "front_default":
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/120.png"
                    }
                },
                "maxHp": 38,
                "attack": 21,
                "defense": 24,
                "speed": 35,
                "level": 18,
                "hp": 38,
                "ko": false
            },
            {
                "id": 3,
                "type": {
                    "id": 121,
                    "name": "Starmie",
                    "sprites": {
                        "back_default":
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/back/121.
png",
                        "front_default":
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/121.png"
                    }
                },
                "maxHp": 56,
                "attack": 36,
                "defense": 40,
                "speed": 53,
                "level": 21,
                "hp": 56,
                "ko": false
            }
        ],
        "nextTurn": false
    }
```

```
69 }
```

Le calcul des dégats se fait bien côté serveur.

L'API battle doit donc :

- appeler l'API trainers pour récupérer les équipes des deux dresseurs lorsqu'un nouveau combat est créé
- stocker le combat (en mémoire pour commencer)
- appeler l'API PokemonTypes pour récupérer les statistiques de base des types de Pokemon et calculer les valeurs des statisques des Pokemons en fonction de leur niveau
- Lors d'un appel à `/attack`, effectuer une attaque entre les deux pokemons, en calculant les dégâts, et retourner le résultat

Il vous faudra faire évoluer l'API pokemon-type, pour exposer les statistiques des Pokemons. Les stats sont déjà présentes dans le fichier JSON de l'API pokemon-type.

3.6. UML

Voici un exemple de diagramme UML pour vous donner l'inspiration :)

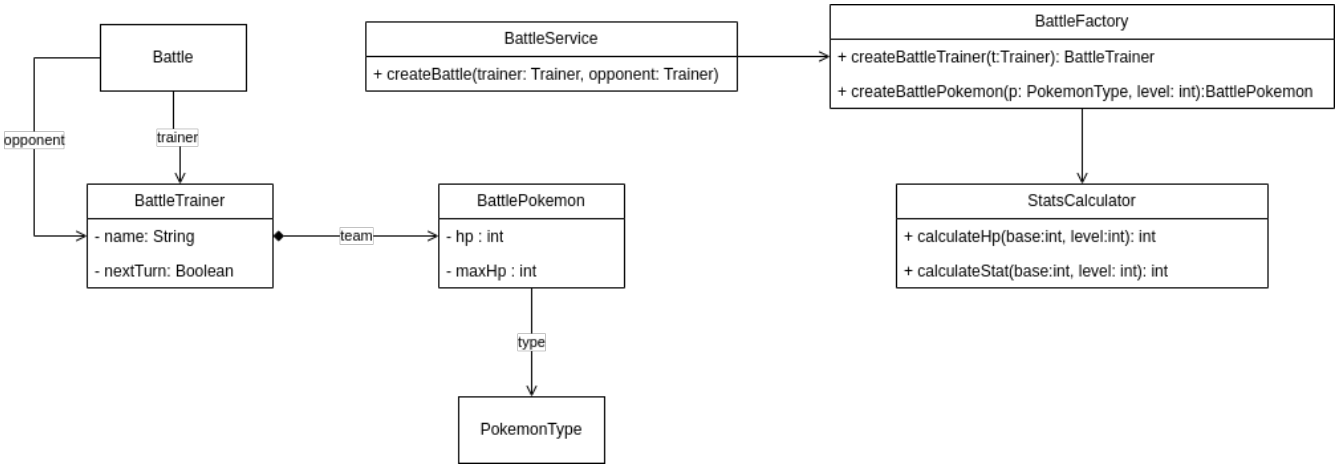


Figure 1. Battle UML