HEI

Haute École d'Informatique

Penser - Travailler - Impacter

PROG3 - Implémentation d'API REST

TD2 - Application de gestion de restaurant

Contexte fictif :
La chaîne de restauration "La Gastronomie Pizza" est une référence nationale du fast-food à Madagascar depuis plus d'une vingtaine d'années. Elle possède des points de vente partout à Madagascar, dont plusieurs points de ventes implantés dans les grandes villes. Notre application a deux objectifs principaux :

- Au sein de chaque point de vente, faciliter la gestion des stocks, la gestion des commandes et des ventes, ainsi que le flux de trésorerie à travers les ventes et achats effectués.
- Au sein du siège, qui effectue un suivi centralisé des opérations de chaque point de vente, avoir un dashboard qui synthétise les données issues de tous les points de vente en temps réel.

Dans le cadre de notre projet, nous allons faire évoluer petit à petit l'application en développant petit à petit les fonctionnalités attendues.

Partie 1 : Gestion des plats

Le cœur de l'activité de l'entreprise est de proposer des plats et des menus à leurs clients. Nous allons commencer ainsi par le cœur même du métier.

a. Un **Plat ou Dish (en anglais)** possède un nom, un prix de vente unitaire fixe, et il est composé de plusieurs **Ingrédients** ou **Ingredient (en anglais).**

b. Un **Ingredient** possède un nom, la date et heure de dernière modification, un prix unitaire fixe et une unité qui peut être soit en grammes (g), en litres (L) ou en unité (U). Par exemple, on peut avoir les ingrédients suivants :

ID	Nom	Date et heure de dernière modification	Prix unitaire	Unité
1	Saucisse	2025-01-01 00:00	20	G
2	Huile	2025-01-01 00:00	10 000	L
3	Oeuf	2025-01-01 00:00	1 000	U
4	Pain	2025-01-01 00:00	1 000	U

c. Il est à noter qu'un **Dish** doit être composé d'au moins un Ingredient, et un Ingredient peut être retrouvé dans plusieurs plats ou dish différents. Pour chaque **Dish,** il faut spécifier la quantité unitaire nécessaire à chaque Ingredient qui le compose, quantité unitaire veut dire qu'il faut à la fois spécifier la quantité nécessaire (exprimé en nombre décimale) ainsi que l'unité évoqué plus tôt, soit en gramme, soit en litre, soit en unité.
Voici une illustration d'un plat :

ID	Nom	Prix de vente unitaire
1	Hot dog	15 000

Et voici les ingrédients qui composent le plat "Hot Dog" :

Ingrédients	Quantité nécessaire	Unité
Saucisse	100	G
Huile	0,15	L
Oeuf	1	U
Pain	1	U

Notez que ces ingrédients ou leurs quantités peuvent changer et donc sont variables. Par changer les ingrédients, nous voulons dire qu'on peut à la fois ajouter d'autres ingrédients comme en supprimer.

Travail à faire à corriger d'ici samedi 22 février 2025 :

1. Créez un nouveau projet sur Java, de type console. Installer les dépendances nécessaires pour pouvoir connecter l'application avec une base de données Postgres, ainsi que d'effectuer des tests unitaires avec JUnit.

2. Créer les scripts de création du schéma de la base de données correspondantes et insérer dans le projet, dans un package db.migration, et ajouter des données de test selon les illustrations données plus tôt (Hot Dog et ses ingrédients avec la même quantité de composition et les mêmes prix) dans un autre package db.testdata. En particulier, les unités doivent être exprimés en ENUM directement dans la base de données.

3. Créer les classes nécessaires pour pouvoir convertir les données issues de la table en objet sur Java, notamment la DataSource, les DAO, etc. En particulier :

    a. Les unités doivent être exprimées en type ENUM et non en tant que String sur Java.

    b. La classe **Dish** (désignant un **Plat**), doit directement avoir un attribut de type liste d'ingrédients qui le compose ainsi que les quantités unitaires nécessaires.

4. À travers les données de tests énoncés précédemment, ainsi que la structure des classes évoquées plus tôt, vérifiez à travers un test d'intégration (non mockées) que l'ensemble des coûts des ingrédients du plat Hot Dog = 5500.

    PS : n'oubliez pas de configurer les variables d'environnement avant de lancer les tests avec les valeurs réelles.

5. Créez une fonction qui permet de filtrer, trier et paginer la liste des ingrédients et ajouter les tests nécessaires pour vérifier que ça marche. En particulier, on peut filtrer et trier en même temps par nom, par unité, par un intervalle de prix, une intervalle de date de dernière modification ou tous ces attributs en même temps, et à la toute fin, les données retournées doivent être paginées correctement.

HEI

Haute École d'Informatique

Penser - Travailler - Impacter

TD2 - Application de gestion de restaurant | Partie 1 : Guide pour mieux comprendre

Pour ceux qui rencontrent des difficultés sur les notions à maîtriser, nous allons décomposer les travaux à faire selon les matières que vous avez eu en L1 (PROG et DONNÉES), et suivez les étapes dans cet ordre.

Pour la partie SQL :

1. Créer d'abord un nouvel utilisateur avec les informations de connexion (identifiant et mot de passe). Pour faire cela, il faut employer les syntaxes CREATE USER nom_user WITH PASSWORD 'votre mot de passe', qui pour rappel est équivalent à CREATE ROLE nom_user WITH LOGIN PASSWORD 'votre mot de passe'.

