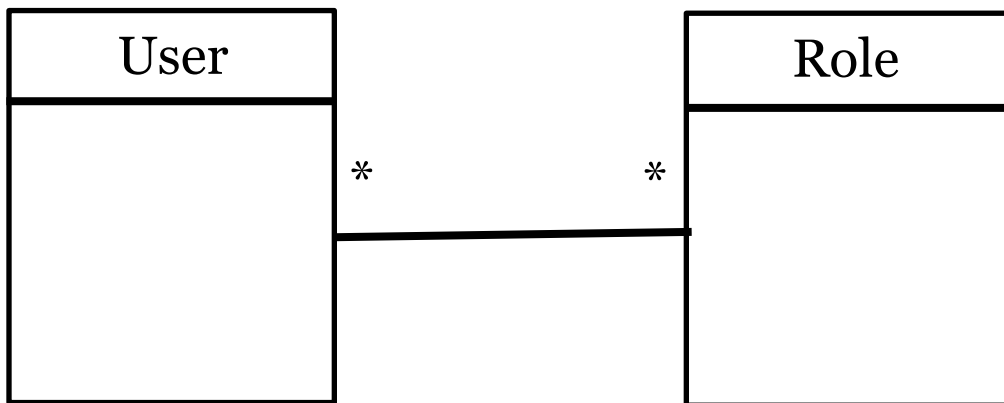


**Cas Many to many
Users/Roles**

AZEHAF Issam

Introduction

Ce tp consiste a traiter le cas Many to Many entre deux entité User et Role, Chaque user peut avoir plusieurs roles, et chaque role peut concerner plusieurs user,



Demonstration

Tout d'abord on va commencer par la création des entités User et Role et d'effectue le Mapping entre eux

```
UserRolesApplication.java x User.java x Role.java x
1 package ma.azehaf.user_roles.Entities;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 import javax.persistence.*;
8 import java.util.ArrayList;
9 import java.util.List;
10
11 @Entity
12 @Data @AllArgsConstructor @NoArgsConstructor
13 public class User {
14     @Id
15     private String userId;
16     private String userName;
17     private String password;
18     @ManyToMany(mappedBy = "users", fetch = FetchType.EAGER)
19     private List<Role> roles = new ArrayList<>();
20 }
21
```

Figure 1 : Classe User

```

1  package ma.azehaf.user_roles.Entities;
2
3  import lombok.AllArgsConstructor;
4  import lombok.Data;
5  import lombok.NoArgsConstructor;
6
7  import javax.persistence.*;
8  import java.util.ArrayList;
9  import java.util.List;
10
11 @Entity
12 @Data @AllArgsConstructor @NoArgsConstructor
13 public class Role {
14     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
15     private Long id;
16     private String desc;
17     private String roleName;
18     @ManyToMany(fetch = FetchType.EAGER)
19     @JoinTable(name = "USERS_ROLES")
20     private List<User> users = new ArrayList<>();
21 }
22

```

Figure 2 : Classe Role

@Entity : Signifie que la classe est une entité JPA

@Data : C'est une annotation Lombok qui crée les setters et les getters

@AllArgsConstructor : C'est une annotation Lombok qui crée un constructeur avec paramètre

@NoArgsConstructor : C'est une annotation Lombok qui crée un constructeur sans paramètre au cas de besoin

