# ALOM - TP 6 - Interoperability

# Table of Contents

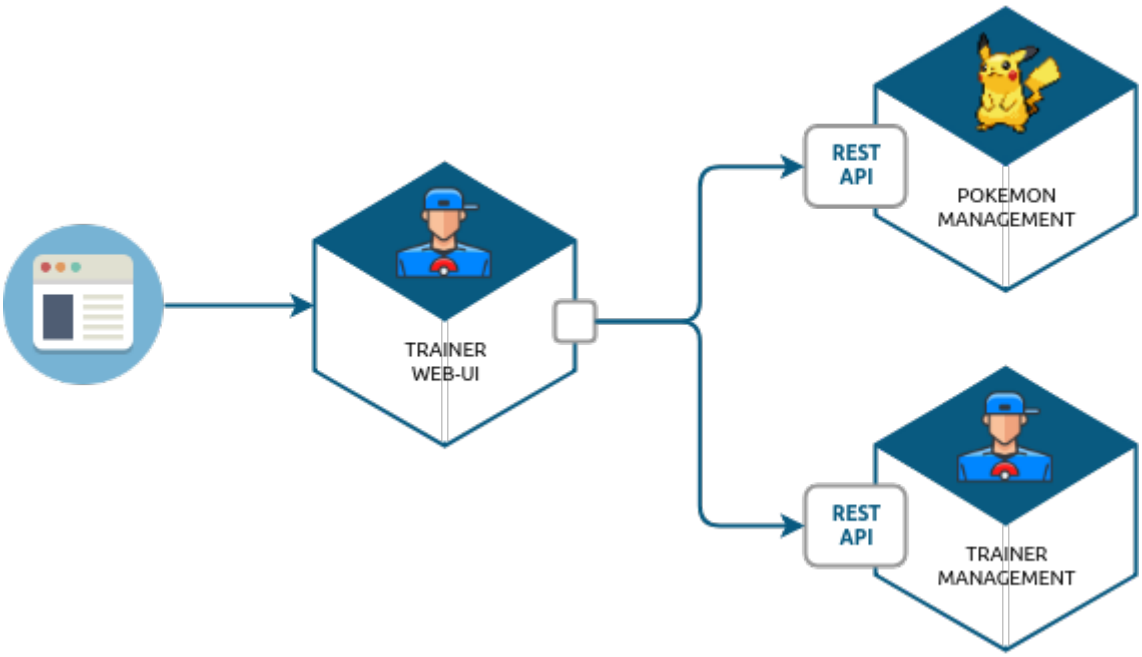


# 1. Présentation et objectifs

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

Nous allons aujourd'hui implémenter des fonctionnalités de traduction dans notre micro-service pokemon-type!

En effet, nos données de Pokemon sont aujourd'hui en anglais uniquement, ce qui peut être décourageant pour nos futurs joueurs français !




Nous allons développer :

1. La gestion des traductions dans notre api pokemon-type
2. L'affichage du Pokedex traduit !

Nous allons également réécrire nos appels utilisant le `RestTemplate` pour utiliser les *HTTP Interfaces*.

Nous ne repartons pas uniquement de zéro pour ce TP. Nous nous appuyons sur les TP précédents

# 2. pokemon-type-api

## 2.1. Les données de traduction

Pour faciliter le travail, j'ai créé deux fichier JSON contenant les traductions des noms de Pokemon en français et anglais.

Ces fichiers sont disponible ici: translations-fr.json translations-en.json

Déposez ces fichiers dans votre répertoire `src/main/resources`.

## 2.2. Le BO

Créez un record `Translation`

*com.miage.alom.tp.pokemon_type_api.Translation*

```
1 package com.miage.alom.tp.pokemon_type_api.bo;
2
```

```
3 public record Translation(int id, String name) {}
```

## 2.3. Le repository

### 2.3.1. L'interface

L'interface de ce repository de traduction est simple :

*com.miage.alom.tp.pokemon_type_api.TranslationRepository*

```
1 package com.miage.alom.tp.pokemon_type_api.repository;
2
3 import java.util.Locale;
4
5 public interface TranslationRepository {
6     String getPokemonName(int id, Locale locale);
7 }
```

### 2.3.2. Les tests unitaires

Implémentez les tests unitaires suivants :

