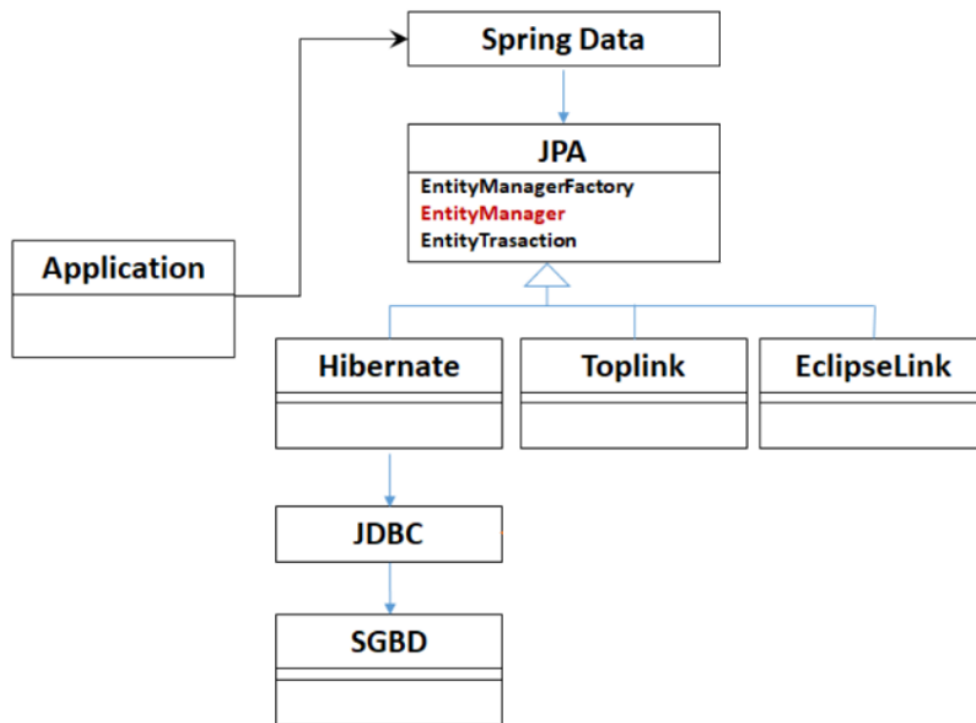


Activité Pratique N°2 - JPA, Hibernate et Spring Data

Hajar TAJAYOUTI

GLSID2



le Mapping Objet Relationnel (ORM), est un type de programme informatique qui se place en interface entre un programme applicatif et une base de données relationnelle pour simuler une base de données orientée objet à travers le JDBC.

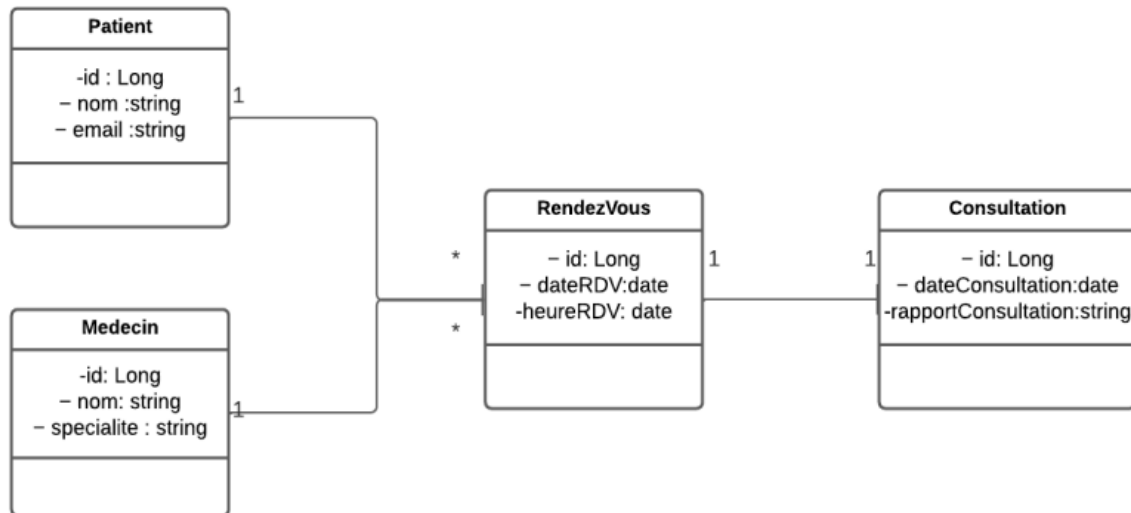
JDBC : Java DataBase Connectivity permet à une application Java de communiquer avec n'importe quel SGBDR.

