

**DEPARTEMENT MATHEMATIQUES ET INFORMATIQUE**

# **compte rendu 5**

## **JEE**

**Filière :**  
**« Génie du Logiciel et des Systèmes Informatiques Distribués »**  
**GLSID**

**Le 18/03/2022**

**Préparé par : Ahlam CHAKRANE**

**Année Universitaire : 2020-2021**



## Introduction

Ce TP, vise à mettre en évidence la gestion des relation association entre les tables, et de maitriser mieux l'utilisation de JPA, et Spring.

Pour créer notre application nous sommes besoin d'une classe user et d'une classe Rôle.

La classe user est définie par ces propre attributs : id, username, password, et une liste des rôles

```
Seance5Appli... X User.java X Role.java UserService... applic
1 package ma.enset.Seance5.entities;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import javax.persistence.Column;
7 import javax.persistence.Entity;
8 import javax.persistence.FetchType;
9 import javax.persistence.Id;
10 import javax.persistence.ManyToMany;
11 import javax.persistence.Table;
12
13 import com.fasterxml.jackson.annotation.JsonProperty;
14 import com.fasterxml.jackson.annotation.JsonProperty.Access;
15
16 import lombok.AllArgsConstructor;
17 import lombok.Data;
18 import lombok.NoArgsConstructor;
19 @Entity
20 @Table(name = "users")
21 @Data @NoArgsConstructor @AllArgsConstructor
22 public class User {
23     @Id
24     @Column(length = 50)
25     private String id;
26     @Column(unique = true, length = 20)
27     private String username;
28     @JsonProperty(access = Access.WRITE_ONLY)
29     private String password;
30     @ManyToMany(mappedBy = "users", fetch = FetchType.LAZY)
31     private List<Role> roles= new ArrayList<>();
32 }
33
```

Dans la classe rôle on trouve : l'id, le nom du rôle, la description est la liste des utilisateurs qui ont le même rôle.

```
Seance5Appli... User.java Role.java X UserService... applic
1 package ma.enset.Seance5.entities;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import javax.persistence.Column;
7 import javax.persistence.Entity;
8 import javax.persistence.FetchType;
9 import javax.persistence.GeneratedValue;
10 import javax.persistence.GenerationType;
11 import javax.persistence.Id;
12 import javax.persistence.JoinColumn;
13 import javax.persistence.ManyToMany;
14
15 import com.fasterxml.jackson.annotation.JsonProperty;
16 import com.fasterxml.jackson.annotation.JsonProperty.Access;
17
18 import lombok.AllArgsConstructor;
19 import lombok.Data;
20 import lombok.NoArgsConstructor;
21 import lombok.ToString;
22 @Entity
23 @Data @AllArgsConstructor @NoArgsConstructor
24 public class Role {
25     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
26     private Long id;
27     @Column(unique = true, length = 20)
28     private String roleName;
29     @Column(name = "description") //au niveau mySQL il faut renommer
30     private String desc;
31     @ManyToMany(fetch = FetchType.LAZY)
32     //la table association
33     @JoinColumn(name = "Users_Roles")
34     //ne pas ecrire la liste des utilisateurs dans la fonction toString
35     @ToString.Exclude
36     @JsonProperty(access = Access.WRITE_ONLY)
37     private List<User> users= new ArrayList<>();
38
39 }
40
```

Les annotations jouent un rôle crucial dans les deux classes. L'annotation `@Data` permet de gérer automatiquement les getters et les setters.

L'annotation `@AllArgsConstructor` permet de générer un constructeur avec paramètre.

L'annotation `@NoArgsConstructor` permet de générer un constructeur sans paramètre.