*com.miage.alom.tp.pokemon_type_api.repository.TranslationRepositoryImplTest.java*

```
1  package com.miage.alom.tp.pokemon_type_api.repository;
2
3  import org.junit.jupiter.api.Test;
4  import org.springframework.context.annotation.AnnotationConfigApplicationContext;
5
6  import java.util.Locale;
7
8  import static org.junit.jupiter.api.Assertions.*;
9
10 class TranslationRepositoryImplTest {
11
12     private TranslationRepositoryImpl repository = new TranslationRepositoryImpl();
13
14     @Test
15     void getPokemonName_with1_inFrench_shouldReturnBulbizarre(){
16         assertEquals("Bulbizarre", repository.getPokemonName(1, Locale.FRENCH));
17         assertEquals("Bulbizarre", repository.getPokemonName(1, Locale.FRANCE));
18     }
19
20     @Test
21     void getPokemonName_with1_inEnglish_shouldReturnBulbizarre(){
22         assertEquals("Bulbasaur", repository.getPokemonName(1, Locale.ENGLISH));
23         assertEquals("Bulbasaur", repository.getPokemonName(1, Locale.UK));
24         assertEquals("Bulbasaur", repository.getPokemonName(1, Locale.US));
25     }
```

```
26
27      @Test
28      void applicationContext_shouldLoadPokemonRepository(){
29          var context = new
    AnnotationConfigApplicationContext("com.miage.alom.tp.pokemon_type_api");
30          var repoByName = context.getBean("translationRepositoryImpl");
31          var repoByClass = context.getBean(TranslationRepository.class);
32
33          assertEquals(repoByName, repoByClass);
34          assertNotNull(repoByName);
35          assertNotNull(repoByClass);
36      }
37
38  }
```

### 2.3.3. L'implémentation

Développez l'implémentation du `TranslationRepository`.

*com.miage.alom.tp.pokemon_type_api.TranslationRepositoryImpl.java*

```
1  package com.miage.alom.tp.pokemon_type_api;
2
3  import com.fasterxml.jackson.databind.ObjectMapper;
4  import com.miage.alom.tp.pokemon_type_api.bo.Translation;
5  import org.springframework.core.io.ClassPathResource;
6  import org.springframework.stereotype.Repository;
7
8  import java.io.IOException;
9  import java.util.List;
10 import java.util.Locale;
11 import java.util.Map;
12
13 @Repository
14 public class TranslationRepositoryImpl implements TranslationRepository {
15
16     record Key(Locale locale, int pokemonId){} ③
17
18     private Map<Key, Translation> translations;
19
20     private ObjectMapper objectMapper;
21
22     public TranslationRepositoryImpl() {
23         try {
24             // TODO ②
25         } catch (IOException e) {
26             e.printStackTrace();
27         }
28     }
29
```

```
30      @Override
31      public String getPokemonName(int id, Locale locale) {
32          // TODO ①
33      }
34  }
```

① Implémentez la récupération du nom d'un Pokemon !

② Alimentez la map des traductions en chargeant les fichiers, et en récupérant leur contenu

③ On utilise un record local à notre classe comme clé de Map !

La récupération d'un fichier dans le classpath peut se fair en Spring avec la classe `ClassPathResource`. Inspirez-vous du `PokemonTypeRepository` pour le reste.

## 2.4. Le service

Maintenant que nous avons un repository capable de gérer les traductions, nous devons les utiliser. Un bon endroit pour cela est la couche service.

Spring utilise la classe `AcceptHeaderLocaleResolver` dans la `DispatcherServlet` pour venir alimenter un objet `LocaleContextHolder`. Nous pouvons donc utiliser cet objet pour récupérer la langue demandée par la requête courante !