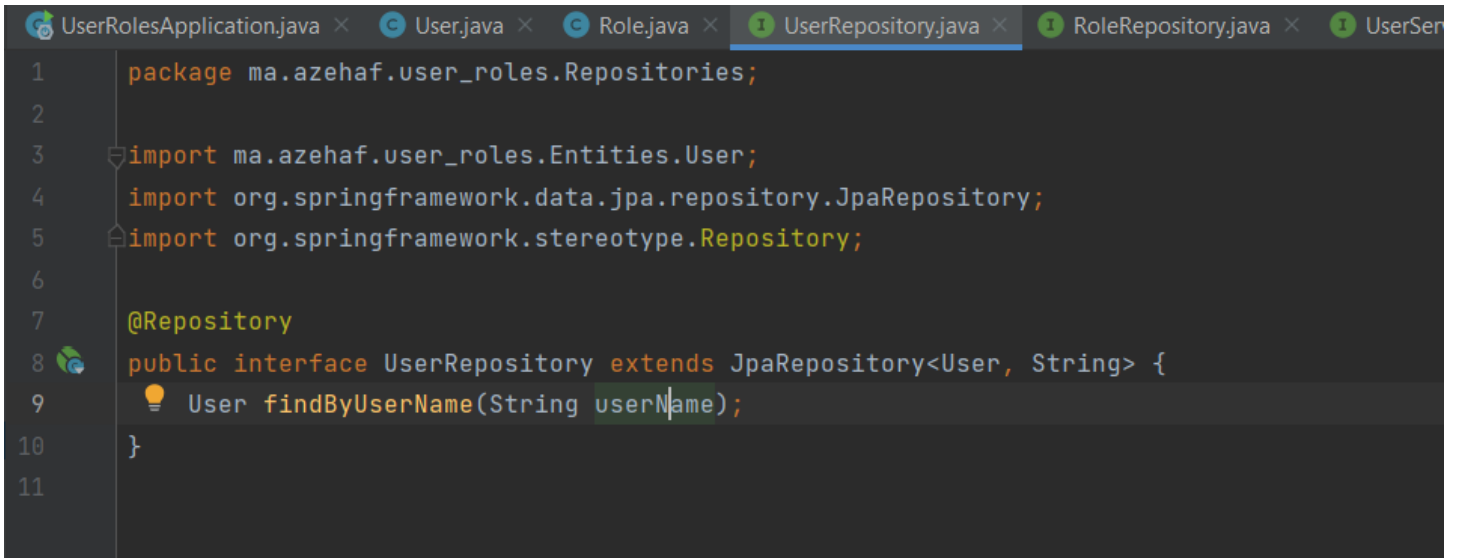
La classe User contient 3 attributs (Id, userName, password) et une liste des roles

La classe Role contient 3 attributs (Id, Desc, roleName) et une liste des users

@Id : est une annotation JPA qui montre que l'attribut est la clé primaire de la table

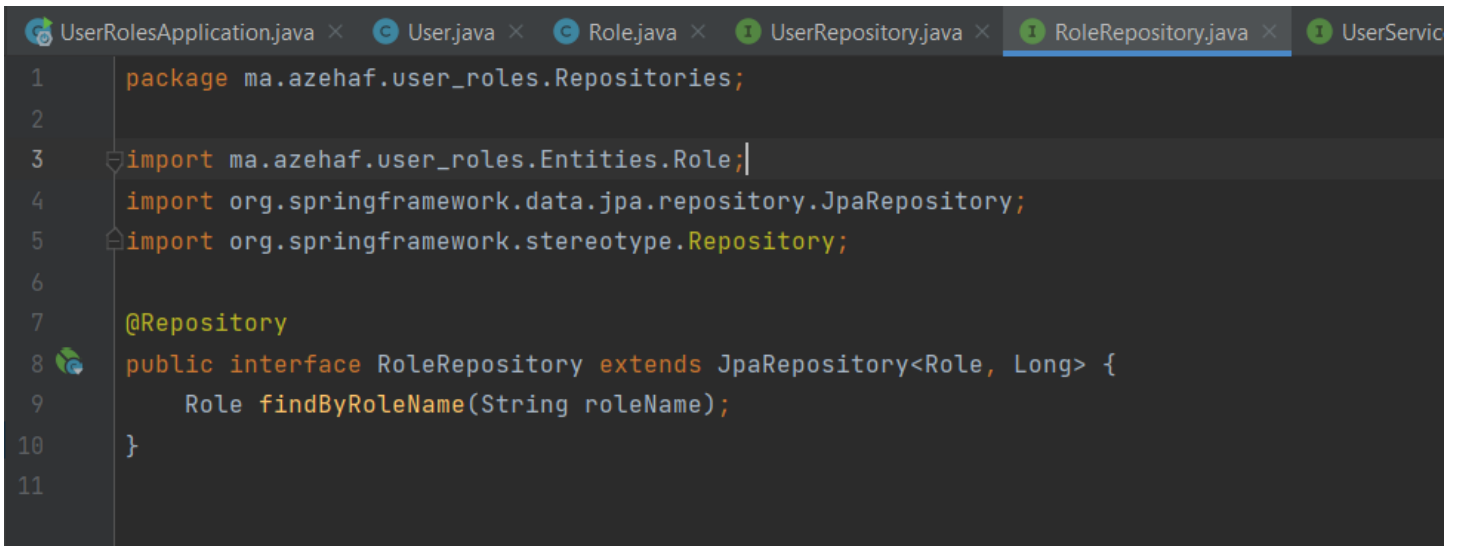
@ManyToMany : c'est pour montrer le mapping entre les deux classes comme quoi c'est une association ManyToMany

