09/01/2026

Miniprojet N° 2 – Gestion des congés

Adonaï Clervil

1. Encapsulation et intégrité des données

a. Quels attributs avez-vous choisi de rendre privés ou protégés dans vos classes ?

Dans nos classes modèles (Employe, Personne), nous avons utilisé l'encapsulation pour protéger les données sensibles. Par exemple, le « solde_conges » de l'employé est une donnée critique. Bien que Python utilise la convention du souligné (_solde_conges), l'accès se fait via des propriétés (getters/setters). Cela permet de contrôler toute modification de la donnée.

b. Donnez un exemple de méthode qui protège l'intégrité des données (par exemple, empêcher un solde négatif) et expliquez comment cela se combine avec les mises à jour en base.

La méthode traiter_demande dans GestionConges protège l'intégrité du solde.

- Logique : Avant de valider un congé, le service compare le solde actuel en base avec les jours calculés par l'objet DemandeConge.
- Combinaison base/objet : Si solde_actuel >= jours_a_deduire, la mise à jour SQL est déclenchée. Sinon, l'opération est avortée. Cela garantit qu'aucune transaction SQL ne peut mener à un solde négatif, car la validation métier précède l'écriture.

2. Choix de l'héritage

a. Pourquoi avez-vous choisi d'utiliser (ou non) une hiérarchie de classes pour les types de congé ?

Nous avons choisi une hiérarchie (CongeAnnuel, CongeMaladie, etc.) car chaque type possède des règles de calcul différentes. Utiliser l'héritage évite les structures conditionnelles (if/else) complexes et répétitives dans la logique métier, rendant le code plus lisible et robuste.

b. Donnez un exemple de méthode polymorphe dans votre modèle (même signature, comportement différent) et indiquez comment les données associées sont stockées dans la table demandes_conge.

La méthode jours_a_deduire() est l'exemple parfait.

- Comportement : Elle retourne le nombre réel de jours pour CongeAnnuel, mais retourne 0 pour CongeMaladie.
- Stockage : Dans la table demandes_conge, le type est stocké via une colonne type_conge (alimentée par la propriété type_label de l'objet). Lors de la lecture, le service utilise ce label pour réinstancier la bonne sous-classe.

3. Conception de la classe de gestion et DAO

a. Quelles sont les responsabilités principales de la classe GestionConges (ou équivalent) ?

C'est le chef d'orchestre (couche Service). Ses responsabilités sont :

1. Appliquer les règles métier (vérification des droits RH via AuthService).
2. Coordonner les actions entre les modèles objets et la persistance (DAO).

3. Calculer les conséquences d'une action (déduction de solde).

b. Citez au moins un exemple de méthode qui ne devrait pas appartenir à la classe Employe mais à la classe de gestion (ou à une couche DAO), et justifiez.

La méthode modifier_solde ne doit pas appartenir à la classe Employe. Car un objet Employe représente une entité de données. Lui donner la responsabilité de se mettre à jour en base de données violerait le principe de responsabilité unique. C'est le rôle du EmployeDAO.

c. Expliquez comment vous avez organisé les méthodes liées aux requêtes SQL pour éviter de les mélanger avec la logique métier.

Nous avons séparé le code en dossiers : dao/ contient uniquement le SQL, et services/ contient la logique. Le service appelle les méthodes du DAO (ex: self.conge_dao.ajouter(demande)) sans jamais manipuler de requêtes INSERT ou UPDATE directement.

4. Évolutivité du système

a. Comment pourriez-vous adapter votre modèle si, à l'avenir, l'entreprise souhaite gérer d'autres types d'absences (formation, télétravail, mission) ?

Pour gérer le télétravail, les missions ou les formations, il suffit de créer de nouvelles sous-classes héritant de DemandeConge.

b. Modifications nécessaires :

- Modèle objet : Créer CongeFormation(DemandeConge) et définir sa règle dans jours_a_deduire().
- Schéma DB : Aucune modification nécessaire ! La colonne type_conge est une chaîne de caractères capable d'accueillir le nouveau label.
- Code d'accès (DAO) : Aucune modification. Le DAO est générique, il enregistre l'objet qu'on lui donne.

5. Lien avec les concepts du cours

a. Exemples concrets dans le code :

- Encapsulation : Utilisation de AuthService._utilisateur_connecte pour cacher l'état de la session.
- Héritage : Responsable(Personne) et Employe(Personne).
- Polymorphisme : jours_a_deduire() qui change selon le type de congé.
- Abstraction : La classe DemandeConge(ABC) qui force les sous-classes à implémenter leurs propres règles.

b. Intérêt de la POO et SQLite :

La POO permet de modéliser le monde réel (Employés, RH, Congés) sous forme d'objets interactifs, ce qui facilite la maintenance dans une entreprise où les règles RH changent souvent. L'usage de SQLite ajoute une couche de réalisme cruciale : les données ne sont plus volatiles.

Elles persistent après la fermeture du programme, simulant ainsi un véritable environnement de production sécurisé.