

2Rapport d'Iteration – Projet Application de Gestion de Tâches

Projet : Application de Gestion de Tâches

Itération : 3

Groupe : *Marchal Enzo, Nacime Laghezali, Cabot Matthieu, Vincent Julien*

Date : *18/12/2025*

1. Objectif de l'itération

Dans cette itération, notre but principal était de terminer l'intégration du DAO dans nos classes afin que l'application puisse enfin utiliser la base de données pour enregistrer et charger les informations. On a aussi travaillé pour que la vue Kanban soit vraiment utilisable : le drag-and-drop fonctionne, on peut ajouter des tâches dans la colonne sélectionnée, ajouter ou supprimer des colonnes, et tout se met à jour correctement. Pour finir, on a mis en place le patron Décorateur, ce qui nous a permis d'ajouter des étiquettes et des descriptions sur les tâches afin de les personnaliser plus facilement.

2. Fonctionnalités et cas d'utilisation

Dans cette troisième itération, le but était surtout de connecter l'application à la base de données et de rendre la vue Kanban vraiment utilisable. Les fonctionnalités ci-dessous résument ce qu'on a mis en place, avec à chaque fois l'objectif, le fonctionnement global, l'utilisateur visé et le développeur responsable :

Intégration du DAO et connexion à la base de données

- Objectif : pouvoir enregistrer et charger les données (projets, colonnes, tâches) depuis la base.
- Déroulement : au lancement, l'appli récupère les infos, et chaque action (création, suppression, modif) est sauvegardée via le DAO.
- Utilisateur : utilisateur principal.
- Dév : Julien, Enzo

Fonctionnement complet de la vue Kanban

- Objectif : afficher un tableau Kanban interactif relié aux données réelles.
- Déroulement : les colonnes et tâches sont chargées depuis la BD, puis affichées

dans l'interface.

- Utilisateur : utilisateur principal.
- Dév : Matthieu

Drag-and-drop de tâches

- Objectif : permettre à l'utilisateur de déplacer visuellement une tâche d'une colonne à une autre.
- Déroulement : l'utilisateur clique-dépose la tâche ; l'interface met à jour l'affichage et le DAO sauvegarde le changement en BD.
- Utilisateur : utilisateur principal.
- Dév : Matthieu

Ajout d'une tâche dans une colonne sélectionnée

- Objectif : faciliter la création de tâches directement dans la bonne colonne
- Déroulement : l'utilisateur clique sur une colonne, saisit le nom de la tâche, et elle est ajoutée en BD puis affichée.
- Utilisateur : utilisateur principal.
- Dév : Matthieu.