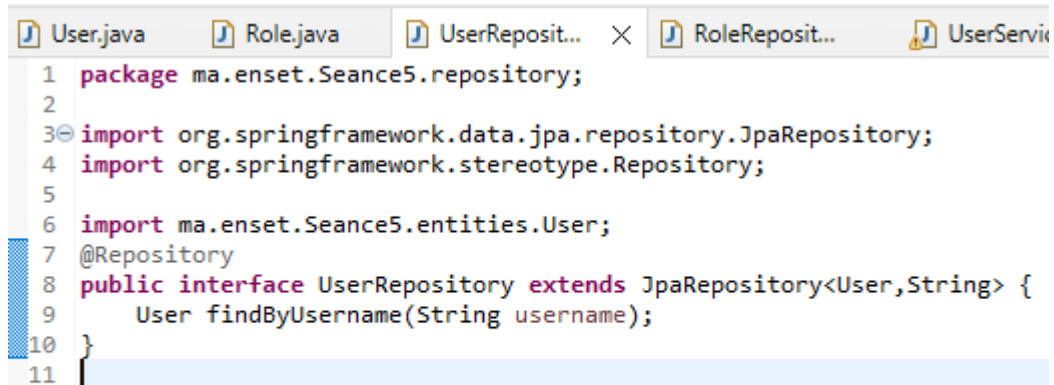
L'annotation `@Entity` permet de déclarer que la classe représente une entité de la base de données.

L'annotation `@Id` permet de spécifier l'id de la table

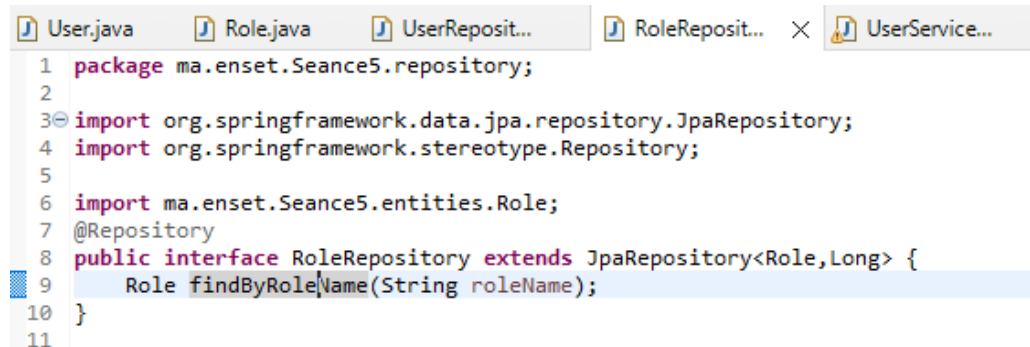
@JsonProperty permet de ne pas tourner en une boucle infinie lors de l'affichage des attributs de la classe user ou le contraire.

Puisque c'est une relation plusieurs à plusieurs, elle va générer une classe association, que nous l'avons la nommé : Users\_Roles.

Et pour profiter des fonctionnalités offertes par JPA, nous avons créé deux classe repository pour les deux interfaces suivantes, qui nous offrir des fonctions pour effectuer des opérations DML sur la base de données :



```
1 package ma.enset.Seance5.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 import ma.enset.Seance5.entities.User;
7 @Repository
8 public interface UserRepository extends JpaRepository<User,String> {
9     User findByUsername(String username);
10 }
11
```



```
1 package ma.enset.Seance5.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 import ma.enset.Seance5.entities.Role;
7 @Repository
8 public interface RoleRepository extends JpaRepository<Role,Long> {
9     Role findByRoleName(String roleName);
10 }
11
```

Et pour faire appel à ces fonctions, et d'autres fonctions, nous avons créé les interfaces Services suivantes :

```
User.java × Role.java UserReposit... RoleReposit... UserService.... ×
1 package ma.enset.Seance5.service;
2
3 import ma.enset.Seance5.entities.User;
4
5 public interface UserService {
6     User addUser(User user);
7     User findUserByUsername(String username);
8     void addRoleToUser(String username, String rolename);
9     User authenticate(String username, String password);
10 }
11
```

```
RoleService.java ×
1 package ma.enset.Seance5.service;
2
3 import ma.enset.Seance5.entities.Role;
4
5 public interface RoleService {
6     Role addRole(Role role);
7     Role findRoleByRoleName(String roleName);
8 }
9
```

Pour utiliser ces interfaces, nous avons créé des implémentations pour eux :

