

DEPARTEMENT MATHÉMATIQUES ET INFORMATIQUE

Compte rendu de l'activité pratique 4

**Gérer les associations et l'héritage entre les entités Partie II
« Association ManyToMany »**

**Filière :
« Génie du Logiciel et des Systèmes Informatiques Distribués »
GLSID2**

Module : Architecture Distribuée et Middlewares

Élaboré par :

ELMAJNI Khaoula

Encadré par :

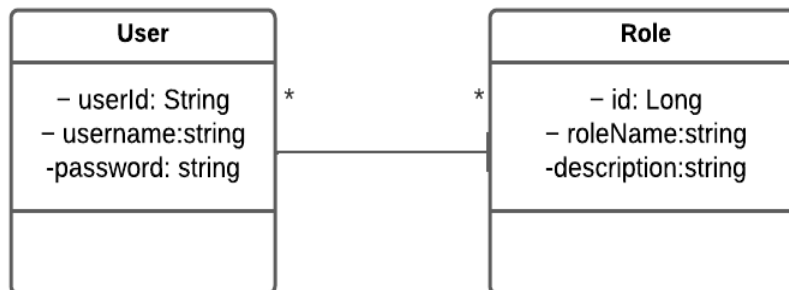
M. YOUSSEFI Mohammed

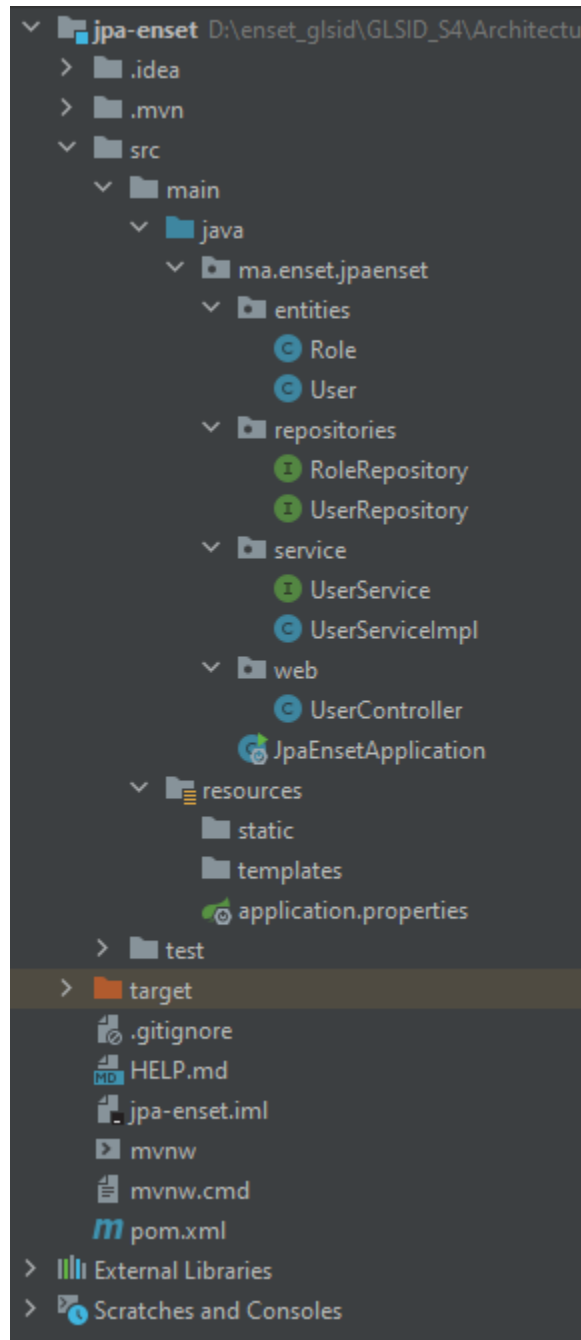
Année Universitaire : 2021-2022

Introduction

Dans ce travail on va continuer le Mapping Objet Relationnel des associations et bien préciser l'association '*ManyToMany*' entre 2 entités.

Le modèle conceptuel de données de l'application





Exigences fonctionnelles :

L'application permet de :

- Gérer des utilisateurs :
 1. Ajouter un utilisateur
 2. Consulter tous les utilisateurs

3. Ajouter un role à un utilisateur
4. Chercher un utilisateur part son nom
- Gérer les roles :
 1. Ajouter un role
 2. Consulter les roles
 3. Ajouter un utilisateur

Les entités JPA :

Entité User :

```
@Entity
@Table(name = "USERS")
@Data @NoArgsConstructor @AllArgsConstructor
public class User {
    @Id
    private String userId;
    @Column(name = "USER_NAME", unique = true, length =
20) //index unique
    private String username;
    @JsonProperty(access =
JsonProperty.Access.WRITE_ONLY)
    private String password;

    @ManyToMany (mappedBy = "users", fetch =
FetchType.EAGER) //relation pls à pls
    private List<Role> roles = new ArrayList<>();
}
```

Entité Role :

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Role {
    @Id @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;
    @Column(unique = true, length = 20)
    private String roleName;
    @ManyToMany(fetch = FetchType.EAGER)
```

```
//relation pls à pls
//@JoinTable(name = "USERS_ROLES",)
    @ToString.Exclude //pour exclure la liste des
utilisateurs durant la récupération
    @JsonProperty(access =
JsonProperty.Access.WRITE_ONLY)
//pour ne pas afficher à chaque utilisateurs ses roles et
vice versa
    private List<User> users = new ArrayList<>();
    @Column(name = "DESCRIPTION")
    private String desc;
}
```

Les interfaces JPA “couche DAO”

