**TP JDBC**

**Nous allons enfin exploiter la base de données!**

**Vous aurez besoin à terme des ressources ici** [**https://github.com/mKapusta/rest-api.git**](https://github.com/mKapusta/rest-api.git)

**1/ Créer une connexion à la base de données**

Commencer par ajouter la dépendance nécessaire à la suite du TP : spring-boot-starter-jdbc

Vous aurez également besoin de H2 : une base de données embarquée au sein de votre application.

groupId : com.h2database

artifcatId: h2

Reportez vous au cours pour créer la connexion à la base de données! (configuration)

Enfin, vous allez ajouter Flyway à votre application.

groupId : org.flywaydb

artificatId: flyway-core

version : 5.2.3

N’oubliez pas d’activer flyway dans la configuration.

Créer un répertoire pour ajouter les scripts SQL fournis. Pour rappel, ils sont à placer dans src/main/resources/db/migration.

Si tout est fonctionnel vous devriez voir dans les logs/console de votre IDE la création de la base et son remplissage par les scripts SQL fournis.

**2/ Création d’un premier accès : Programmation par contrat**

Créer une **interface** ProfesseurRepository. Cette interface propose une méthode pour chacune des opérations suivantes :

* Rechercher un professeur par son identifiant
* Rechercher un professeur par nom ou prenom
* Retourner tous les professeurs
* Créer un professeur
* Mettre à jour un professeur
* Supprimer un professeur.

Créer ensuite une **classe** ProfesseurRepositoryImpl qui implémentera chacune de ces méthodes.

Cette classe devra être annotée avec @Repository

Au sein de cette classe vous devrez, au choix, **INJECTER** un JdbcTemplate ou un NamedParameterJdbcTemplate pour effectuer vos requêtes.

**Concernant la recherche** : Il est possible que les deux critères soient utilisés (nom et prénom, un seul des deux, voire aucun)

**Concernant la modification et la suppression :** Une seule ligne sera modifiée à chaque fois. Utilisez bien l’identifiant fournie en “Path” de la ressource.

**Concernant les requêtes de lecture** : Utilisez le BeanPropertyRowMapper à la place du RowMapper pour les requêtes de lecture de la base.

**Concernant la création** : il est important de retourner l’identifiant auto généré du professeur.

Utiliser le code ci dessous pour y parvenir

KeyHolder keyHolder = new GeneratedKeyHolder();

PreparedStatement ps = null;

jdbcTemplate.update(new PreparedStatementCreator() {

@Override

public PreparedStatement createPreparedStatement(Connection connection) throws SQLException {

PreparedStatement ps = connection.prepareStatement(

"INSERT INTO PROFESSEUR(NOM,PRENOM) VALUES (?,?)", Statement.RETURN\_GENERATED\_KEYS);

ps.setString(1, professeur.getNom());

ps.setString(2, professeur.getPrenom());

return ps;

}

}, keyHolder);

return keyHolder.getKey().intValue();

Ce code retourne la valeur de l’ID auto généré, en plus de faire l’insertion du professeur.

**3/ Un RowMapper**

Cette fois vous allez créer un Repository de Cours. Le but est de remonter un cours par son identifiant avec le professeur qui le dispense.

L’interface ne proposera qu’une seule méthode à appeler dans votre Controller : getCoursWithProfesseur().

1. Ajouter une propriété de type Professeur au Cours si ce n’est pas déjà fait
2. Créer le Repository, son implémentation et le traitement au sein de la méthode. Vous aurez besoin d’une **Jointure** dans la requête pour lier Professeur et Cours.
3. Il vous faudra écrire un **RowMapper** (classe qui implémente **RowMapper<Cours>**). Elle convertit un ResultSet, le résultat d’une requête SQL faite avec JDBC, en un objet , ici un Cours. Votre RowMapper prendra la place du BeanPropertyRowMapper utilisé dans le cas des Professeurs.
4. Vérifiez que tout est fonctionnel en appelant le repository dans votre Controller des cours pour récupérer un cours par son identifiant.