

Royaume du Maroc



Ministère de l'Education Nationale,  
du Préscolaire et des Sports



# **Rapport de mini projet**

## **Sujet : Réalisation d'une application de Gestion d'Épicerie**

**Réaliser par :**

**Elharhour Zakaria**

# Sommaire

## I. Introduction Générale

- 1.Contexte du marché
- 2.Enjeux
- 3.Avantages de l'application
- 4.Objectifs du projet
  - 4.1. Gestion des produits
  - 4.2. Gestion des ventes et facturation

## II. Besoins utilisateurs

- 1.Exigences fonctionnelles
- 2.Exigences de qualité
- 3.Exigences de performance

## III. Conception

1. Identification des acteurs
2. Identification des cas d'utilisation
3. Regroupement des exigences fonctionnelles par intentions des acteurs
4. Diagramme de cas d'utilisation
5. Classement des cas d'utilisation
6. Planification du projet

## IV. Itérations de développement

### Itération #1 : Ajouter produit

- 1.1 Flow of Events : Ajouter un produit
- 1.2 Diagramme de séquence
- 1.3 Maquette

1.4 Concepts du Domaine

1.5 Diagramme de classes participantes

1.6 Diagramme d'interaction

Itération #2 : Modifier produit

2.1 Description Textuelle

2.2 Diagramme de séquence

2.3 Concept du domaine

2.4 Diagramme des classes participantes

2.5 Diagramme d'interaction

Itération #3 : Rechercher un produit

3.1 Description Textuelle

3.2 Diagramme de séquence

3.3 Concept de Domaine

3.4 Diagrammes des Classes participantes

3.5 Diagramme d'interaction

Itération #4 : Supprimer produit à vendre

4.1 Description Textuelle

4.2 Diagramme de séquence

4.3 Maquette

4.4 Concept du domaine

4.5 Diagramme d'interaction

## V. Tests et Validation

1. Introduction

2. Outils utilisés

3. Configuration Maven

4. Classes de tests

- Classe de test : DBConnectionTest.java

- Classe de test : LigneVenteDAOTest.java

- Classe de test : LigneVenteTest.java

- Classe de test : ProduitDAOTest.java
- Classe de test : ProduitTest.java
- Classe de test : VenteDAOTest.java
- Classe de test : VenteTest.java

#### 5. Rapport de couverture de code (JaCoCo)

### VI. Conclusion

# **I. Introduction Générale**

La gestion d'une épicerie est un secteur commercial nécessitant une organisation rigoureuse pour assurer un service efficace aux clients et une rentabilité optimale. Dans un contexte de numérisation croissante des commerces de proximité, l'utilisation d'outils informatisés devient essentielle pour répondre aux défis quotidiens de gestion.

## **1. Contexte du marché**

Le marché des épiceries de proximité fait face à une forte concurrence des grandes surfaces et du commerce en ligne. La digitalisation des services est désormais primordiale pour optimiser la gestion des produits et offrir un service client de qualité.

## **2. Enjeux**

Les principaux enjeux de la gestion d'une épicerie sont :

- La gestion efficace des produits et de leur prix
- La simplification des processus de vente
- La facturation précise et transparente
- La traçabilité des transactions

## **3. Avantages de l'application**

L'application de gestion d'épicerie présente de nombreux avantages :

- Interface intuitive facilitant les opérations quotidiennes
- Automatisation des transactions de vente
- Facturation immédiate et précise

## **4. Objectifs du projet**

Ce projet vise à développer une application simple mais efficace de gestion d'épicerie. Il se concentre exclusivement sur deux fonctionnalités essentielles :

#### 4.1. Gestion des produits :

- a. Ajout de nouveaux produits
- b. Consultation des caractéristiques des produits
- c. Mise à jour des informations des produits
- d. Suppression des produits

#### 4.2. Gestion des ventes et facturation :

- a. Création de nouvelles ventes
- b. Sélection des produits à vendre
- c. Génération automatique de factures

Cette application a pour ambition d'apporter une solution simple et efficace aux épiceries de taille modeste qui souhaitent digitaliser leur gestion quotidienne.

## **II. Besoins utilisateurs**

### **1. Exigences fonctionnelles**

L'application "Gestion d'Épicerie" devra regrouper les fonctionnalités nécessaires à la gestion des produits et des ventes.