JPA : Java Persistence API est une spécification créée par SUN pour standardiser le ORM, et on peut considérer que ce JPA est une interface globale et comme implémentation nous avons le choix de travailler avec différents outils : Hibernate / TopLink / EclipseLink.

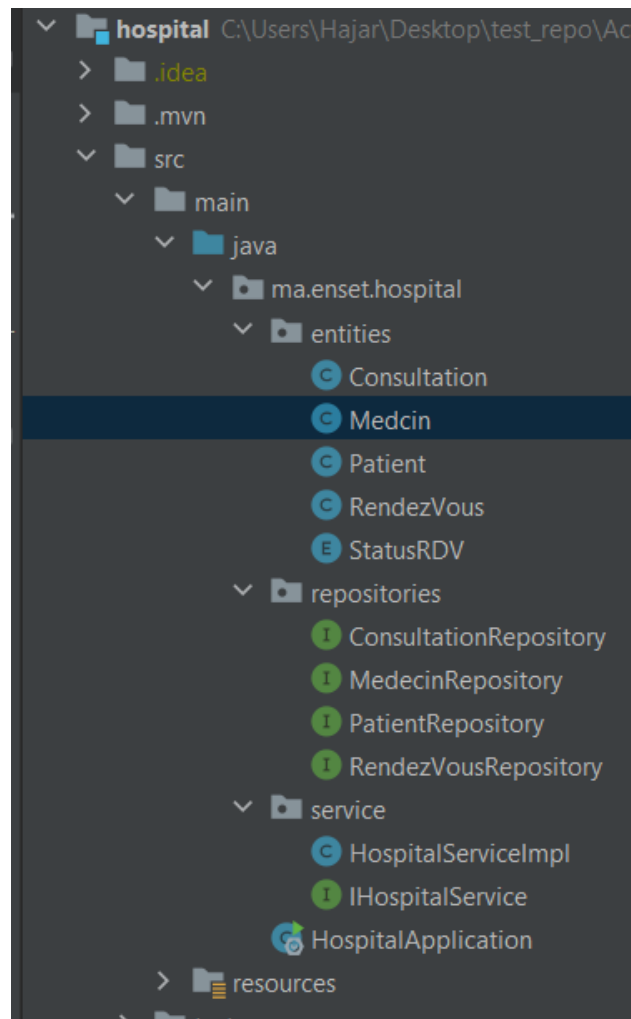
Spring Data : un module de Spring qui facilite le Mapping Objet Relationnel basé sur JPA.

Hospital

Le modèle conceptuel de données de l'application



Structure technique du projet :



L'application permet de :

- Gérer les patients :

1. Ajouter un patient
2. Afficher la liste des patients
3. chercher un patient par un mot clé.

- Gérer les médecins :

1. Ajouter un médecin
2. Afficher la liste des médecins

- Gérer les rendez-vous :

1. ajouter un rendez-vous pour un patient

2. Consulter les rendez-vous

- Gérer les consultations :

1. Ajouter une consultation qui concerne un rendez-vous

Les entités JPA :

Entité Consultation :