Après j'ai créer deux interface UserRepository et RoleRepository qui herite de JpaRepository, c'est du Spring Data



```
1 package ma.azehaf.user_roles.Repositories;
2
3 import ma.azehaf.user_roles.Entities.User;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 @Repository
8 public interface UserRepository extends JpaRepository<User, String> {
9     User findByUserName(String userName);
10 }
11
```

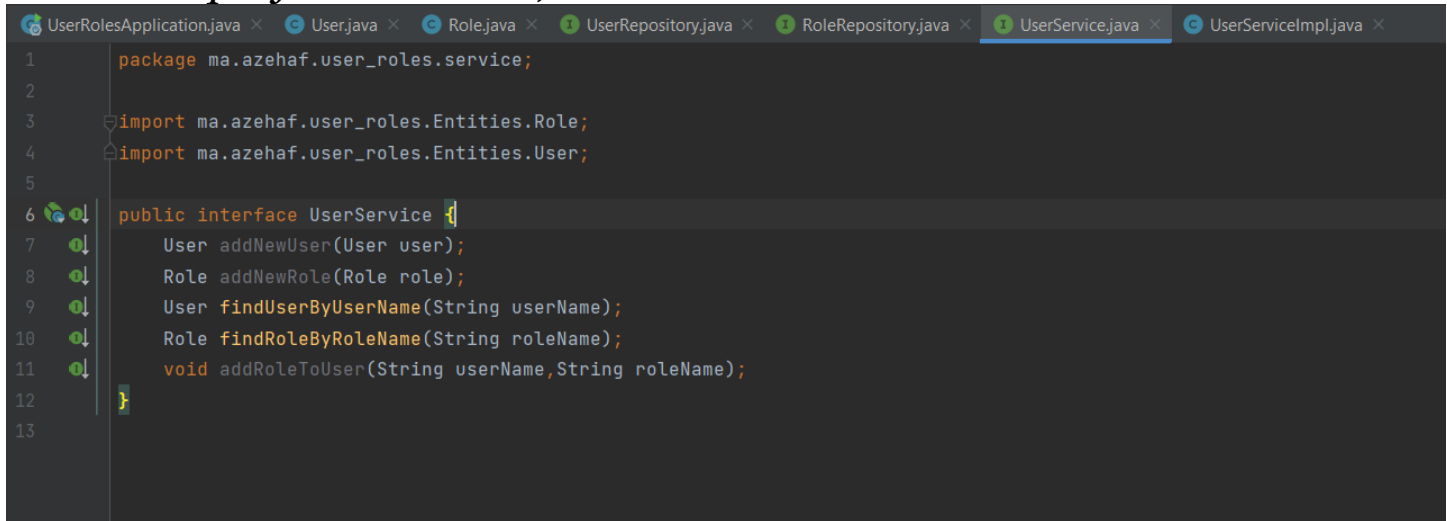
Figure 3 : Interface UserRepository



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

Figure 4 : Interface RoleRepository

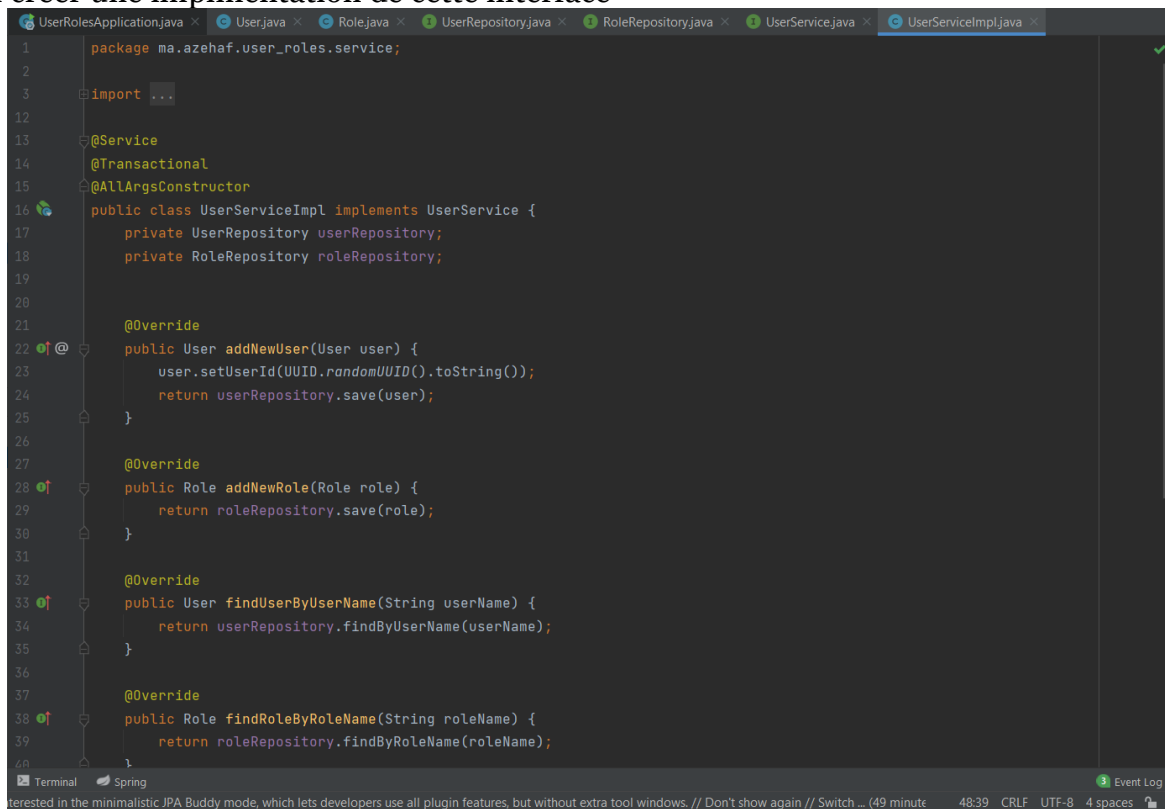
Dans la couche Metier j'ai créer une interface qui contient tous les méthodes que j'aurai besoin,



```
1 package ma.azehaf.user_roles.service;
2
3 import ma.azehaf.user_roles.Entities.Role;
4 import ma.azehaf.user_roles.Entities.User;
5
6 public interface UserService {
7     User addNewUser(User user);
8     Role addNewRole(Role role);
9     User findUserByUserName(String userName);
10    Role findRoleByRoleName(String roleName);
11    void addRoleToUser(String userName,String roleName);
12 }
13
```

Figure 5 : Interface UserService

Après j'ai créer une implmentation de cette interface



```
1 package ma.azehaf.user_roles.service;
2
3 import ...
4
5 @Service
6 @Transactional
7 @AllArgsConstructor
8 public class UserServiceImpl implements UserService {
9     private UserRepository userRepository;
10    private RoleRepository roleRepository;
11
12    @Override
13    public User addNewUser(User user) {
14        user.setUserId(UUID.randomUUID().toString());
15        return userRepository.save(user);
16    }
17
18    @Override
19    public Role addNewRole(Role role) {
20        return roleRepository.save(role);
21    }
22
23    @Override
24    public User findUserByUserName(String userName) {
25        return userRepository.findByUserName(userName);
26    }
27
28    @Override
29    public Role findRoleByRoleName(String roleName) {
30        return roleRepository.findByRoleName(roleName);
31    }
32 }
33
```