UserServiceImpl.java × application.properties Seance5/pom.xml UserCo

```
13 import ma.enset.Seance5.repository.UserRepository;
14
15 @Service
16 @Transactional
17 @AllArgsConstructor
18 public class UserServiceImpl implements UserService {
19
20     private UserRepository userRepository;
21     private RoleService roleService;
22
23     @Override
24     public User addUser(User user) {
25         user.setId(UUID.randomUUID().toString());
26         return userRepository.save(user);
27     }
28
29     @Override
30     public User findUserByUsername(String username) {
31         return userRepository.findByUsername(username);
32     }
33
34     @Override
35     public void addRoleToUser(String username, String roleName) {
36         User user= findUserByUsername(username);
37         Role role= roleService.findRoleByRoleName(roleName);
38         user.getRoles().add(role);
39         role.getUsers().add(user);
40     }
41
42     @Override
43     public User authenticate(String username, String password) {
44         User user = userRepository.findByUsername(username);
45
46         if(user==null) throw new RuntimeException("Bad credentials");
47         if(user.getPassword().equals(password)) return user;
48         throw new RuntimeException("Bad credentials");
49     }
50
51 }
```



```
RoleService.java  RoleServiceImpl.java X
1 package ma.enset.Seance5.service;
2
3 import javax.transaction.Transactional;
10 @Service
11 @Transactional
12 @AllArgsConstructor
13 public class RoleServiceImpl implements RoleService{
14     private RoleRepository roleRepository;
15     @Override
16     public Role addRole(Role role) {
17         return roleRepository.save(role);
18     }
19
20     @Override
21     public Role findRoleByRoleName(String roleName) {
22         return roleRepository.findByName(roleName);
23     }
24
25 }
26
```

Nous avons créé également un contrôleur pour tester les routes. La route appelle une route qui permet de récupérer un utilisateur après avoir mentionné son nom d'utilisateur dans la route déclarée :

```
RoleService.java  RoleServiceImpl.java  UserController.java X
1 package ma.enset.Seance5.web;
2
3 import org.springframework.beans.factory.annotation.Autowired;
10
11 @RestController
12 public class UserController {
13     @Autowired
14     private UserService userService;
15     @GetMapping("/users/{username}")
16     public User user(@PathVariable String username) {
17         User user = userService.findUserByUsername(username);
18         return user;
19     }
20 }
21
```

Et alors pour tester le fonctionnement de ces méthodes, nous avons les appelés dans la classe principale :

```
application.properties  Seance5Application.java X
18 public static void main(String[] args) {
19     SpringApplication.run(Seance5Application.class, args);
20 }
21
22 @Bean
23 CommandLineRunner start(UserService userService, RoleService roleService) {
24     return args -> {
25         User user= new User();
26         user.setUsername("Ahlam");
27         user.setPassword("12345");
28         userService.addUser(user);
29
30         User user2= new User();
31         user2.setUsername("Ahlam2");
32         user2.setPassword("ER12345");
33         userService.addUser(user2);
34
35         Stream.of("STUDENT", "USER", "ADMIN").forEach(r->{
36             Role role1= new Role();
37             role1.setDesc("role "+r);
38             role1.setRoleName(r);
39             roleService.addRole(role1);
40         });
41         userService.addRoleToUser("Ahlam", "STUDENT");
42         userService.addRoleToUser("Ahlam", "USER");
43         userService.addRoleToUser("Ahlam", "ADMIN");
44         userService.addRoleToUser("Ahlam2", "STUDENT");
45
46         try {
47             User u= userService.authenticate("Ahlam", "12345");
48             System.out.println(u.getId());
49             System.out.println(u.getUsername());
50             u.getRoles().forEach(r->{
51                 System.out.println("Role=> "+r.getRoleName());
52             });
53
54         } catch (Exception e) {
55             System.err.println("Ooops");
56         }
57     }
58 }
```

