TP1: Entités et Associations (Base de données MySQL)

Introduction

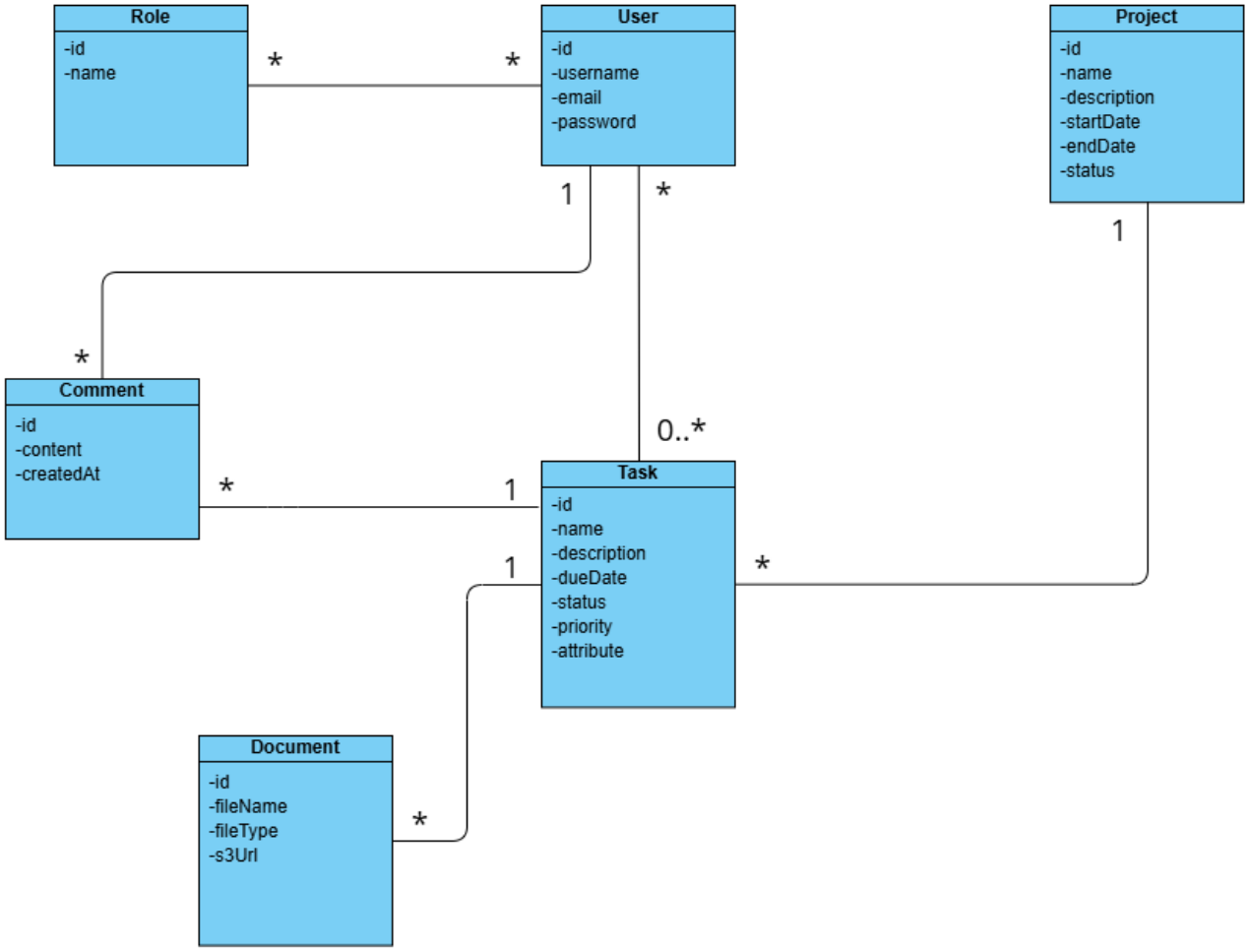
Ce TP a pour objectif de vous guider dans la création des entités JPA et la définition de leurs relations pour une application de gestion de projets. Vous apprendrez à configurer une base de données MySQL et à l'initialiser avec des données de test. L'implémentation des entités est une étape fondamentale pour structurer les données de votre application et établir les bases de la persistance.

Objectifs d'apprentissage :

- Comprendre la création d'entités JPA.
- Définir les relations entre les entités (One-to-Many, Many-to-One, Many-to-Many).
- Configurer une base de données MySQL pour le développement.
- Initialiser la base de données avec des données de test.

Diagramme de Classe Global

Voici le diagramme de classe UML qui représente les entités principales de notre application de gestion de projets et leurs relations. Ce diagramme vous servira de référence tout au long de l'implémentation des TPs.