```
Repository //component pour la couche DAO
public interface RoleRepository extends
JpaRepository<Role, Long> {
    Role findByRoleName(String roleName);
}
```

```
Repository
public interface UserRepository extends
JpaRepository<User, String> {
    User findByUsername(String username);
}
```

la couche “METIER”:

```
public interface UserService {
    User addNewUser(User user);
    Role addNewRole(Role role);
    User findUserByUsername(String username);
    Role findRoleByRoleName(String roleName);
    void addRoleToUser(String username, String rolename);
    User authenticate(String username, String password);
}
```

L'implémentation:

```
@Service
@Transactional
@AllArgsConstructor
public class UserServiceImpl implements UserService {
    private UserRepository userRepository;
    private RoleRepository roleRepository;
    @Override
    public User addUser(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.findByName(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);
        }
        //role.getUsers().add(user); ==> pas besoin car
il est transactionnel
    }
    @Override
    public User authenticate(String username, String
```

```
password) {  
    User user =  
userRepository.findByUsername(username);  
    if (user==null){  
        throw new RuntimeException("BAD  
credentials");  
    }  
    if (user.getPassword().equals(password)) {  
        return user;  
    }  
    throw new RuntimeException("BAD credentials");  
}  
}
```

La couche “présentation” :

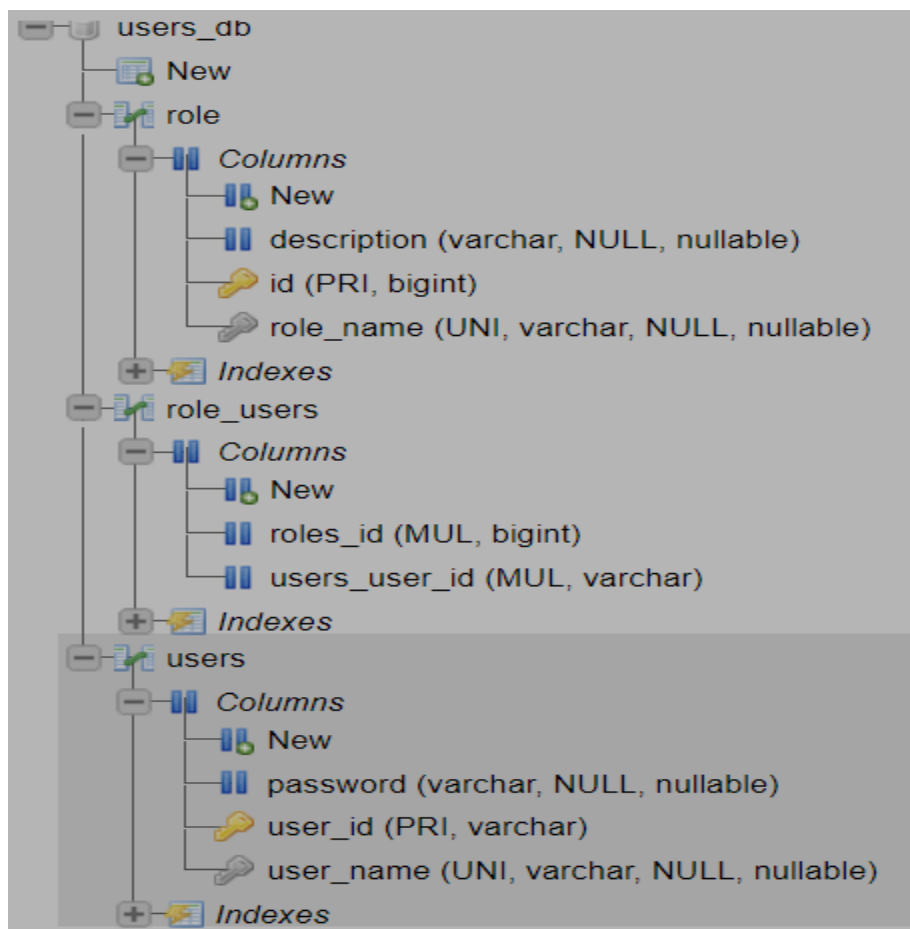
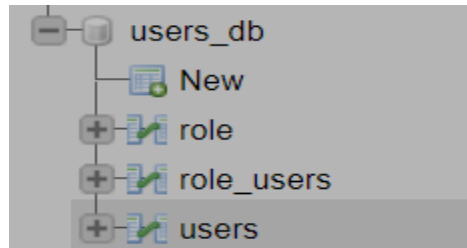
```
@RestController  
public class UserController {  
    @Autowired  
    private UserService userService;  
    @GetMapping("/users/{username}")  
    public User user(@PathVariable String username) {  
        User user =  
userService.findUserByUsername(username);  
        return user;  
    }  
}
```

Configuration de “application.properties”:

```
spring.datasource.url=jdbc:mysql://localhost:3306/USERS_D  
B?createDatabaseIfNotExist= true  
spring.datasource.username=root  
spring.datasource.password=  
spring.jpa.hibernate.ddl-auto=create  
spring.jpa.properties.hibernate.dialect=org.hibernate.dia  
lect.MariaDBDialect  
spring.jpa.show-sql=true  
server.port=8080
```

Résultat

La base de données mysql générée(les tables avec l'association ManyToMany) :



Conclusion

Ce travail nous a permis de bien connaître comment le Mapping Objet Relationnel se fait au niveau du Java grâce à l'API JPA, et on a initié avec Hibernate, et bien évidemment la manière de configuration d'un projet Spring pour faciliter la tâche au développeur.

Ecole Normale Supérieure de
l'Enseignement Technique Mohammedia
Université Hassan II de Casablanca



المدرسة العليا لأساتذة التعليم التقني
المحمدية
جامعة الحسن الثاني بالدار البيضاء