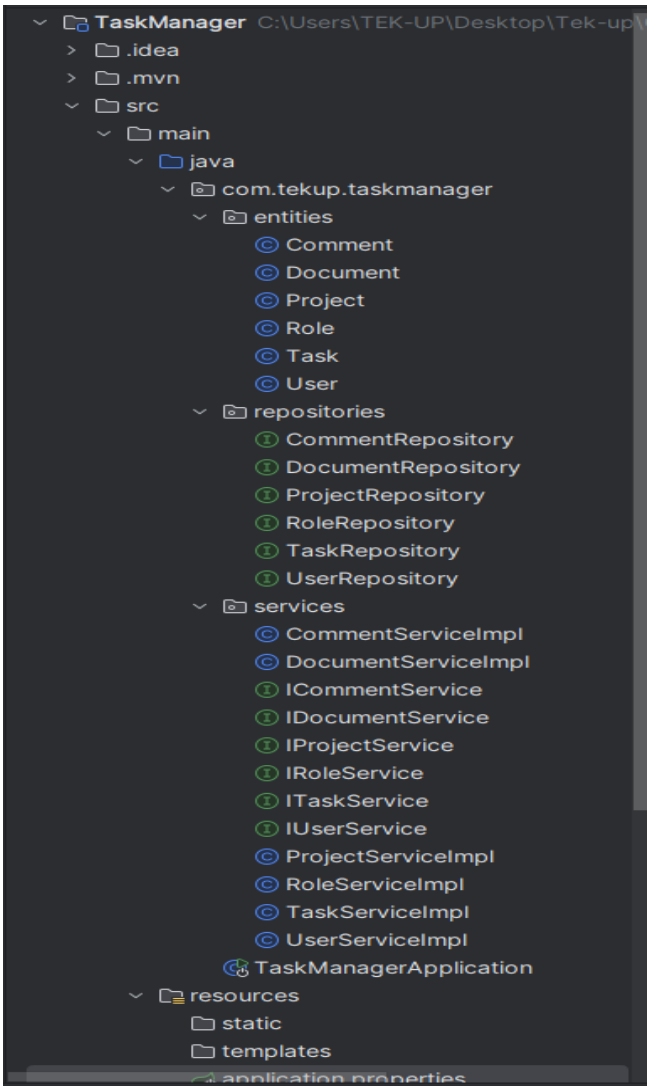
# TP2 : Les Services

## Introduction

Ce TP se concentre sur l'intégration de Spring Data JPA pour simplifier l'accès aux données et sur le développement de classes de service pour encapsuler la logique métier. Vous apprendrez à à développer des services qui utilisent ces dépôts, et à appliquer des concepts clés comme l'injection de dépendances ( @Autowired ) et la gestion des transactions ( @Transactional ).

## Architecture du Projet

Ci-dessous, l'arborescence du projet présente l'organisation des répertoires et des fichiers selon une structure claire et modulaire.



Pour chaque entité, vous allez créer une classe de service qui encapsulera la logique métier (voir l'exemple de UserService). Ces services utiliseront les interfaces Repository que vous avez créées pour interagir avec la base de données. Ils géreront également les transactions pour assurer la cohérence des données

**Service Utilisateur :**

1. **IUserService.java**
   - **Description :** Représente définit l'interface du service utilisateur, en spécifiant les opérations métiers liées à la gestion des utilisateurs. Elle fournit une abstraction claire pour les fonctionnalités telles que la création, la mise à jour, la suppression, la récupération ou la validation des utilisateurs, tout en facilitant la séparation des responsabilités et l'implémentation flexible du service.
   - **Implémentation :**

```java
import java.util.List;
import java.util.Optional;

public interface IUserService {  no usages

    User createUser(User user); // Créer un nouvel utilisateur  no usages
    User updateUser(Long id, User user);     // Mettre à jour un utilisateur existant  no usages
    void deleteUser(Long id); // Supprimer un utilisateur par son ID  no usages
    Optional<User> getUserById(Long id); // Récupérer un utilisateur par son ID  no usages
    List<User> getAllUsers(); // Récupérer tous les utilisateurs  no usages
    boolean existsByEmail(String email); // Vérifier l'existence d'un utilisateur par email  no usages

}
```