Tâches à réaliser :

Tâche 1 : Initialisation du projet et définition des entités

Pour commencer, vous devez créer un projet Spring Boot et définir les entités nécessaires à notre application de gestion de projets. Chaque entité représente un concept clé du domaine et sera mappée à une table dans la base de données.

Entités à implémenter:

1. **User.java**
    - **Description :** Représente un utilisateur de l'application. Il contient les informations de base de l'utilisateur (identifiant, nom d'utilisateur, email) et gère les relations avec les rôles, les projets créés, les tâches assignées et les commentaires.
    - **Implémentation:** Créez la classe `User` avec les attributs `id`, `username`, `email`. Ajoutez les annotations JPA nécessaires (`@Entity`, `@Id`, `@GeneratedValue`, `@Table`). Implémentez les contraintes de validation

`@NotBlank`, `@Size` `@Email`

User.java

```java
package com.tekup.taskmanager.entities;

import jakarta.persistence.GeneratedValue;
import jakarta.persistence.*;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;
import lombok.Data;

import java.util.Set;

@Entity
@Data
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "Username cannot be empty")
    @Size(min = 3, max = 50, message = "Username must be between 3 and 50 characters")
    private String username;
    @NotBlank(message = "Email cannot be empty")
    @Email(message = "Email should be valid")
    private String email;
    @ManyToMany
    @JoinTable(name = "user_roles",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id"))
    private Set<Role> roles;
    @ManyToMany(mappedBy = "assignedUsers")
    private Set<Task> assignedTasks;
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL,
            orphanRemoval = true)
    private Set<Comment> comments;
}
```

2. **Role.java**

- **Description:** Représente un rôle d'utilisateur (ex: ADMIN, USER). Cette entité est séparée de l'entité `User` pour une meilleure gestion des rôles et une conformité aux principes de conception.
- **Implémentation:** Créez la classe `Role` avec les attributs `id` et `name`. Ajoutez les annotations JPA nécessaires.

  Implémentez les contraintes de validation :

  - Un rôle ne peut pas être vide
  - Un rôle doit être entre 3 et 20 caractères
- **Relations:**
  - `@ManyToMany` avec `User` (mappé par `roles` dans la classe `User`).

```
    @ManyToMany(mappedBy = "roles")
    private Set<User> users;
}
```

3. **Project.java**

- **Description:** Représente un projet. Il inclut des informations comme le nom, la description, les dates de début et de fin, le statut, et le créateur du projet.
- **Implémentation:** Créez la classe `Project` avec les attributs `id`, `name`, `description`, `startDate`, `endDate`, `status`. Ajoutez les annotations JPA et les contraintes de validation (`@FutureOrPresent`, `@NotNull`) :
  - Le nom du projet ne peut pas être nul `(@NotBlank)` et doit contenir entre 3 et 100 caractères.
  - La description ne doit pas dépasser 500 caractères `(@Size)`.
  - La date de début du projet est obligatoire (`@NotBlank`) et doit être aujourd'hui ou dans le futur `(@FutureOrPresent)`.
  - La date de fin du projet est également obligatoire (`@NotBlank`).
- **Relations:**

  `@OneToMany` avec `Task` (un projet peut contenir plusieurs tâches). Utilisez `CascadeType.ALL` et `orphanRemoval = true`.

```
@OneToMany(mappedBy = "project", cascade = CascadeType.ALL,
        orphanRemoval = true)
private Set<Task> tasks;
```

4. **Task.java**

- **Description:** Représente une tâche au sein d'un projet. Elle est liée à un projet, peut être assignée à plusieurs utilisateurs, et peut avoir des documents et des commentaires associés.
- **Implémentation:** Créez la classe `Task` avec les attributs `id`, `name`, `description`, `dueDate`, `status`, `priority`. Ajoutez les annotations JPA et les contraintes de validation :
    - Le nom de la tâche ne peut pas être vide (@NotBlank) et doit contenir entre 3 et 100 caractères (@Size(min = 3, max = 100)).
    - La description ne doit pas dépasser 500 caractères (@Size(max = 500)).
    - La date d'échéance est obligatoire (@NotNull) et doit être aujourd'hui ou dans le futur (@FutureOrPresent).
    - Le statut de la tâche ne peut pas être vide (@NotBlank).
    - La priorité de la tâche ne peut pas être vide (@NotBlank).