Ajoutez les tests unitaires suivant au `PokemonTypeServiceImplTest`:

*PokemonTypeServiceImplTest.java*

```
1  @Test
2  void pokemonNames_shouldBeTranslated_usingLocaleResolver(){
3      var pokemonTypeService = new PokemonTypeServiceImpl();
4
5      var pokemonTypeRepository = mock(PokemonTypeRepository.class);
6      pokemonTypeService.setPokemonTypeRepository(pokemonTypeRepository);
7      when(pokemonTypeRepository.findPokemonTypeById(25)).thenReturn(new
   PokemonType());
8
9      var translationRepository = mock(TranslationRepository.class);
10     pokemonTypeService.setTranslationRepository(translationRepository);
11     when(translationRepository.getPokemonName(25, Locale.FRENCH)).thenReturn
   ("Pikachu-FRENCH");
12
13     LocaleContextHolder.setLocale(Locale.FRENCH);
14
15     var pikachu = pokemonTypeService.getPokemonType(25);
16
17     assertEquals("Pikachu-FRENCH", pikachu.name());
18     verify(translationRepository).getPokemonName(25, Locale.FRENCH);
19 }
20
21 @Test
```

```
22 void allPokemonNames_shouldBeTranslated_usingLocaleResolver(){
23     var pokemonTypeService = new PokemonTypeServiceImpl();
24
25     var pokemonTypeRepository = mock(PokemonTypeRepository.class);
26     pokemonTypeService.setPokemonTypeRepository(pokemonTypeRepository);
27
28     var pikachu = new PokemonType(25, null, null, null);
29     var raichu = new PokemonType(26, null, null, null);
30     when(pokemonTypeRepository.findAllPokemonType()).thenReturn(List.of(pikachu,
   raichu));
31
32     // on simule le repository de traduction
33     var translationRepository = mock(TranslationRepository.class);
34     pokemonTypeService.setTranslationRepository(translationRepository);
35     when(translationRepository.getPokemonName(25, Locale.FRENCH)).thenReturn
   ("Pikachu-FRENCH");
36     when(translationRepository.getPokemonName(26, Locale.FRENCH)).thenReturn
   ("Raichu-FRENCH");
37
38     LocaleContextHolder.setLocale(Locale.FRENCH);
39
40     var pokemonTypes = pokemonTypeService.getAllPokemonTypes();
41
42     assertEquals("Pikachu-FRENCH", pokemonTypes.get(0).name());
43     assertEquals("Raichu-FRENCH", pokemonTypes.get(1).name());
44     verify(translationRepository).getPokemonName(25, Locale.FRENCH);
45     verify(translationRepository).getPokemonName(26, Locale.FRENCH);
46 }
```

Pour faire passer les tests unitaires, remplacez le nom du type de pokemon, après l'avoir récupéré du repository, par sa traduction.

Les records sont immutables, donc vous allez devoir trouver un moyen pour copier les données.

## 2.5. Le test d'intégration

Modifiez le `PokemonTypeControllerIntegrationTest` pour ajouter un test d'intégration :

*PokemonTypeControllerIntegrationTest.java*

```
1 @Test
2 void getPokemon_withId1_shouldReturnBulbasaur() {
3     var bulbasaur = this.restTemplate.getForObject("http://localhost:" + port +
   "/pokemon-types/1", PokemonType.class);
4     assertNotNull(bulbasaur);
5     assertEquals(1, bulbasaur.id());
6     assertEquals("Bulbasaur", bulbasaur.name()); ①
7 }
```

```
8
9  @Test
10 void getPokemon_withId1AndFrenchAcceptLanguage_shouldReturnBulbizarre() {
11     var headers = new HttpHeaders();
12     headers.setAcceptLanguageAsLocales(List.of(Locale.FRENCH)); ②
13
14     var httpRequest = new HttpEntity<>(headers);
15
16     var bulbizarreResponseEntity = this.restTemplate.exchange("http://localhost:" +
   port + "/pokemon-types/1", HttpMethod.GET, httpRequest, PokemonType.class);
17     var bulbizarre = bulbizarreResponseEntity.getBody();
18
19     assertNotNull(bulbizarre);
20     assertEquals(1, bulbizarre.id());
21     assertEquals("Bulbizarre", bulbizarre.name()); ③
22 }
```

① Cette requête sans paramètre particulier doit renvoyer la traduction par défaut (en anglais)

