TP2 : SpringBoot et SpringData

2.1 Spring Data JPA

Auto configuration par défaut de Spring JPA

Créer un projet avec

- une dépendance sur le starter ***Spring JPA***
- une dépendance sur ***hsqldb***

Récupérer les classes modèles fournies ainsi que le script sql.

Les classes Member et Document sont utilisées pour la partie JPA, la classe Customer pour la partie NoSQL

Si nécessaire ajouter une dépendance Maven ou Gradle

Configurer Hibernate afin qu'il montre les instructions SQL exécutées

Mettre au point un fichier *import.sql* qui insère des données de test en base

Démarrer l'application et vérifier que les insertions ont bien lieu

Faire un premier démarrage et observer la console

Définissez des interfaces *Repository* qui implémentent les fonctionnalités suivantes :

- CRUD sur Member et documents
- Rechercher tous les documents
- Trouver les membres ayant un email particulier
- Trouver le membre pour un email et un mot de passe donné
- Tous les membres dont le nom contient une chaîne particulière
- Rechercher tous les documents d'un membre à partir de son nom (Penser à utiliser l'annotation *@Query*)
- Compter les membres
- Compter les documents
- Trouver un membre à partir de son ID avec tous les documents associés pré-chargés

Modifier la classe de test générée par *SpringIntializr* pour vérifier que les méthodes effectuent les bonnes requêtes

Optionnellement :

Injecter un *EntityManager* ou un *Datasource* pour travailler directement au niveau de JPA ou JDBC

Configuration Datasource et pool de connexions

Ajouter une dépendance sur le driver PostgresSQL

Définir une base *postgres* utilisant un pool de connexions (maximum 10 connexions) dans un profil de production.

Créer une configuration d'exécution qui active ce profil

Implémentation Service

Implémenter une méthode métier qui permet d'ajouter un document à l'ensemble des utilisateurs de la base

Tester la méthode

2.2 Ajout d'un Repository MongoDB

Ajouter la dépendance sur MongoDB

Déclarer une classe modèle comme suit :

```
public class Customer {

    @Id
    public String id;

    public String firstName;
    public String lastName;

    public Customer() {}

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Override
    public String toString() {
        return String.format(
                "Customer[id=%s, firstName='%s', lastName='%s']",
                id, firstName, lastName);
    }
}
```

Définir une interface de type MongoRepository qui permet de recherche des classes Customer par les attributs firstName ou lastName

Au démarrage de l'application exécuter le code suivant :

```
private void _playWithMongo() {
            customerRepository.deleteAll();
```

```
		// save a couple of customers
		customerRepository.save(new Customer("Alice", "Smith"));
		customerRepository.save(new Customer("Bob", "Smith"));

		// fetch all customers
		System.out.println("Customers found with findAll():");
		System.out.println("-------------------------------");
		for (Customer customer : customerRepository.findAll()) {
			System.out.println(customer);
		}
		System.out.println();

		// fetch an individual customer
		System.out.println("Customer found with findByFirstName('Alice'):");
		System.out.println("--------------------------------");
		System.out.println(customerRepository.findByFirstName("Alice"));

		System.out.println("Customers found with findByLastName('Smith'):");
		System.out.println("--------------------------------");
		for (Customer customer : customerRepository.findByLastName("Smith"))
{
			System.out.println(customer);
		}

	}
```

Installer MongoDB pour tester ou ajouter la dépendance permettant d'avoir MongoDB en embarqué