Figure 6 : Classe UserServiceImpl

Cette Classe contient l'implementation des metthode déclaré dans l'interface

```
@Override
public User addNewUser(User user) {
    user.setUserId(UUID.randomUUID().toString());
    return userRepository.save(user);
}
```

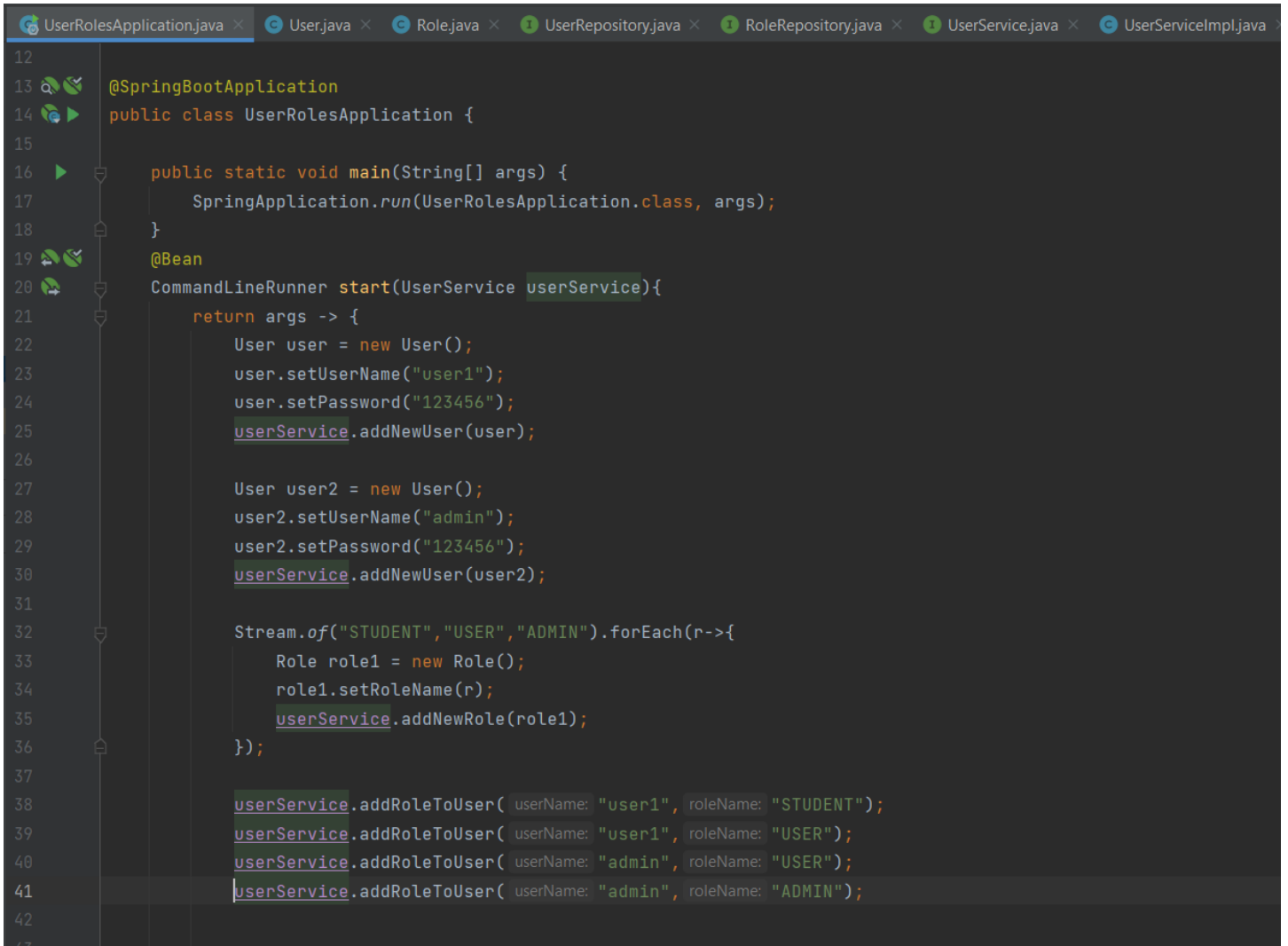
```
@Override
public Role addNewRole(Role role) {
    return roleRepository.save(role);
}
```

```
@Override
public User findUserByUserName(String userName) {
    return userRepository.findByUserName(userName);
}
```

```
@Override
public Role findRoleByRoleName(String roleName) {
    return roleRepository.findByRoleName(roleName);
}
```

```
@Override
public void addRoleToUser(String userName, String roleName) {
    User user = findUserByUserName(userName);
    Role role = findRoleByRoleName(roleName);
    if(user.getRoles()!=null){
        user.getRoles().add(role);
        role.getUsers().add(user);
    }
    //userRepository.save(user);
}
```

