Université de Sherbrooke

Faculté des sciences

IFT785 : Approches orientées objet

Projet de session

Auteur :

Hannachi Mallek

hanm1601@usherbrooke.ca

CIP : hanm1601

Présenté à :

M. Hubert Kenfack Ngankam

Sherbrooke, le 11 avril 2025

Table des matières

1 Introduction

Ce rapport présente le projet de développement d'une application de gestion des ressources humaines, réalisée dans le cadre du cours **IFT785**. L'objectif principal était de développer une application respectant les **principes SOLID**, d'appliquer les bonnes pratiques de développement, d'utiliser des **design patterns** appropriés, et de garantir une couverture de tests suffisante pour assurer la robustesse du système.

Le rapport présente l'architecture du système, la conception de l'API, les choix technologiques, et la stratégie de test mise en place pour assurer la fiabilité de l'application.

2 Architecture de l'Application

2.1 Architecture en Couches

L'architecture suit une approche en trois couches principales :

- **Couche API** : contient les routeurs responsables des points de terminaison. Ces routeurs traitent les requêtes HTTP, valident les données et transmettent les résultats à la couche service.

  Exemples :

  — `app/api/employees.py` : requêtes liées aux employés.

  — `app/api/leave_api.py` : gestion des demandes de congé.

- **Couche Service** : contient la logique métier, comme la validation des données et l'application des règles de gestion.

  Exemples :

  — `app/services/employee_service.py`

  — `app/services/leave_service.py`

- **Couche Repository** : accède aux données de la base. Elle est responsable des opérations CRUD.

  Exemples :

  - `app/repositories/employee_repository.py`
  - `app/repositories/leave_repository.py`

- **Couche Modèle** : représente les entités de la base de données avec SQLAlchemy.

  Exemples :

  - `app/models/employee.py`
  - `app/models/leave.py`

Cette architecture assure une **séparation claire des responsabilités**, une maintenabilité accrue, et facilite l'extension du système.

