Projet Pokematch

Marié Brayan

Liste des pokémons déjà recherchés, pour cela, nous avons mis en place une nouvelle classe RankedPokemon qui va nous permettre de savoir si un pokémon est classé et avec combien d'occurrence, cette classe possède un champ pokémon et un int. Donc dans une classe Result, nous allons avoir 2 deux champ :

- Une map de **Pokemon** vers **Interger**
- Une liste de **RankedPokemon**

Dans la map, il va y avoir tous les pokémons qui ont déjà été recherchés et dans la liste du classement au moment de l'affichage. Dès que nous allons rechercher un pokémon fétiche nous allons regarder dans votre map les pokémons, faire l'ajout correctement et ensuite mettre à jour notre liste triée pour pouvoir l'envoyer à notre controller pour que l'utilisateur puisse le voir.

Le cache est implémenté de la façon suivante, nous allons avoir une une Map de Pokémon vers un tableau de byte qui représente nos images. A chaque fois que nous aurons besoin de récupérer l'image d'un pokémon pour l'afficher nous irons regarder dans cette Map si les bytes sont déjà présentes. Si oui on envoie l'image dans le controller, sinon on envoie une requête à l'api et on envoie les bytes après avoir stocké les bytes pour le pokémon correspondant.

Pour la partie graphQL, le code a été repri d'un lien trouvé sur internet :

https://stackoverflow.com/questions/70519410/how-to-invoke-graphql-api-from-a-java-spring-boot-application-is-there-any-anno

Nous avons deux fichiers de query, l'un pour récupérer tous les pokémons et l'autre pour récupérer les images. Les images sont récupérées dans des **PokemonForm** et l'image est représentée par une **String** qui sera parsée à la main pour ensuite la transformer en tableau de bytes pour les afficher et les stocker en cache. Le traitement des requêtes GraphQL est fait dans la classe **RequestGraphQL**.

Mis à part le code pour la partie graphQL, tout le code a été produit pour moi.