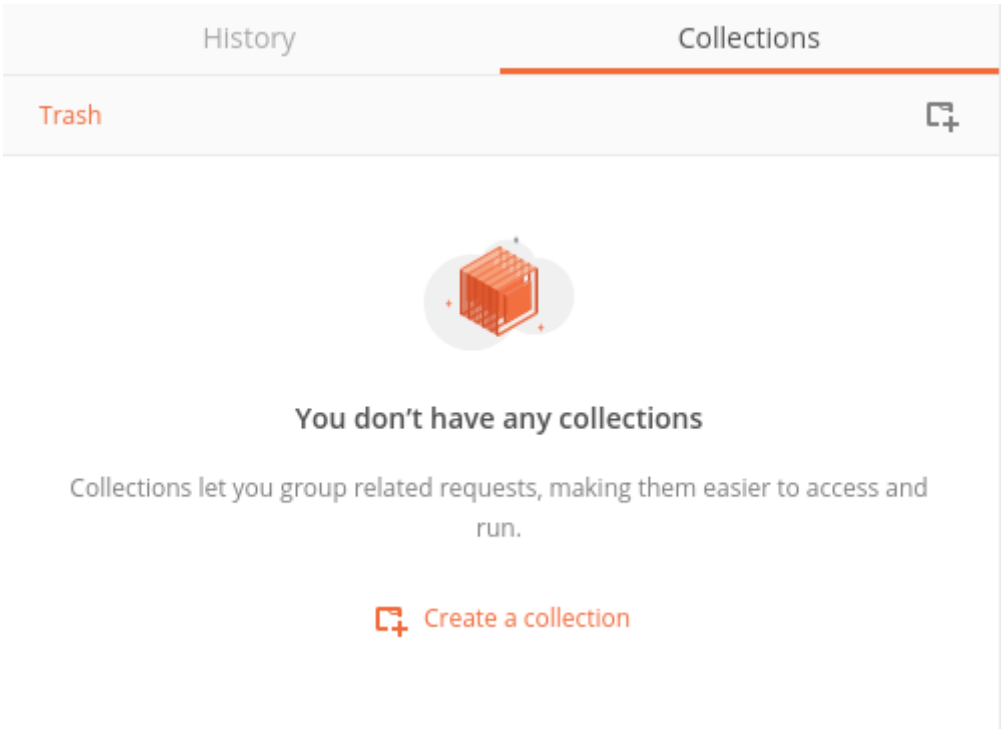
② On construit une requête en y ajoutant un header "Accept-Language"

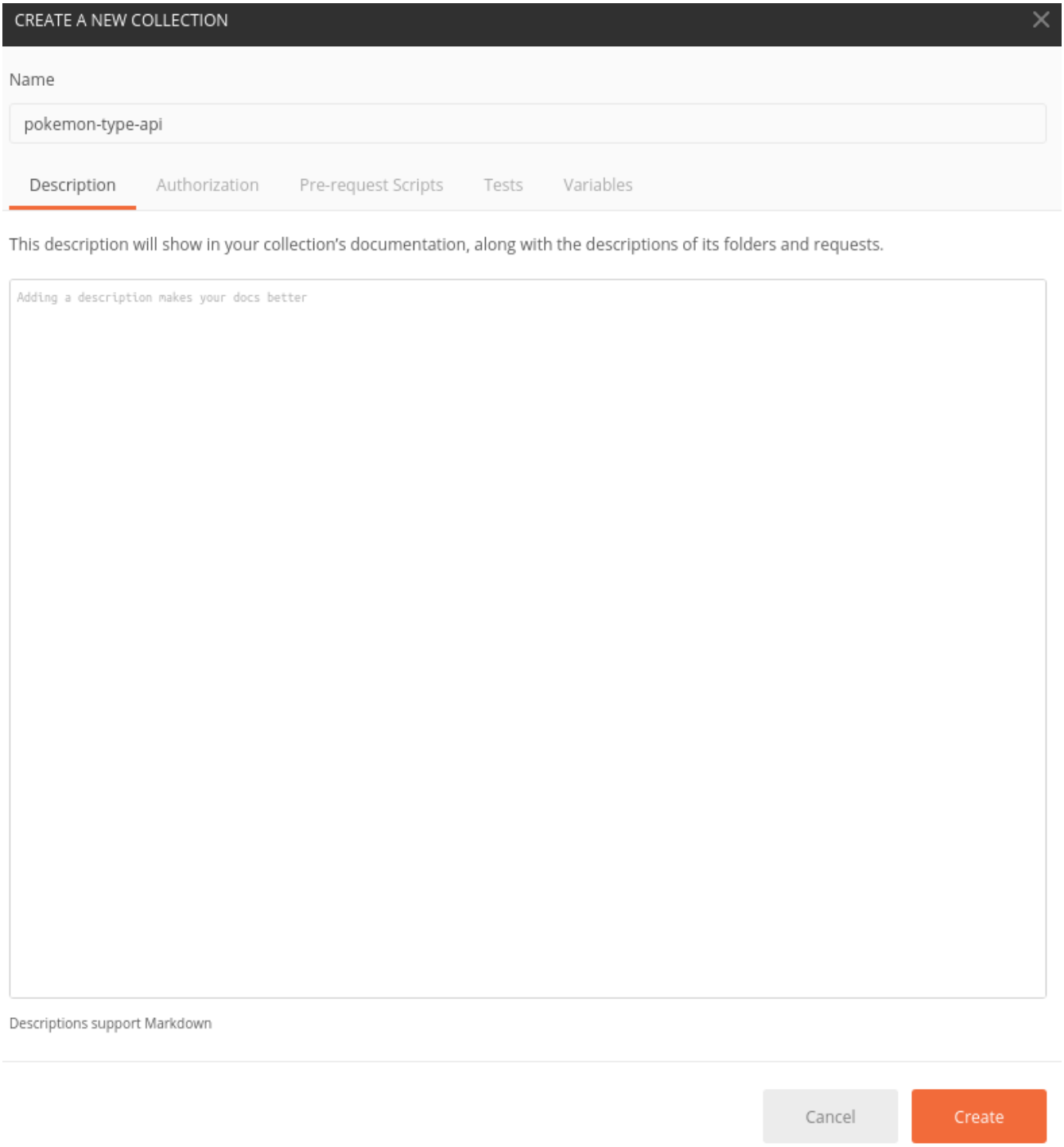
③ On doit bien récupérer le nom du type de Pokemon traduit !