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Consultation {
    @Id @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;
    private Date dateConsultation;
    private String rapportConsultation;
    @OneToOne
    @JsonProperty(access =
JsonProperty.Access.WRITE_ONLY)
    private RendezVous rendezVous;
}
```

Entité Médecin :

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Medecin {
    @Id @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private String email;
```

```

        private String specialite;
        @OneToMany(mappedBy = "medecin", fetch =
            FetchType.LAZY)
        @JsonProperty(access =
            JsonProperty.Access.WRITE_ONLY)
        private Collection<RendezVous> rendezVous;
    }

```

Entité Patient:

```

@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Patient {
    @Id @GeneratedValue(strategy =
        GenerationType.IDENTITY)
    private Long id;
    private String nom;
    @Temporal(TemporalType.DATE)
    private Date dateNaissance;
    private boolean malade;
    @OneToMany(mappedBy = "patient", fetch =
        FetchType.LAZY)
    private Collection<RendezVous> rendezVous;
}

```

Entité Rendez-vous :

```

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class RendezVous {
    @Id // @GeneratedValue(strategy =
        GenerationType.IDENTITY)
    private String id;
    private Date dateCreation;
    //pour le stockage de type string
    @Enumerated(EnumType.STRING)
    private StatusRDV status;
    @ManyToOne
    @JsonProperty(access =
        JsonProperty.Access.WRITE_ONLY)

```

```

        private Patient patient;
        @ManyToOne
        private Medecin medecin;
        @OneToOne(mappedBy = "rendezVous", fetch =
FetchType.LAZY)
        private Consultation consultation;
    }
}

```

Les interfaces JPA

```

public interface ConsultationRepository extends
JpaRepository<Consultation, Long> {
}

```

```

public interface MedecinRepository extends
JpaRepository<Medecin, Long> {
    Medecin findByNom(String nom);
}

```

```

public interface PatientRepository extends
JpaRepository<Patient, Long> {
    Patient findByNom(String nom);
}

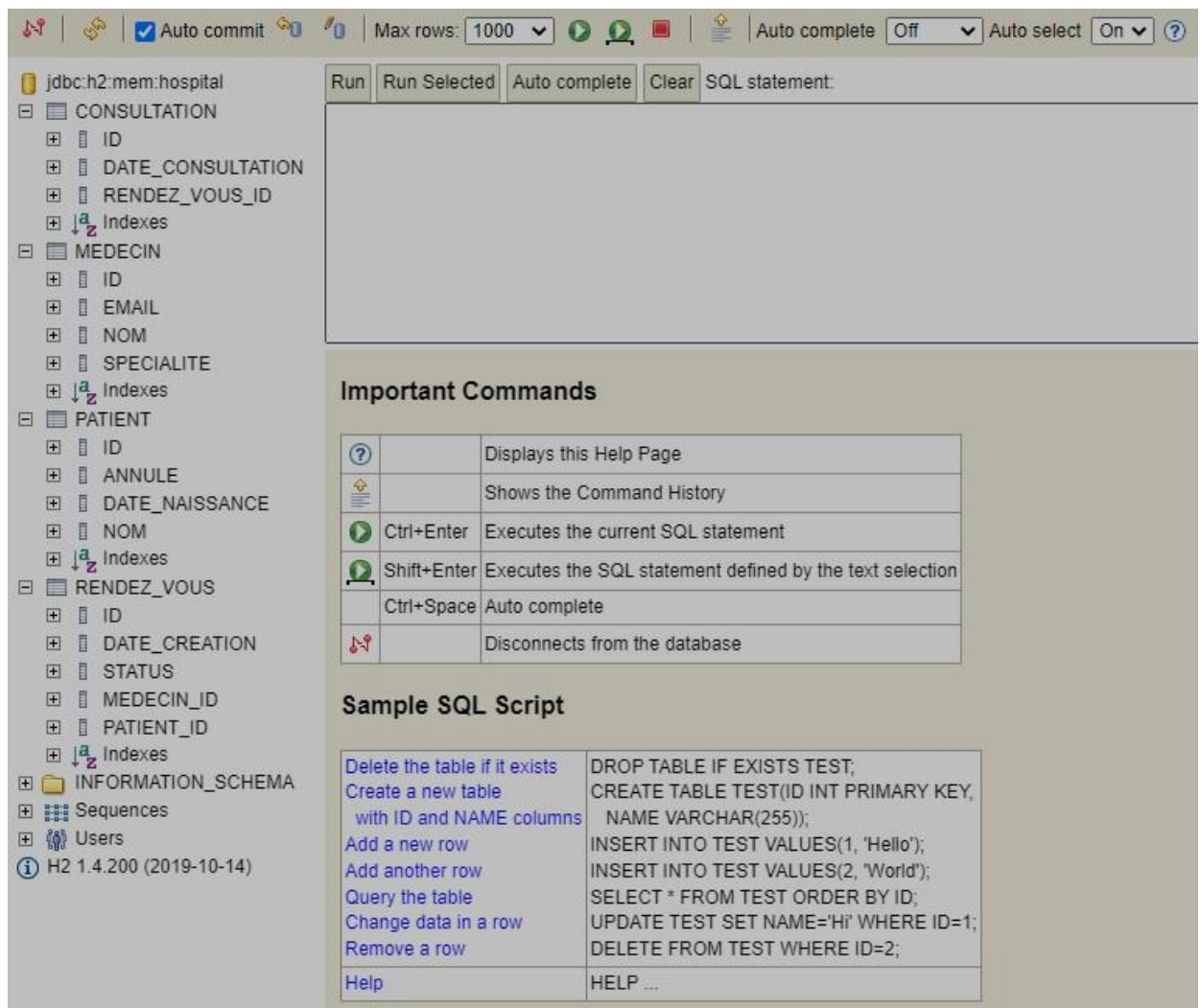
```

```

public interface RendezVousRepository extends
JpaRepository<RendezVous, String> {
}