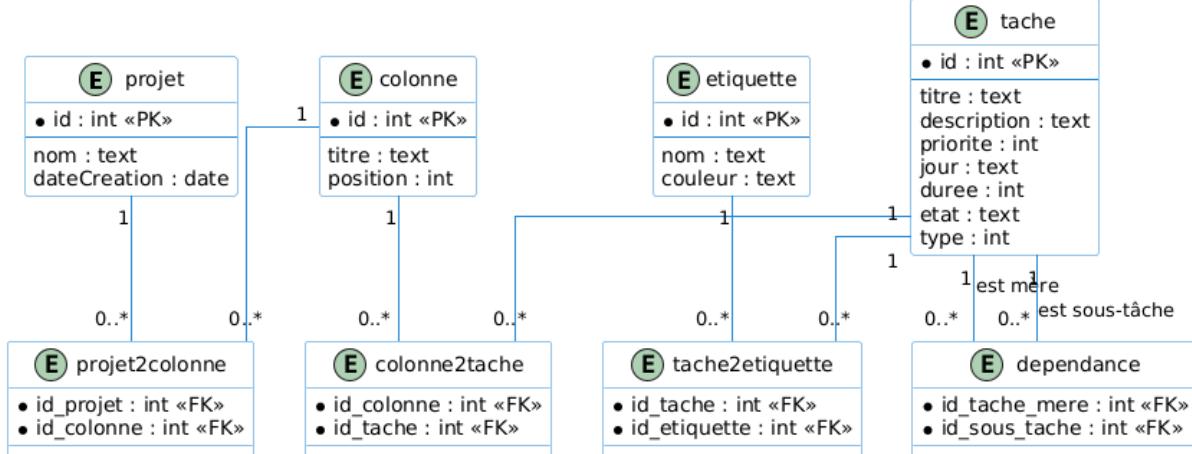
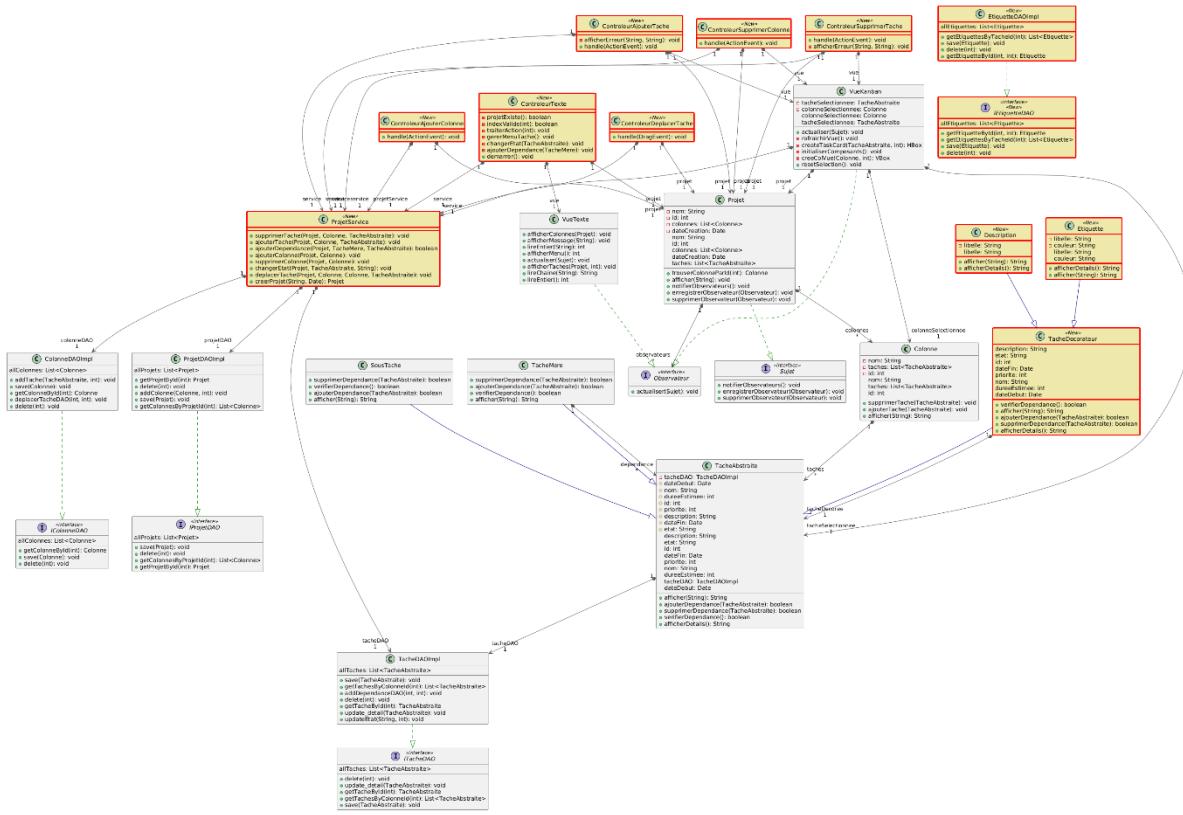
Ajout et suppression de colonnes via la vue Kanban