## 2.6. Les tests avec Postman

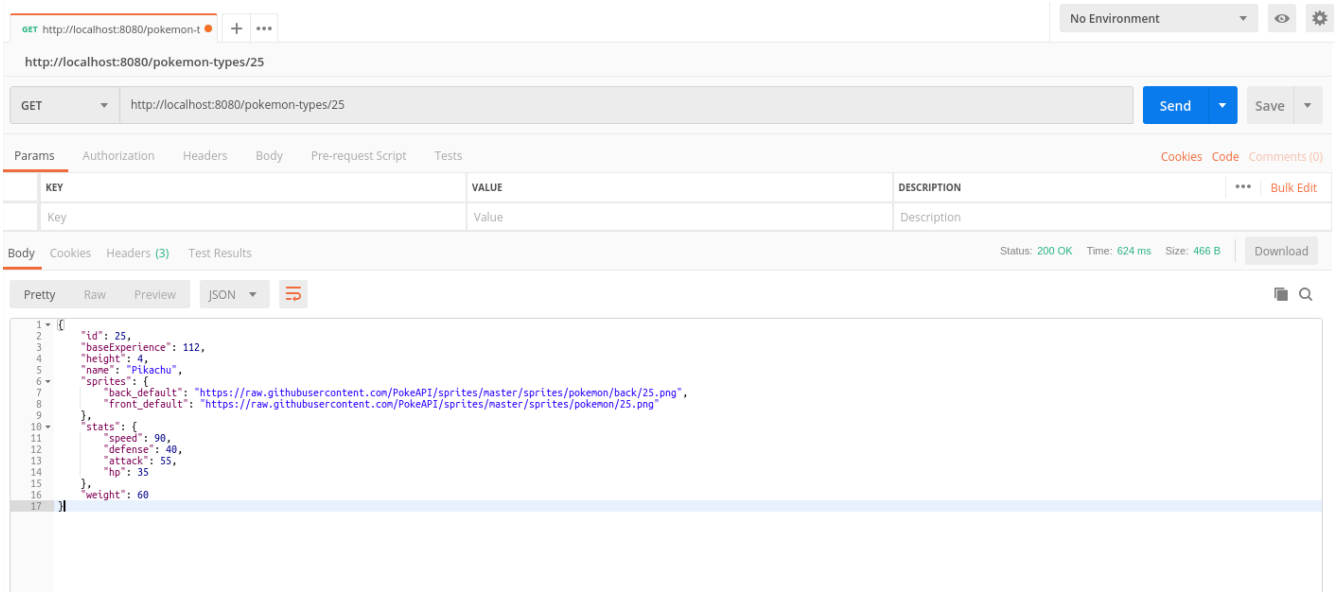
Pour bien valider nos développements, nous pouvons également créer des tests avec Postman.

