OPENCLASSROOMS

Corrigé Activité P1C3

Ce corrigé illustre la décomposition d'une exigence fonctionnelle en une spécification technique opérationnelle, mettant l'accent sur la clarté et l'anticipation des tests.

1. Description du Flux de Données et des Interactions

Fonctionnalité : Mise à jour du statut d'une tâche. **Composants impliqués :** UI (Frontend), Task_Service (Couche métier/Backend), Task_Repository (Couche de persistance/DB).

Composant	Rôle	Interaction
UI (Interface Utilisateur)	Envoie la requête HTTP de modification.	Envoie une requête **PUT /api/v1/tasks/{task_id}/status** avec l'ID de la tâche et le nouveau statut.
Task_Service	Reçoit la requête, vérifie les permissions, applique la logique métier, et coordonne la mise à jour.	Appelle Task_Repository.get_task(task_id) pour charger l'objet, puis appelle Task_Repository.save(task) après la modification du statut.
Task_Repository	Gère la communication directe avec la Base de Données (DB).	Effectue la requête SQL ou ORM pour la mise à jour effective.

Flux de Données Détaillé :

1. L'utilisateur (Employé technique) clique sur l'option "Terminé" dans l'UI.
2. L'UI envoie une requête API (PUT) contenant task_id, user_id, et new_status (terminé).
3. Le Task_Service reçoit la requête.
4. **Validation des Permissions :** Le Task_Service vérifie si le user_id a le droit de modifier le statut de cette task_id. Si non, il renvoie une erreur (HTTP 403 Interdit).
5. **Mise à Jour :** Si la permission est validée, le service met à jour l'objet Task avec le new_status.
6. Le Task_Service persiste la modification via Task_Repository.save(task).
7. Le Task_Service renvoie une réponse de succès (HTTP 200 OK) à l'UI.

2. Pseudocode Synthétique de la Fonction Principale

Nous adoptons la convention de nommage **snake_case** pour les fonctions et les variables, afin de garantir une lisibilité uniforme pour l'équipe (norme de code).

```
None
// Fichier : task_service.py (ou TaskService.java)

FONCTION mettre_a_jour_statut_tache(id_tache: Entier, nouvel_statut: Chaîne, id_utilisateur: Entier) : Tâche ou Erreur

    // 1. Vérification des données d'entrée

    SI nouvel_statut N'EST PAS VALIDE ALORS

        RETOURNER Erreur("Statut de tâche invalide")

    FIN SI

    // 2. Récupération de l'objet Tâche (via Task_Repository)

    tache_existante = task_repository.recuperer_par_id(id_tache)

    SI tache_existante EST VIDE ALORS

        RETOURNER Erreur("Tâche non trouvée", code_http=404)

    FIN SI

    // 3. Vérification des permissions (Point clé pour la sécurité)
```

```
    SI utilisateur_a_pas_permission(tache_existante, id_utilisateur) ALORS
        RETOURNER Erreur("Accès refusé. L'utilisateur n'est pas autorisé à modifier cette tâche.", code_http=403)
    FIN SI
    // 4. Application du changement
    tache_existante.statut = nouvel_statut
    tache_existante.date_modification = date_actuelle
    // 5. Persistance et Sauvegarde
    task_repository.sauvegarder(tache_existante)
    // 6. Déclenchement de la notification (si applicable, selon critères P1C2)
    notification_service.notifier_changement_statut(tache_existante)
    RETOURNER tache_existante
FIN FONCTION
```

Ce pseudocode démontre la décomposition de la *User Story* en étapes logiques, la séparation claire des responsabilités (service métier vs. persistance) et l'intégration des exigences non fonctionnelles (sécurité/permissions). En fournissant cette granularité de spécification, vous permettez aux développeurs de TechFlow Solutions d'implémenter la fonctionnalité avec précision, sans risque de malentendu sur le processus de résolution.