2. Créer la base de données et attribuer les rôles nécessaires au nouvel utilisateur pour pouvoir effectuer les manipulations des schémas et des données pour la base de données nouvellement créée. La meilleure façon est de n'attribuer (GRANT) que les autorisations relatives à l'usage attendu qui est le CRUD, mais vous pouvez directement attribuer tous les privilèges pour éviter que ça ne vous ralentisse. Nous pouvons discuter plus de détails lors des séances en présentiel TD.

3. Créer les tables et les types correspondantes aux données décrites précédemment dont voici la représentation dans le modèle relationnel (colonnes soulignées sont des clés primaires, ceux qui sont préfixés d'un hashtag (#) sont des clés étrangères, et si ils sont à la fois soulignés et préfixés d'un hashtag donc à la fois clés primaires et étrangères)

    a. Dish (id_dish, name, unit_price)
    b. Ingredient (id_ingredient, name, unit_price, unit, update_datetime)
    c. Dish_Ingredient (#id_dish, #id_ingredient, required_quantity, unit)
    d. Unit de type ENUM : G, L, U

    L'intérêt d'avoir une table de jointure Dish_Ingredient ici est de faire en sorte qu'on n'ait pas à ajouter une nouvelle entrée Ingredient pour chaque plat, sachant que les propriétés telles que les prix sont les même pour n'importe quel ingrédient, qu'ils composent le plat "Hot Dog" ou le plat "Omelette", pour le cas de l'oeuf par exemple.

4. Pour obtenir les ingrédients d'un plat ainsi que les quantités nécessaires, il suffit d'utiliser la table de jointure Dish_Ingredient, qui permet de gérer les relations many-to-many, et dans notre cas présent, associer un ingrédient à un plat. Il est tout à fait possible d'effectuer le calcul qui confirme la sommes des coûts des ingrédients

du Hot Dog = 5500, en utilisant proprement la fonction SUM() et GROUP BY (). Les données concernées sont : unit_price * required_quantity pour obtenir le coût lié à chaque Ingrédient composant le plat et id_dish pour regrouper le calcul par le plat concerné.

5. Pour les filtres, tris et paginations :
    a. Exécutez un script SQL qui permet d'obtenir la liste des ingrédients qui contiennent "u" dans leur nom, dont le prix unitaire est inférieur ou égale à 1000. Normalement vous devez obtenir uniquement les ingrédients "Saucisse" et "Oeuf" comme résultat. Pour cela, vous devez employer correctement la syntaxe "WHERE" qui permet d'ajouter des conditions, "ILIKE" ou "LIKE" pour des comparaison de chaînes de caractères basés sur des patterns qui utilisent des caractères spéciaux comme "%", ou encore les opérateur < ou > ou = pour comparer des valeurs dans une conditions.
    b. Pour effectuer les tris, ajouter tout juste la syntaxe ORDER BY <colonne à trier> <valeur ASC ou DESC> à la colonne, que vous devez trier. Notez que vous pouvez trier votre résultat à travers plusieurs colonnes à la fois et selon l'ordre des colonnes à trier que vous allez établir, les résultats peuvent changer. Par exemple, **ORDER BY name ASC, unit_price DESC** ne retourne pas forcément le même résultat que **ORDER BY unit_price DESC, name ASC.**
    c. Pour effectuer la pagination, modifier le script SQL précédent en gardant le tri ORDER BY name ASC, unit_price DESC, et essayez d'obtenir uniquement un élément de la deuxième page, autrement dit, chaque page pour nous correspond à un élément. Les mots clés à retenir sont "LIMIT" qui va limiter le nombre de résultats retournés, et "OFFSET" qui va permettre de "pointer le curseur" directement aux résultats attendus. Pour rappel, la première valeur de OFFSET commence à 0. Si tout fonctionne, normalement vous devrez avoir comme résultat l'ingrédient Saucisse.

Pour la partie JAVA :

1. Créer des classes correspondantes aux énoncés ci-dessus.

2. Avant d'effectuer la configuration de la connexion avec la base de données, essayez de créer des données statiques issues d'un ArrayList directement, qui correspondent aux données de tests (Plat "Hot Dog" et ses ingrédients). Plutôt que de prendre les données à manipuler dans la base de données dans le calcul des coûts des ingrédients du plat "Hot Dog", vous allez récupérer les données directement depuis l'ArrayList qui contiennent les données de tests statiques. Vous pouvez utiliser le nom "StaticDataSource" par exemple pour cela.

3. Créer une méthode getIngredientCost() qui retourne un Integer, dans la classe Dish. Dans l'implémentation de cette méthode, vous aurez à itérer tous les éléments de la liste Ingredients à travers une boucle et un accumulateur par exemple pour calculer les coûts de chaque ingrédient d'un plat en fonction de la quantité qui est requise.

   PS : La bonne pratique reste l'utilisation des Stream à la place des boucles classiques (for, while, etc).

4. Créer une classe de test, ajouter la méthode qui va vérifier que le résultat attendu est bien 5500. Respecter la forme du standard de nos tests divisés en 3 parties : les données de test et les résultats attendus en première partie, le sujet de test (la méthode à tester) en deuxième partie, et les vérifications en troisième partie.

5. Une fois que les tests passent, configurer proprement les DAO qui permettent d'obtenir les données de test depuis la base de données et non depuis le StaticDataSource. Notamment, l'objectif est d'avoir une classe DishDAO, qui a une méthode qui retourne un Dish par ID par exemple, implémenter là de sorte à obtenir une liste de Ingredients correspondants et les quantités requises lorsqu'on instancie un nouveau Dish, c'est à dire, lorsqu'il y a new Dish(...). Si vous avez correctement implémenté l'accès aux données, le test qui a marché lorsque les données étaient statiques va toujours continuer de passer.