3 Diagramme de classes

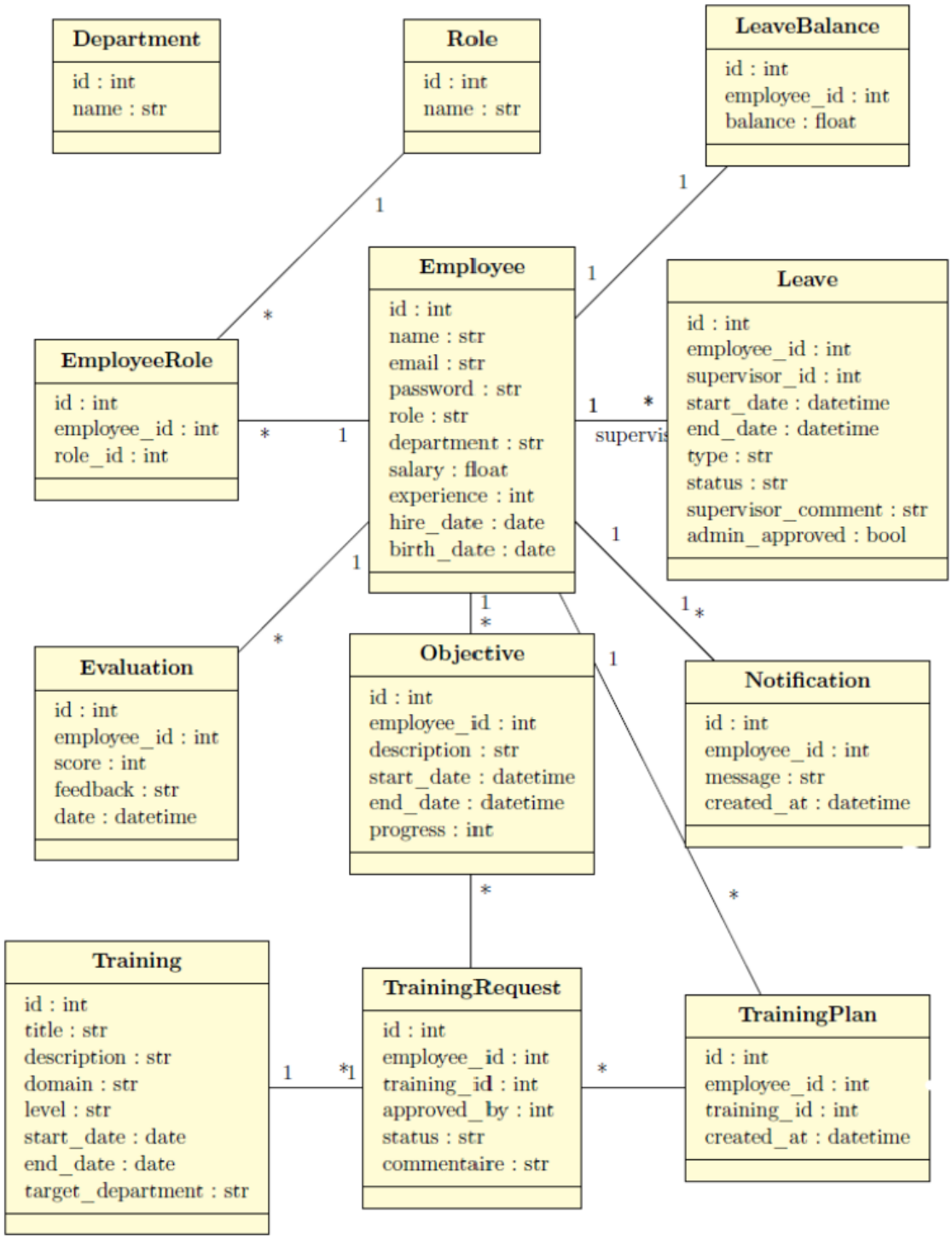

FIGURE 1 – Diagramme de classes structuré de l'application RH

4 Modélisation des entités

Le diagramme de classes présenté à la figure 1 représente la structure des entités principales du système de gestion des ressources humaines (RH). Il illustre les différentes classes, leurs attributs ainsi que les relations entre elles.

— **Employee** est la classe centrale du système. Elle contient les informations personnelles, professionnelles et contractuelles de chaque employé.

— Les classes **Role** et **Department** permettent de structurer l'organisation. Un employé peut être lié à plusieurs rôles via la classe associative **EmployeeRole**.

— La gestion des congés est modélisée via les classes **Leave**, **LeaveBalance** et les relations supervisées. Chaque congé est soumis à un superviseur et validé ou rejeté.

— Les classes **Evaluation** et **Objective** permettent de suivre la performance individuelle des employés, avec des objectifs mesurés dans le temps.

— La classe **Notification** assure la communication interne en envoyant des messages liés aux demandes, validations ou commentaires.

— La formation est structurée autour des classes **Training**, **TrainingRequest**, **TrainingPlan** et **EmployeeTraining**, permettant de gérer les offres de formation, les demandes des employés et les plans personnalisés.

5 Description des Design Patterns Utilisés

L'application repose sur plusieurs **design patterns** issus de la programmation orientée objet. Leur utilisation vise à assurer une architecture modulaire, extensible, testable et conforme aux principes SOLID.

MVC (Model-View-Controller)

Permet de structurer l'application en trois couches : les modèles (représentation des données), les vues (affichage), et les contrôleurs (logique de traitement).

— `Models` : `app/models/`

— `Views` : `frontend/templates/`

— `Controllers` : `app/routes/`, `app/api/`

Pourquoi ? Facilite la séparation des responsabilités, rendant le code plus maintenable et testable.

Singleton Pattern

Assure qu'une seule instance d'un service est créée.

— `app/services/auth_service.py`

Pourquoi ? Centralise la gestion de l'authentification dans une seule instance partagée.

Factory Method Pattern

Encapsule la création d'objets.

— `app/factories/employee_factory.py`

— `app/factories/app_factory.py`

Pourquoi ? Permet de créer dynamiquement des objets sans coupler le code à des classes spécifiques.

Observer Pattern

Permet à un objet de notifier automatiquement ses observateurs lors d'un changement d'état.

— `leave_service.py`, `notification_service.py`

Pourquoi ? Notifie automatiquement les superviseurs et employés des changements liés aux congés.

Strategy Pattern

Permet d'interchanger dynamiquement des algorithmes.

— `leave_service.py`

Pourquoi ? Adapte la validation des congés selon leur type sans modifier la structure du code principal.

Command Pattern

Encapsule une requête comme un objet.

— `leave_service.py`, `dashboard__controller.py`

Pourquoi ? Facilite l'historique, la file d'attente et l'annulation d'actions (approbation, rejet).

Facade Pattern

Fournit une interface simplifiée à un ensemble complexe de sous-systèmes.

— `leave_workflow_facade.py`

Pourquoi ? Regroupe les appels aux services de congé, de notification et de solde dans une seule façade propre.

DAO (Data Access Object) Pattern

Fournit une interface d'accès aux données abstraite.

— `employee_repository.py`, `leave_repository.py`

Pourquoi ? Sépare la logique métier de l'accès aux données (CRUD), facilitant les tests unitaires.

Decorator Pattern

Ajoute dynamiquement des fonctionnalités à un objet.

— `notification_service.py`

Pourquoi ? Permet de configurer dynamiquement des notifications (format, priorité, canal de communication).

6 Documentation de l'API

L'API expose plusieurs *endpoints* RESTful :

— `GET /api/employees` : Récupère la liste des employés.

— `PUT /api/employees/{id}` : Modifie un employé.

— `DELETE /api/employees/{id}` : Supprime un employé.

— `POST /api/employees` : Crée un nouvel employé.

Chaque route est validée avec `Pydantic` et documentée avec Swagger (FastAPI).

7 Technologies Utilisées

— **Python** : langage principal.

— **FastAPI** : framework web rapide et asynchrone,Facilité d'intégration,Documentation automatique.

— **PostgreSQL** : base de données relationnelle,Fonctionnalités avancées,Scalabilité.