```

Résultat



L'affichage des patients de la base de données :Le controleur des patients (/patients)

```
@RestController
public class PatientController {
    @Autowired
    private PatientRepository patientRepository;
    @GetMapping("/patients")
    public List<Patient> patientList(){
        return patientRepository.findAll();
    }
}
```

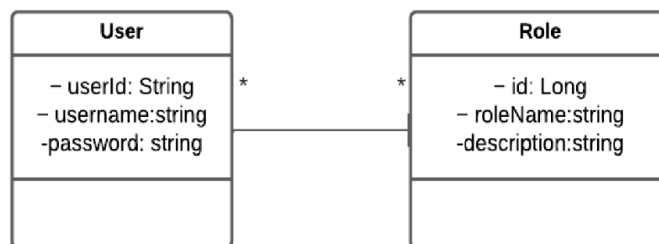


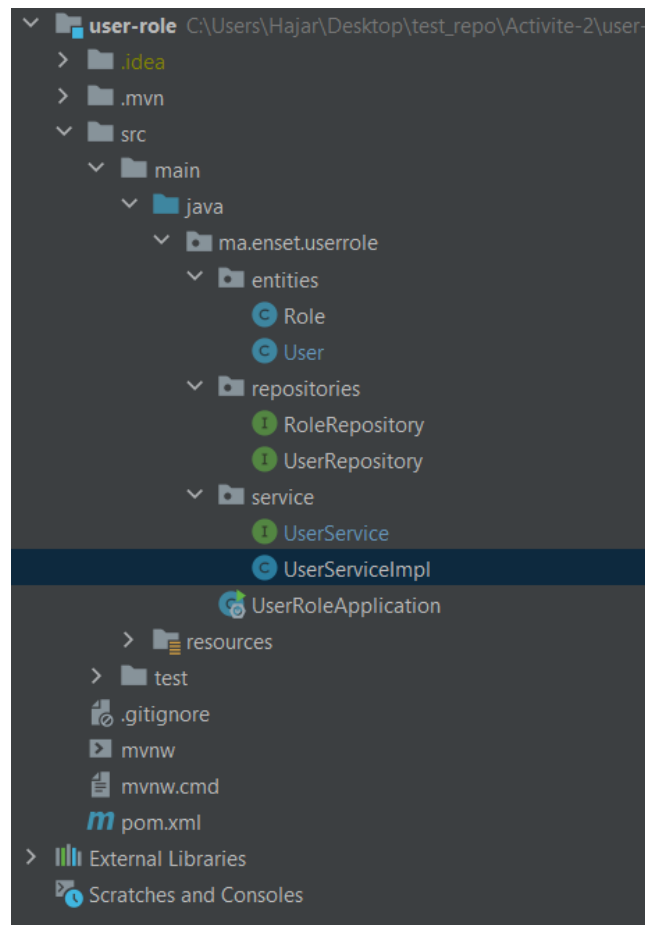
```

    "id": 1,
    "nom": "moahemmed",
    "dateNaissance": "2022-03-12",
    "malade": false,
    "rendezVous": [
      {
        "id": "acdbdc5e-812e-4905-933b-022c5135ea0d",
        "dateCreation": "2022-03-12T10:29:22.517+00:00",
        "status": "CANCELED",
        "medecin": {
          "id": 1,
          "nom": "khadija",
          "email": "khadija@gmail.com",
          "specialite": "chirurgie"
        },
        "consultation": {
          "id": 1,