- Objectif : permettre de modifier la structure du tableau visuellement.
- Déroulement : l'utilisateur crée ou supprime une colonne, la vue se met à jour et la BD est modifiée via le DAO.
- Utilisateur : utilisateur principal.
- Dév : Matthieu

Mise en place du Décorateur pour gérer des étiquettes de tâches

- Objectif : enrichir visuellement une tâche avec des éléments optionnels (ex : labels, catégories, couleurs...).
- Déroulement : l'utilisateur ajoute une étiquette à une tâche ; le décorateur l'encapsule et l'affichage est mis à jour.
- Utilisateur : utilisateur principal.
- Dév : Nacime

3. Diagramme de classes et celui de la BDD et choix de conception



Pour cette troisième itération, on a surtout cherché à faire évoluer l'architecture pour qu'elle puisse fonctionner avec une vraie base de données et une interface Kanban interactive. On a donc gardé l'organisation globale du modèle (Projet → Colonnes → Tâches), mais en ajoutant de nouvelles couches pour gérer la persistance, l'affichage et les interactions.

Logique d'architecture

Le modèle tourne maintenant autour du trio Modèle / DAO / Vue.

Le Projet reste l'élément racine et stocke les colonnes.

Chaque Colonne gère une liste de tâches.

Les Tâches restent hiérarchiques, mais leur gestion passe désormais par un service métier (ProjectService) et par les DAO pour la sauvegarde automatique.

On a aussi ajouté plusieurs contrôleurs (ajout, suppression, déplacement...) qui dialoguent directement avec la vue (VueKanban, VueTexte) et le service. Cette organisation nous permet :

- d'enregistrer et charger les données via le DAO,
- de manipuler les colonnes et tâches depuis la vue Kanban,
- de mettre à jour automatiquement la vue quand un modèle change.

Patrons utilisés

Dans cette itération, plusieurs patrons de conception ont été intégrés :

- DAO : central pour séparer l'accès aux données du reste du code et faire tourner l'application avec une base.
- Décorateur : utilisé pour ajouter des étiquettes aux tâches sans modifier directement la classe de base.

Conformité avec les objectifs de l'itération

Les choix faits correspondent complètement à ce qu'on voulait atteindre :

- l'application fonctionne enfin avec une vraie persistance,
- les vues (notamment le Kanban) réagissent aux actions utilisateur (drag and drop, ajout, suppression),
- la structure reste propre et évolutive grâce aux patrons,
- on a préparé le terrain pour les prochaines itérations (plus de décorateurs, filtres, autres vues, etc.).

4. Bilan de l'itération

Dans cette troisième itération, on a surtout réussi à faire fonctionner l'application avec la base de données grâce à l'intégration du DAO. Les projets, colonnes et tâches peuvent maintenant être enregistrés et rechargés sans problème. On a aussi rendu la vue Kanban vraiment utilisable : on affiche les données, on peut ajouter ou supprimer des colonnes, créer des tâches dans la colonne sélectionnée et déplacer les tâches avec du drag-and-drop. Enfin, on a ajouté le patron Décorateur pour permettre

d'associer des étiquettes aux tâches. Au final, l'application est plus stable, plus pratique à utiliser et mieux organisée pour la suite.

5 Bugs et difficulté

Pendant l'intégration du DAO, on a eu plusieurs soucis. Au début, on avait directement mis des attributs DAO dans les classes concrètes (Projet, Colonne, Tâche) et on modifiait leurs méthodes pour qu'elles appellent les DAO afin de mettre à jour la base. On s'est vite rendu compte que ce n'était pas une bonne idée, parce que les classes du modèle ne sont pas censées connaître la base ni s'occuper de leur propre sauvegarde. Du coup, on a créé une classe service qui centralise tous les DAO et on a déplacé les méthodes dedans pour gérer la persistance proprement. On a aussi rencontré quelques petits bugs côté vue texte : par exemple, pour supprimer une colonne, on récupérait son indice dans la liste au lieu de récupérer son id, ce qui faisait qu'aucune suppression n'était faite dans la liste ni dans la base.