- **Relations :**
    - `@ManyToOne` avec `Project` (une tache appartient à un seul projet)
    - `@ManyToOne` avec `User` (Plusieurs utilisateurs peuvent etre assignés à une tache). Utilisez la table de jointure `task_assigned_users`
    - `@OneToMany` avec `Document` (une tache peut avoir plusieurs documents). Utilisez `CascadeType.All` et `orphanRemoval =true`
    - `@OneToMany avec Comment (Une tache peut avoir plusieurs commentaires.` Utilisez `CascadeType.All` et `orphanRemoval =true`

```java
@ManyToOne
@JoinColumn(name = "project_id")
private Project project;
@ManyToMany
@JoinTable(name = "task_assigned_users",
joinColumns = @JoinColumn(name = "task_id"),
inverseJoinColumns = @JoinColumn(name = "user_id"))
private Set<User> assignedUsers;
@OneToMany(mappedBy = "task", cascade = CascadeType.ALL,
        orphanRemoval = true)
private Set<Document> documents;
@OneToMany(mappedBy = "task", cascade = CascadeType.ALL,
        orphanRemoval = true)
private Set<Comment> comments;
```

5. **Comment.java**

- **Description :** Représente un commentaire laissé sur une tâche par un utilisateur. Il enregistre le contenu du commentaire et la date de création.
- **Implémentation :** Créez la classe `Comment` avec les attributs `id`, `content`, `createdAt`. Ajoutez les annotations JPA et les contraintes de validation :
    - Le contenu du commentaire ne peut pas être vide `(@NotBlank)` et ne doit pas dépasser 1000 caractères `(@Size(max = 1000))`.

  Utilisez `@PrePersist` pour définir automatiquement la date de création.

```
@PrePersist
protected void onCreate() {
    createdAt = LocalDateTime.now();
}
/*
L'annotation @PrePersist en Java (avec JPA/Hibernate) est utilisée pour exécuter une méthode automatiquement juste avant que l'entité soit insérée (persistée) dans la base de données pour la première fois.
*/
```

- **Relations:**
    - `@ManyToOne` avec `Task` (Un commentaire est lié à une seule tâche).
    - `@ManyToOne` avec `User` (Un commentaire est laissé par un seul utilisateur).

```
@ManyToOne
@JoinColumn(name = "task_id")
private Task task;
@ManyToOne
@JoinColumn(name = "user_id")
private User user;
```

6. **Document.java**

- **Description :** Représente un document attaché à une tâche. Il stocke le nom du fichier, son type, et l'URL S3 où il est stocké.
- **Implémentation :** Créez la classe `Document` avec les attributs `id`, `filename`, `fileType`, `s3Url`. Ajoutez les annotations JPA et les contraintes de validation :
    - Le nom du fichier ne peut pas être vide (@NotBlank) et ne doit pas dépasser 255 caractères (@Size(max = 255)).
    - Le type de fichier est obligatoire (@NotBlank).
    - L'URL S3 ne peut pas être vide (@NotBlank).

- **Relations :**
    - `@ManyToOne` avec `Task` (un document est lié à une seule tâche).

```
@ManyToOne
@JoinColumn(name = "task_id")
private Task task;
```

Tâche 2: Configuration MySQL

Configurez le fichier `src/main/resources/application.properties` pour établir la connexion à votre base de données MySQL. Assurez-vous que votre serveur MySQL est en cours d'exécution et que la base de données `projectdb` existe ou sera créée automatiquement grâce à la propriété `createDatabaseIfNotExist=true`

```
spring.application.name=TaskManager
spring.datasource.url=jdbc:mysql://localhost:3306/projectdb?
createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Tâche 3: Création des dépôts (Repositories)

Pour chaque entité définie dans le TP1 (User, Role , Project , Task , Comment , Document), vous devez créer une interface JpaRepository . Ces interfaces vous permettront d'interagir avec la base de données sans écrire de code d'implémentation pour les opérations CRUD de base. Vous pouvez également ajouter des méthodes de requête personnalisées si nécessaire.

1. **UserRepository.java**
    - **Description :** Interface de dépôt pour l'entité User. Elle étend JpaRepository et inclut une méthode personnalisée pour rechercher un utilisateur par son nom d'utilisateur : **`findByUsername(String username)`**
2. **RoleRepository.java**
    - **Description :** Interface de dépôt pour l'entité Role. Elle étend JpaRepository et inclut une méthode personnalisée pour rechercher un rôle par son nom **`findByName(String name)`**
3. **ProjectRepository.java**
    - **Description :** Interface de dépôt pour l'entité Project. Elle étend JpaRepository
4. **TaskRepository.java**
    - **Description :** Interface de dépôt pour l'entité Task. Elle étend JpaRepository

5. **CommentRepository.java**
    - **Description :** Interface de dépôt pour l'entité Comment. Elle étend JpaRepository.
6. **DocumentRepository.java**
    - **Description :** Interface de dépôt pour l'entité Document. Elle étend JpaRepository.