```

User-role





L'application permet de :

- Gérer des utilisateurs :

1. Ajouter un utilisateur
2. Consulter tous les utilisateurs

3. Ajouter un rôle à un utilisateur

4. Chercher un utilisateur part son nom

- Gérer les rôles :

1. Ajouter un rôle
2. Consulter les rôles
3. Ajouter un utilisateur

Les entités JPA :

Entité User :

```
18 usages  hajar tajayouti *  
@Entity  
@Table(name = "USERS")  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
  
public class User {  
    @Id  
    private String userId;  
    @Column(unique = true ,length = 20)  
    private String userName;  
    private String pwd;  
    @ManyToMany(mappedBy = "users",fetch = FetchType.EAGER)  
    /*sois dans role ou bien user  
    dans la classe role il y a un attribus qui s appelle users*/  
    private List<Role> roles=new ArrayList<>();  
}
```

Entité Role :

16 usages hajar tajayouti

```
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @Column(unique = true ,length = 20)
    private String roleName;
    @ManyToMany(fetch = FetchType.EAGER)
    private List<User> users=new ArrayList<>();
}
```

Les interfaces JPA “couche DAO”

```
Repository //composant 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 //compoent pour la couche metier
@Transactional
@AllArgsConstructor //injection via le const
public class UserServiceImpl implements UserService {
    2 usages
    private UserRepository userRepository;
    2 usages
    private RoleRepository roleRepository;
    2 usages hajar tajayouti
    @Override
    public User addNewUser(User user) {
        user.setUserId(UUID.randomUUID().toString()); //GENERE UN USER ID SOUS
        return userRepository.save(user);
    }

    1 usage hajar tajayouti
    @Override
    public Role addNewRole(Role role) {
        return roleRepository.save(role);
    }

    1 usage hajar tajayouti
    @Override
    public User findUserByUsername(String username) {
        return userRepository.findByUserName(username);
    }
}
```

```

@Override
public Role findRoleByRoleName(String rolename) {

    return roleRepository.findByRoleName(rolename);
}

4 usages  hajar tajayouti
@Override
public void addRoleToUser(String username, String rolename) {
    User user=findUserByUserName(username);
    Role role=findRoleByRoleName(rolename);

    user.getRoles().add(role);
    role.getUsers().add(user);
    //userRepository.save(user);

    //aller vers la collection des roles de user et ajouter le role
}
}