Et bien évidemment, il faut déclarer le port, le nom de la base de données et d'autres paramètres dans le fichier application.properties.

```
application.properties x
1 #spring.h2.console.enabled=true
2 #spring.datasource.url=jdbc:h2:mem:users_db
3 #ajouter la dépendance de sql dans pom.xml
4 spring.datasource.url=jdbc:mysql://localhost:3306/users_db?createDatabaseIfNotExist=true
5 spring.datasource.username=root
6 spring.datasource.password=
7 server.port=8088
8 spring.jpa.hibernate.ddl-auto = create
9 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MariaDBDialect
10 spring.jpa.show = true
11
```

Après avoir tourné l'application par H2 Database, voici les tables créées :

English Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:users\_db

User Name: sa

Password:

Connect Test Connection

Table utilisateurs :

jdbc:h2:mem:users\_db

- ROLE
- ROLE\_USERS
- USERS
- INFORMATION\_SCHEMA
- Sequences
- Users

H2 1.4.200 (2019-10-14)

Run Run Selected Auto complete Clear SQL statement:

SELECT \* FROM USERS |

SELECT \* FROM USERS;

ID	PASSWORD	USERNAME
4f6a0154-b07a-4c2b-a510-ba332a3e1c22	12345	Ahlam
8b37500a-a3f3-41f0-8ea1-acb8248fb9c6	ER12345	Ahlam2

(2 rows, 6 ms)

Edit

Tables rôles :

jdbc:h2:mem:users\_db

- ROLE
- ROLE\_USERS
- USERS
- INFORMATION\_SCHEMA
- Sequences
- Users

H2 1.4.200 (2019-10-14)

Run Run Selected Auto complete Clear SQL statement:

SELECT \* FROM ROLE |

SELECT \* FROM ROLE;

ID	DESC	ROLE_NAME
1	role STUDENT	STUDENT
2	role USER	USER
3	role ADMIN	ADMIN

(3 rows, 7 ms)

Edit

Table Users\_Roles :

jdbc:h2:mem:users\_db  
+ ROLE  
+ ROLE\_USERS  
+ USERS  
+ INFORMATION\_SCHEMA  
+ Sequences  
+ Users  
i H2 1.4.200 (2019-10-14)

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM ROLE_USERS
```

---

```
SELECT * FROM ROLE_USERS;
```

ROLES_ID	USERS_ID
2	4f6a0154-b07a-4c2b-a510-ba332a3e1c22
3	4f6a0154-b07a-4c2b-a510-ba332a3e1c22
1	4f6a0154-b07a-4c2b-a510-ba332a3e1c22
1	8b37500a-a3f3-41f0-8ea1-acb8248fb9c6

(4 rows, 6 ms)

Avec MySQL, voici les tables cr  es :

Current server: MySQL

Recent Favorites

- New
- information\_schema
- mysql
- performance\_schema
- sys
- users\_db
  - New
  - role
  - role\_users
  - users

Showing rows 0 - 2 (3 total, Query took 0.0007 seconds.)

```
SELECT * FROM `role`
```

☐ Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

☐ Show all | Number of rows: 25 | Filter rows: Search this table

+ Options

				id	description	role_name
<input type="checkbox"/>	Edit	Copy	Delete	1	role STUDENT	STUDENT
<input type="checkbox"/>	Edit	Copy	Delete	2	role USER	USER
<input type="checkbox"/>	Edit	Copy	Delete	3	role ADMIN	ADMIN

↑ ☐ Check all With selected: Edit Copy Delete

☐ Show all | Number of rows: 25 | Filter rows: Search this table

Query results operations

Si on test la route avec le nom d'utilisateur Ahlam voici le résultat trouvé :

← → ↻ ⓘ localhost:8088/users/Ahlam

```
{"id":"d07cbebe-f7bb-46d5-8d3a-ebd19ea847ef","username":"Ahlam","roles":[{"id":1,"roleName":"STUDENT","desc":"role STUDENT"}]}
```

## Conclusion

Grace à ce TP, nous avons pu comprendre comment gérer les relations plusieurs à plusieurs entre les tables, de plus nous avons mieux développer ce que nous avons pris dans ce module de plus en plus.