Dans Postman, créez une `Collection`






Ajoutez-y quelques requêtes. Pour ce faire, créez une nouvelle requête, et enregistrez la dans votre collection.




Utilisez l'onglet `Tests` pour y ajouter quelques tests. Cet onglet permet d'exécuter du code javascript, permettant par exemple de valider les codes de retour HTTP ou le JSON reçu.

Créez les requêtes suivantes, avec les tests associés :

### 2.6.1. GET http://localhost:8080/pokemon-types/1

```
pm.test("Bulbasaur", function () {
    var bulbasaur = pm.response.json();
    pm.expect(bulbasaur.id).to.eq(1);
    pm.expect(bulbasaur.name).to.eq("Bulbasaur");
});
```

### 2.6.2. GET http://localhost:8080/pokemon-types/1 - Accept-Language: fr

```
pm.test("Bulbasaur", function () {
    var bulbasaur = pm.response.json();
    pm.expect(bulbasaur.id).to.eq(1);
    pm.expect(bulbasaur.name).to.eq("Bulbizarre");
});
```

### 2.6.3. GET http://localhost:8080/pokemon-types

```
pm.test("all pokemon types", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.length).to.eq(151);
});

pm.test("Bulbasaur", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData[0].name).to.eq("Bulbasaur");
```

```
});

pm.test("Ivysaur", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData[1].name).to.eq("Ivysaur");
});
```

### 2.6.4. GET http://localhost:8080/pokemon-types - Accept-Language: fr

```
pm.test("all pokemon types", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.length).to.eq(151);
});

pm.test("bulbizarre", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData[0].name).to.eq("Bulbizarre");
});

pm.test("Herbizarre", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData[1].name).to.eq("Herbizarre");
});
```

### 2.6.5. Export de la collection

Exportez votre collection Postman, dans le répertoire `src/test/resources` de votre API. Cela vous permettra de la réutiliser plus tard et de la partager avec les autres développeurs !

## 2.7. OpenApi et Swagger

Nous allons également exposer une interface de type *Swagger* afin de faciliter nos tests et nos développements.

Cette interface nous permettra également de donner aux consommateurs de notre API un moyen facile de voir les ressources disponibles et les tester !

Pour exposer un swagger, nous allons utiliser la librairie springdoc.

1. Cette librairie analyse les `Controlleurs` Spring, pour générer de la documentation au format swagger.

Cette librairie ne fait pas partie de Spring. Spring propose la génération de documentation à travers leur module spring rest-docs

Ajoutez la dépendance suivante à votre `pom.xml` :

*pom.xml*

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.2.0</version>
</dependency>
```

Votre IHM swagger sera disponible à l'url http://localhost:8080/swagger-ui.html, tandis que le JSON sera disponible à l'url http://localhost:8080/v2/api-docs.

Configurez *springdoc* pour n'afficher que vos propres controlleurs, et ignorer le *Basic Error Controller*.

Trouvez comment faire dans la doc de *springdoc* : https://springdoc.org/#springdoc-openapi-core-properties.

# 3. game-ui

## 3.1. Utilisation des HTTP Interfaces

Modifiez votre micro-service `game-ui` pour y intégrer la gestion de la locale!

Remplacez l'utilisation du `RestTemplate` par des *HTTP Interfaces* Spring.

Passez la Locale en paramètre lors de l'appel au micro-service *Pokemon Type*.

Vous pouvez récupérer la locale avec la méthode `LocaleContextHolder.getLocale()` de Spring directement dans le PokemonTypeServiceImpl du `game-ui`, et la transmettre en utilisant le header `Accept-Language`. . De cette manière, la langue utilisée lors des échanges sera celle du navigateur de l'utilisateur !

# 4. trainer-api

Implémentez sur l'API trainer :

1. l'exposition d'un swagger / open API
2. une collection Postman permettant de
    - récupérer la liste des dresseurs de Pokemon
    - récupérer un dresseur de Pokemon
    - créer un dresseur de Pokemon