```

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;
    }
}

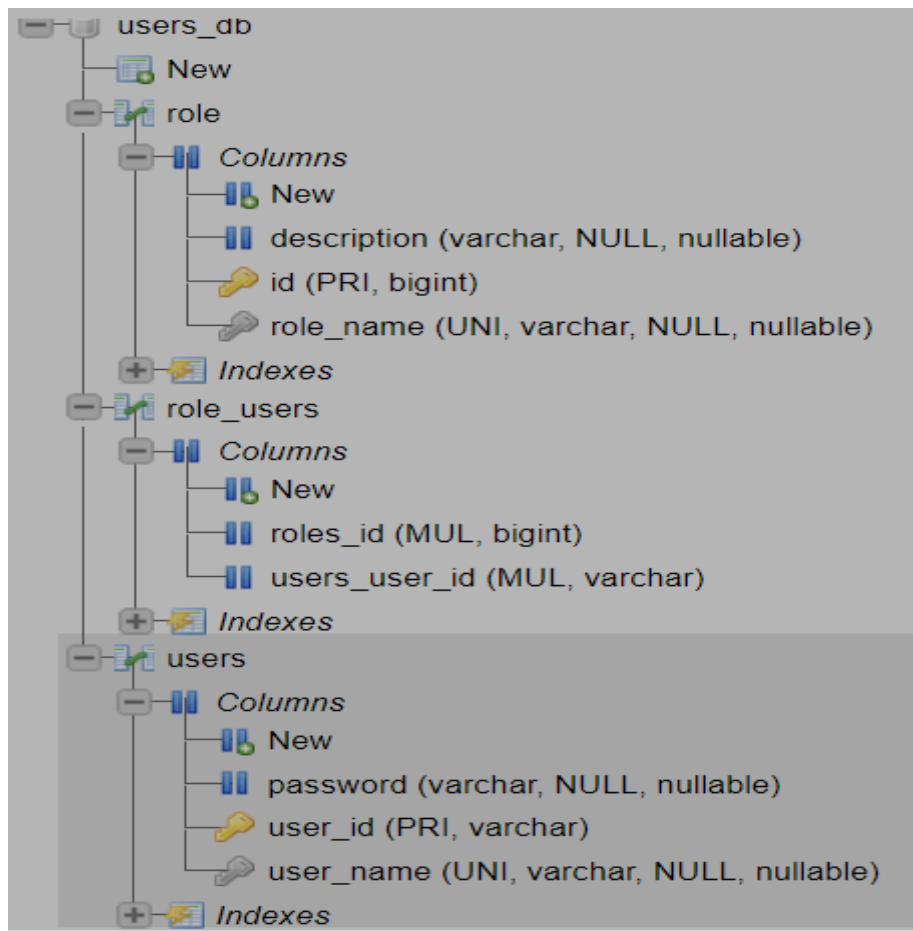
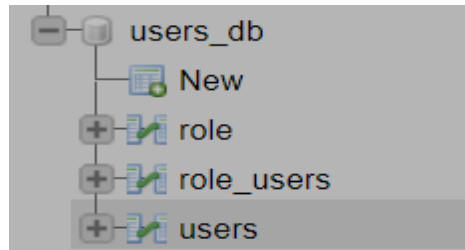
```

Configuration de “application.properties”:

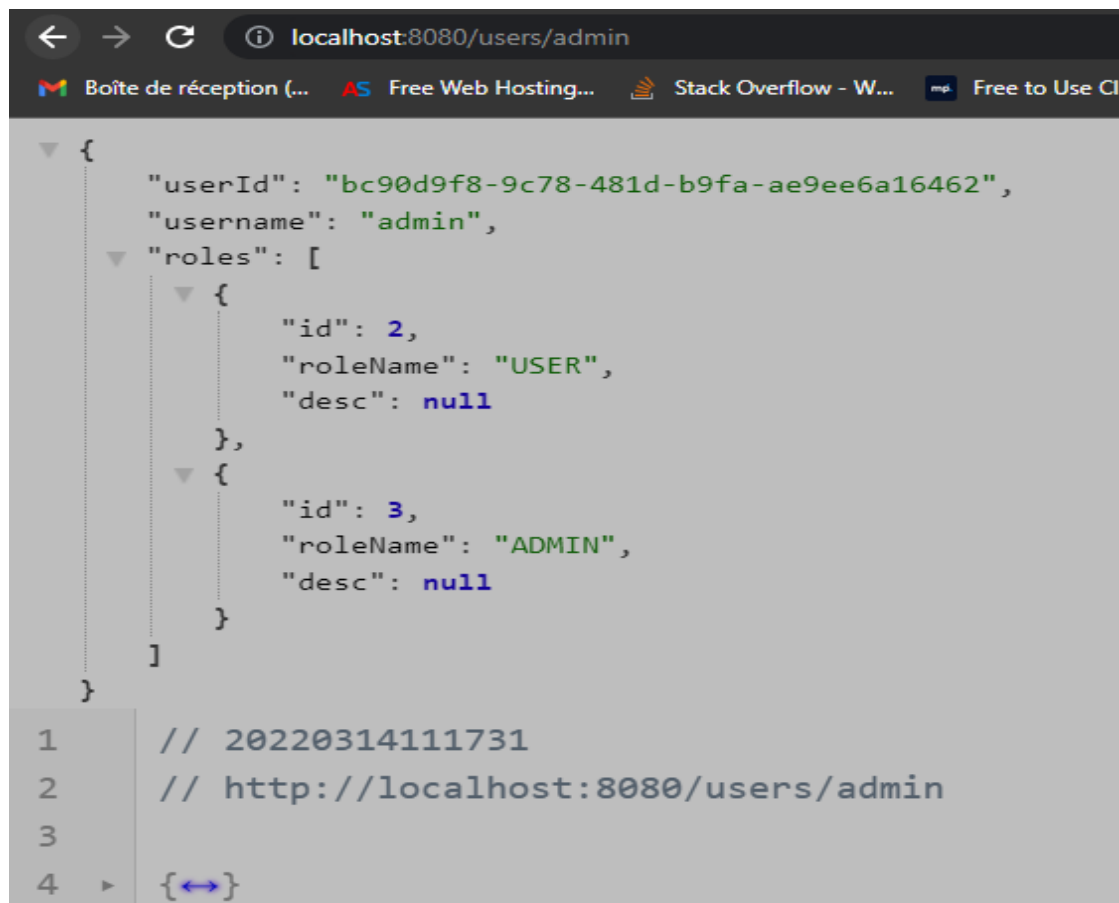
```
spring.datasource.url=jdbc:mysql://localhost:3306/USERS_D
B?createDatabaseIfNotExist= true
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dia
lect.MariaDBDialect
spring.jpa.show-sql=true
server.port=8080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