2. **UserServiceImpl.java :**
   - **Description :** Représente une implémentation complète de UserServiceImpl pour l'interface IUserService. Elle utilise Spring Boot, JPA (via UserRepository), et suit une logique standard :

```java
import com.tekup.taskmanager.entities.User;
import com.tekup.taskmanager.repositories.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class UserServiceImpl implements IUserService {

    private UserRepository userRepository;  9 usages

    @Autowired // Injection de dépendance via l'annotation le constructeur
    public UserServiceImpl(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
```

```java
@Override  no usages
public List<User> getAllUsers() {
    return userRepository.findAll();
}

@Override  no usages
public Optional<User> getUserById(Long id) {
    return userRepository.findById(id);
}

@Override  no usages
public User createUser(User user) {
    return userRepository.save(user);
}
```

```java
@Override  no usages
public User updateUser(Long id, User updatedUser) throws UserNotFoundException {
    Optional<User> existingUserOpt = userRepository.findById(id);
    if (existingUserOpt.isPresent()) {
        User existingUser = existingUserOpt.get();
        existingUser.setUsername(updatedUser.getUsername());
        existingUser.setEmail(updatedUser.getEmail());
        return userRepository.save(existingUser);
    } else {
        throw new UserNotFoundException("User not found with ID: " + id);
    }
}

@Override  no usages
public void deleteUser(Long id) throws UserNotFoundException {
    if (!userRepository.existsById(id)) {
        throw new UserNotFoundException("User not found with ID: " + id);
    }
    userRepository.deleteById(id);
}

@Override  no usages
public boolean existsByEmail(String email) {
    return userRepository.existsByEmail(email);
}
```

# Tâches à réaliser : Développement des services

Pour chacune des entités, vous devez développer les opérations CRUD ainsi que les méthodes avancées listées ci-dessous.

1. User

   - CRUD :

     - Créer un nouvel utilisateur avec validation des champs.
     - Modifier les informations d'un utilisateur.
     - Supprimer un utilisateur (ainsi que ses commentaires).
     - Afficher la liste des utilisateurs.

   - Méthodes avancées :

     - Rechercher un utilisateur par son nom ou email.
     - Lister les utilisateurs associés à un rôle donné.
     - Lister les tâches assignées à un utilisateur.

2. Role

   - CRUD :

     - Créer un nouveau rôle.
     - Modifier un rôle.
     - Supprimer un rôle.
     - Afficher tous les rôles.

   - Méthodes avancées :

     - Lister tous les utilisateurs ayant un rôle spécifique.
     - Affecter ou désaffecter un rôle à un utilisateur.

3. Project

   - CRUD :

     - Créer un nouveau projet avec ses informations de base.
     - Modifier un projet.
     - Supprimer un projet (et toutes ses tâches si elles existent plus tard).
     - Afficher la liste des projets.

   - Méthodes avancées :

     - Lister les projets actifs (par statut).
     - Lister les projets à venir (date de début dans le futur).
     - Rechercher un projet par nom ou mot-clé dans la description.
     - Lister les projets auxquels un utilisateur a participé via les tâches.

## 4. Comment

### ➢ CRUD :

- Ajouter un commentaire à une tâche.
- Modifier le contenu d'un commentaire.
- Supprimer un commentaire.
- Afficher les commentaires d'une tâche donnée.

### ➢ Méthodes avancées :

- Rechercher les commentaires contenant un mot-clé.
- Lister les commentaires d'un utilisateur donné.
- Trier les commentaires par date (du plus récent au plus ancien).