— **SQLAlchemy** : ORM utilisé pour les modèles.

— **Docker** : conteneurisation.

— **GitLab** : CI/CD, versioning.

8 Démonstration Fonctionnelle

Tableau de Bord

FIGURE 2 – Interface d'accueil pour l'admin de l'application HannaWork

Le tableau de bord permet :

— Visualisation des informations des employés.

— Gestion des demandes de congés.

— Affichage des notifications.

— Validation des formations.

L'interface utilisateur de l'application HannaWork est conçue pour être simple, claire et intuitive. Elle permet à l'utilisateur d'accéder rapidement aux fonctionnalités principales : gestion des employés, des congés, des formations et des évaluations.

9 Stratégie de Tests et Couverture

La robustesse de l'application a été vérifiée par une série de tests organisés selon trois niveaux : tests unitaires, tests d'intégration et tests end-to-end. La couverture globale des tests atteint **67%**, ce qui dépasse la couverture minimale exigée de 50%.

Tests Unitaires

Les tests unitaires ont pour but de valider le bon fonctionnement de fonctions ou méthodes isolées, sans dépendances externes.

— `test_leave_service.py` : teste les méthodes de création de demandes de congé.

— `test_evaluation_service.py` : vérifie la création et la récupération d'évaluations des employés.

Tests d'Intégration

Les tests d'intégration permettent de valider l'interaction entre plusieurs composants (services, base de données, API).

— `test_integration_leave_request.py` : vérifie le cycle complet d'une demande de congé (requête, traitement, enregistrement).

Tests End-to-End (E2E)

Les tests E2E simulent le comportement réel d'un utilisateur final en interagissant avec l'ensemble du système.

— `test_e2e_training_request.py` : simule une demande de formation depuis la soumission jusqu'à la validation, incluant tous les traitements API et enregistrement en base.

Résumé de la Couverture de Code

Le tableau ci-dessous présente un extrait des résultats de couverture obtenus :

— `app/services/leave_service.py` : **68%**

— `app/services/evaluation_service.py` : **52%**

— `app/repositories/employee_repository.py` : **61%**

— `app/repositories/leave_repository.py` : **34%**

— `app/api/dashboard_admin.py` : **100%**

— `app/main.py` : **81%**

— `Total global` : **67%**

La couverture a été mesurée à l'aide de l'outil `pytest-cov` sur l'environnement de test configuré dans GitLab CI.

Conclusion

L'ensemble des tests réalisés garantit un niveau élevé de fiabilité du système, tout en assurant que les interactions critiques de l'application respectent les exigences fonctionnelles et techniques du projet.

10 Exemples Concrets d'Utilisation

Cette section illustre la manière dont l'application HannaWork répond à des scénarios d'usage réels dans un contexte de gestion des ressources humaines. Ces cas démontrent l'efficacité de l'automatisation des processus internes et l'intégration fluide entre les différents modules.

Gestion des Congés

Un employé souhaite poser des congés du 10 au 15 juillet. Le système effectue les étapes suivantes :

— Vérifie automatiquement le solde de congés.

— Consulte le calendrier de l'équipe pour éviter les conflits.

— Envoie automatiquement la demande au responsable.

— Notifie l'employé dès qu'une décision est prise.

— Met à jour le planning de l'équipe en cas d'acceptation.

Système de Formation

Une développeuse souhaite suivre une formation en cybersécurité. Le système :

— Affiche le catalogue des formations disponibles.

— Suggère des formations adaptées à son profil.

— Permet la sélection et la soumission d'une demande.

— Notifie le superviseur pour approbation.

— Génère automatiquement un plan de formation une fois la demande validée.

Évaluations Annuelles

Dans le cadre de l'évaluation de fin d'année, le système :

— Prépare les formulaires d'évaluation.

— Rappelle les objectifs fixés en début d'année.

— Centralise les feedbacks des intervenants.

— Génère un rapport de synthèse final.

Ces scénarios démontrent la capacité de l'application à automatiser efficacement les processus RH tout en restant alignée avec les besoins métiers.

11 Conclusion

Le projet HannaWork représente une base fonctionnelle solide pour un système de gestion des ressources humaines, développée dans un cadre académique sur une période de six semaines.

L'architecture modulaire respectant les principes SOLID, l'utilisation judicieuse de plusieurs design patterns et l'intégration de tests à différents niveaux témoignent d'un effort structuré et rigoureux. Le système est capable de répondre à des scénarios réalistes tels que la gestion des congés, des formations et des évaluations.

Cependant, bien que l'application couvre un large éventail de fonctionnalités, elle n'est pas encore entièrement prête à être déployée en environnement de production. Plusieurs aspects pourraient être améliorés, notamment la sécurité, l'ergonomie de l'interface, la gestion fine des rôles utilisateurs ou encore la scalabilité.

Pour un projet de session limité dans le temps, HannaWork démontre néanmoins une conception sérieuse, une bonne maîtrise des outils modernes de développement, et constitue une base extensible pour un produit plus complet dans un contexte professionnel.

Table des figures