Résultat

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



La route (localhost:8080/users/admin): retourne les données de l'utilisateur 'admin':



The screenshot shows a web browser window with the address bar displaying `localhost:8080/users/admin`. Below the address bar, there are several tabs: "Boîte de réception (...)", "AS Free Web Hosting...", "Stack Overflow - W...", and "Free to Use Cl". The main content area displays a JSON response from a REST client. The JSON object has the following structure:

```
{
  "userId": "bc90d9f8-9c78-481d-b9fa-ae9ee6a16462",
  "username": "admin",
  "roles": [
    {
      "id": 2,
      "roleName": "USER",
      "desc": null
    },
    {
      "id": 3,
      "roleName": "ADMIN",
      "desc": null
    }
  ]
}
```

Below the JSON response, there is a list of four items, each with a line number and a comment:

```
1 // 20220314111731
2 // http://localhost:8080/users/admin
3
4 ▶ {↔}
```

La route (`localhost:8080/users`): retourne les données de l'utilisateur 'admin':

```
localhost:8080/users

[
  {
    "userId": "22d48f22-32f2-4cbc-af66-4ae46780ddad",
    "username": "admin",
    "roles": [
      {
        "id": 2,
        "roleName": "USER",
        "desc": null
      },
      {
        "id": 3,
        "roleName": "ADMIN",
        "desc": null
      }
    ]
  },
  {
    "userId": "e872f497-9a7b-4707-be4f-aa4f49cddcf9",
    "username": "user1",
    "roles": [
      {
        "id": 1,
        "roleName": "STUDENT",
        "desc": null
      },
      {
        "id": 2,
        "roleName": "USER",
        "desc": null
      }
    ]
  }
]
```