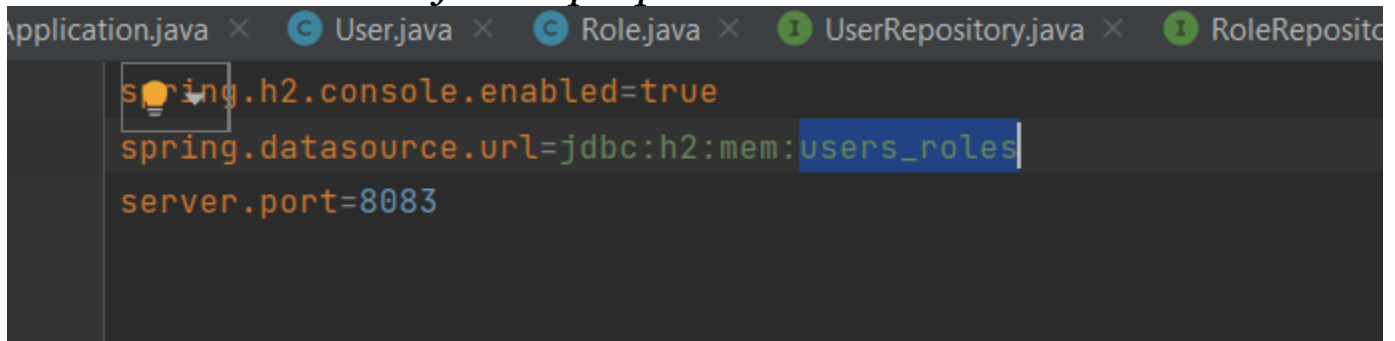
Passant testons nos fonctions, pour cela j'ai créer des utilisateurs et des roles



```
12
13 @SpringBootApplication
14 public class UserRolesApplication {
15
16     public static void main(String[] args) {
17         SpringApplication.run(UserRolesApplication.class, args);
18     }
19
20     @Bean
21     CommandLineRunner start(UserService userService){
22         return args -> {
23             User user = new User();
24             user.setUserName("user1");
25             user.setPassword("123456");
26             userService.addNewUser(user);
27
28             User user2 = new User();
29             user2.setUserName("admin");
30             user2.setPassword("123456");
31             userService.addNewUser(user2);
32
33             Stream.of("STUDENT", "USER", "ADMIN").forEach(r->{
34                 Role role1 = new Role();
35                 role1.setRoleName(r);
36                 userService.addNewRole(role1);
37             });
38
39             userService.addRoleToUser( userName: "user1", roleName: "STUDENT");
40             userService.addRoleToUser( userName: "user1", roleName: "USER");
41             userService.addRoleToUser( userName: "admin", roleName: "USER");
42             userService.addRoleToUser( userName: "admin", roleName: "ADMIN");
43 }
```

Figure 7 : Class Application

Pour voir si cela voici le fichier properties



```
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:users_roles
server.port=8083
```

Figure 8 : application.properties

et voici les resultat dans la base de données

The screenshot shows the H2 database interface with the 'ROLE' table selected in the left sidebar. The SQL statement 'SELECT * FROM ROLE;' is entered in the top bar. The results pane displays the following table:

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

(3 rows, 2 ms)

Figure 9 : table role

The screenshot shows the H2 database interface with the 'USER' table selected in the left sidebar. The SQL statement 'SELECT * FROM USER;' is entered in the top bar. The results pane displays the following table:

USER_ID	PASSWORD	USER_NAME
e457a67b-68fb-4d72-a049-50d1744319ff	123456	user1
2dedd261-8bd1-4af3-a44f-408845cd8d4b	123456	admin

(2 rows, 1 ms)

Figure 10 : table user

The screenshot shows the H2 database interface with the 'ROLE_USERS' table selected in the left sidebar. The SQL statement 'SELECT * FROM ROLE_USERS;' is entered in the top bar. The results pane displays the following table:

ROLES_ID	USERS_USER_ID
1	e457a67b-68fb-4d72-a049-50d1744319ff
2	e457a67b-68fb-4d72-a049-50d1744319ff
2	2dedd261-8bd1-4af3-a44f-408845cd8d4b
3	2dedd261-8bd1-4af3-a44f-408845cd8d4b

(4 rows, 2 ms)

Figure 11 : table association users_roles

*Après j'ai ajouté une methode d'authentification pour tester
Voici l'implementation de cette methode*

```
@Override
public User authenticate(String userName, String password) {
    User user = findUserByUserName(userName);
    if(user==null){
        throw new RuntimeException("Bad Credentials");
    }
    if(user.getPassword().equals(password)){
        return user;
    }
    throw new RuntimeException("Bad Credentials");
}
```

Figure 12 : methode d'authentification

Dans l'application j'ai affiché les informations d'user qui a par exemple bien saisie le nom et le mdp

```
try{
    User userr = userService.authenticate( userName: "user1", password: "123456");
    System.out.println(userr.getUserId());
    System.out.println(userr.getUserName());
    userr.getRoles().forEach(r->{
        System.out.println("Roles : "+r.toString());
    });
}catch (Exception e){
    e.printStackTrace();
}
```

Et voila le resultat de l'execution

```
2022-03-31 01:01:33.961 INFO 24380 --- [main] m.a.user_r
a44a36ec-1209-4288-a011-9738fe31b409
user1
Roles : Role(id=1, desc=null, roleName=STUDENT)
Roles : Role(id=2, desc=null, roleName=USER)
```

Après je voulais utiliser une base de données mySQL a la place de h2 database pour cela il faut changer dans le fichier application.properties

```
#spring.h2.console.enabled=true
#spring.datasource.url=jdbc:h2:mem:users_roles
server.port=8083
spring.datasource.url=jdbc:mysql://localhost:3306/USERS_DB?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect
spring.jpa.show-sql=true
```

Et voici j'ai pu changer la base donnée

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> role	★ Parcourir Structure Rechercher Insérer Vider Supprimer	3	InnoDB	latin1_swedish_ci	32,0 kio	-
<input type="checkbox"/> role_users	★ Parcourir Structure Rechercher Insérer Vider Supprimer	4	InnoDB	latin1_swedish_ci	48,0 kio	-
<input type="checkbox"/> user	★ Parcourir Structure Rechercher Insérer Vider Supprimer	2	InnoDB	latin1_swedish_ci	32,0 kio	-

☐ Tout afficher | Nombre de lignes : 25 ▼ Filtrer les lignes: Ch

+ Options

				id	description	role_name
<input type="checkbox"/>	✎ Éditer	✚ Copier	⊖ Supprimer	1	NULL	STUDENT
<input type="checkbox"/>	✎ Éditer	✚ Copier	⊖ Supprimer	2	NULL	USER
<input type="checkbox"/>	✎ Éditer	✚ Copier	⊖ Supprimer	3	NULL	ADMIN

+ Options

roles_id	users_user_id
1	14facbcd-6eb4-4146-9965-634a3d667b0e
2	14facbcd-6eb4-4146-9965-634a3d667b0e
2	3f438511-436c-49bb-8b20-945400f645a1
3	3f438511-436c-49bb-8b20-945400f645a1

		▼ user_id	password	user_name
<input type="checkbox"/>	✎ Éditer	✂ Copier	Supprimer 14facbcd Geb4-4146-9965-634a3d667b0e 123456	user1
<input type="checkbox"/>	✎ Éditer	✂ Copier	8b20-945400f645a1 123456	admin

Cliquer sur la flèche
pour afficher/masquer la colonne.