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.

6. **Consigne supplémentaire ajoutée en séance TD :** historiser les prix de chaque ingrédient. Par défaut, un ingrédient doit avoir au moins un prix d'ingrédient et peut avoir plusieurs pour chaque date différente. Un prix d'ingrédient n'est associé par contre qu'à un et un seul ingrédient. Par défaut, le prix d'un ingrédient correspond au dernier prix qui lui est associé, et nous avons la possibilité d'obtenir le prix composant un plat à une date donnée.

    a. Pour cela, vous pouvez soit modifier la méthode **getIngredientsCost()** de la classe Dish et y ajouter un argument LocalDate qui va spécifier la date de l'ingrédient qu'il faut, ou bien ajouter un argument List<DishIngredient> et donc le calcul va se faire à travers la liste fournie et non la liste associée par défaut à l'objet qui va être instancié de la classe Dish.

    b. Pour vérifier que ça marche, en faisant en sorte que la date par défaut soit égale à la date du jour, dans vos tests, faites en sorte que les prix les plus récents associés aux ingrédients composant le HotDog permettent d'obtenir le coût de 5500. Ensuite pour vérifier que la méthode qui permet de calculer les coûts des ingrédients à une date donnée fonctionne, ajouter à chaque ingrédient des prix à une daté antérieure (avant la date du jour) et vérifier pour cette date donnée si le coût des ingrédients sont bien mis à jour.

7. Ajouter une méthode à la classe Dish qui permet de calculer les marges brutes associées à un plat : **Double getGrossMargin();**
   Cette méthode est obtenue à travers la différence entre le prix de vente unitaire fixe et le coût des ingrédients. Étant donné que les coûts des ingrédients peuvent être variables en fonction de la date, il faut que par défaut, nous obtenions les coûts de revient le plus proche ou égale à la date du jour. Et qu'il y ait une autre possibilité d'obtenir les marges brutes à travers une date donnée, similairement aux coûts des ingrédients.

   Ajouter les tests qui permettent d'assurer que les résultats obtenus sont corrects.

Au stade actuel de notre application, nous avons la possibilité de gérer la composition des plats à travers les ingrédients, et d'identifier les coûts et les marges bénéficiaires. Nous allons maintenant commencer à gérer les stocks de ces ingrédients.

Partie 2 : Gestion des stocks des ingrédients

Gérer les stocks des ingrédients signifie suivre les mouvements d'entrée et de sortie de chaque ingrédient. En effet, lorsqu'un plat est vendu à une date donnée, ça veut dire que les ingrédients ont été utilisés. Pour approvisionner les nouveaux stocks, il est essentiel de faire des achats de nouveaux ingrédients.

Un mouvement de stock peut être donc défini comme suit :
- Type du mouvement : entrée ou sortie
- Quantité de l'ingrédient : exprimé en nombre décimale
- Unité : G, U ou L
- Date et heure du mouvement du stock

Un mouvement correspond à un Ingrédient spécifique, et un Ingrédient peut avoir plusieurs mouvements.

Par défaut, nous allons supposer que le stock initial est égal à 0 pour chaque ingrédient.

Travail à faire d'ici le 26/02/2025 :

1. Ajouter les tables nécessaires dans la base de données et ajouter les scripts dans le package db.migration pour gérer la gestion des stocks. Insérer des données de test qui correspond à ce que le 1er février 2025 à 8h, le stock a été approvsionné comme tel :
   a. Oeuf, 100 unités
   b. Pain, 50 unités
   c. Saucisse, 10 kg donc 10 000 g

   d. Huile, 20 L

2. Ajouter les classes nécessaires pour accéder et manipuler aux nouvelles tables qui permettent de gérer les données issues de la base de données, concernant la fonctionnalité de gestion de stock. En particulier, dans cette question, attardez vous sur la façon de **récupérer uniquement les mouvements de stock**.

3. Dans la classe Ingrédient, ajouter la ou les méthode(s) nécessaire(s) pour obtenir l'état des stocks (quantité et unité disponible) de l'ingrédient à une date donnée. Par défaut, la date de l'état du stock est la date du jour.
   a. Ajouter les tests nécessaires qui vont vérifier que pour nos ingrédients actuels, l'état des stocks correspond bien aux données énoncées précédemment, tant qu'il n'y a aucun mouvement de stock de type sortie associé aux ingrédients (œufs, pain, saucisse, huile).

   b. Maintenant, ajouter dans vos données de test dans la base de données, les mouvements de stocks de type sortie actuels, selon les dates respectives :

      i. Oeufs, 10 unités le 02 février 2025 à 10h

      ii. Oeufs, 10 unités le 03 février 2025 à 15h

      iii. Pain, 20 unités le 05 février 2025 à 16h

      Modifier maintenants les données 'expected' dans vos tests précédemment (qui tests la méthode **Dish::getAvalaibleQuantity**) pour correspondre à ces nouvelles données et faites en sorte que les quantités respectives de chaque ingrédient à la date par défaut (date d'aujourd'hui) soient :

      - Oeuf, 80 unités le 24 février 2025 à l'heure actuelle
      - Pain, 80 unités le 24 février 2025 à l'heure actuelle
      - Saucisse, 10 000 g le 24 février 2025 à l'heure actuelle
      - Huile, 20L le 24 février 2025 à l'heure actuelle

4. Étant donné que durant la question 2, vous ne vous êtes attardées que sur la récupération des mouvements au sein de la base de données, maintenant implémenter la création des mouvements de stock. En particulier, les mouvements ne doivent pas être modifiables, et donc dit immuables. Du moment, où un mouvement qu'il soit une entrée ou sortie est enregistré, on ne peut plus modifier la quantité ou la date, pour éviter les vols.
   a. Pour que les données des tests antérieurs ne soient pas impactés, créer deux nouveaux ingrédients, le sel avec comme unité le G, ainsi que le riz avec comme unité le G. Comme prix unitaire, respectivement, le sel coûte 2,5 Ar par G, et le riz 3,5 Ar par G à la date d'aujourd'hui.

b. Ajouter des mouvements d'entrées et de sortie directement sur ces nouveaux ingrédients à travers des tests et vérifier que les données sont toujours bien à jour.