#### 1.1 Gestion des produits:

- R1 : Le vendeur doit pouvoir ajouter un nouveau produit dans le système en spécifiant ses caractéristiques (nom, prix).
- R2 : Le vendeur doit pouvoir consulter la liste de tous les produits disponibles.
- R3 : Le vendeur doit pouvoir mettre à jour les informations d'un produit (nom, prix).
- R4 : Le vendeur doit pouvoir rechercher un produit par son nom.
- R# : Le vendeur doit pouvoir supprimer un produit .

#### 1.2 Gestion des ventes et facturation :

- R5 : Le vendeur doit pouvoir ajouter produits sélectionnés dans le catalogue au panier.
- R6 : Le vendeur doit pouvoir supprimer des produits du panier avant finalisation.
- R7 : Le vendeur peut finaliser une vente et imprimer une facture contenant les produits, le prix et la date d'achat.

### **2. Exigences de qualité**

L'application sera utilisée principalement par des vendeurs qui n'ont pas nécessairement de connaissances approfondies en informatique.

Interface sobre et efficace :

- R8 : L'interface utilisateur doit être intuitive, avec des menus clairs et des formulaires simples à remplir.
- R9 : Les actions fréquentes doivent être accessibles rapidement, avec un minimum de clics.
- R10 : Les formulaires doivent inclure des validations en temps réel pour éviter les erreurs de saisie.

### **3. Exigences de performance**

Une épicerie de taille moyenne nécessite une application réactive et fiable :

- R11 : Le temps de réponse pour les opérations courantes ne doit pas excéder 2 secondes.
- R12 : Le système doit pouvoir gérer une base de données contenant jusqu'à 1000 produits et 10000 ventes.

Pour modéliser les exigences on peut utiliser un diagramme SysML(Extension de UML)

### III. Conception

On peut utiliser un SYSMML Pour modéliser les exigences :

<<requirement>> R1
Text = ""vendeur doit pouvoir ajouter un nouveau produit dans le système en spécifiant ses caractéristiques (nom, prix)."" ID = "REQ001" source = "" kind = "Functional" verifyMethod = "" risk = "Medium" status = "Approved"

<<requirement>> R2
Text = ""Le vendeur doit pouvoir consulter la liste de tous les produits disponibles. " ID = "REQ002" source = "" kind = "Functional" verifyMethod = "" risk = "Low" status = "Approved"

<<requirement>> R3
Text = ". Le vendeur doit pouvoir mettre à jour les informations d'un produit (nom, prix). " ID = "REQ003" source = "" kind = "Functional" verifyMethod = "" risk = "Medium" status = "Approved"

<<requirement>> R4
Text = ". Le vendeur doit pouvoir rechercher un produit par son nom. " ID = "REQ004" source = "" kind = "Functional" verifyMethod = "" risk = "Medium" status = "Approved"

<<requirement>> R5
Text = "Le vendeur doit pouvoir ajouter produits sélectionnés dans le catalogue au panier. " ID = "REQ005" source = "" kind = "Functional" verifyMethod = "" risk = "Medium" status = "Approved"

<<requirement>> R6
Text = "Le vendeur doit pouvoir supprimer des produits du panier avant finalisation. " ID = "REQ006" source = "" kind = "" verifyMethod = "" risk = "Medium" status = "Approved"

<<requirement>> R7
Text = " Le vendeur peut imprimer une fac... contenant les produits, le prix et la date d'achat. " ID = "REQ007" source = "" kind = "Functional" verifyMethod = "" risk = "Medium" status = "Approved"

#### 1. Identification des acteurs

- **Vendeur** : Employé de l'épicerie qui gère les produits et enregistre les ventes

#### 2. Identification des cas d'utilisation

À partir des exigences précédentes, nous pouvons identifier les cas d'utilisation principaux suivants :

##### UC1 : Gestion des produits :

- Ajouter un produit
- Consulter les produits



- Modifier un produit
- Supprimer un produit
- Rechercher un produit

#### **UC2 : Gestion des ventes :**

- Créer une vente
- Ajouter des produits à vendre
- Supprimer des produits de la vente
- Imprimer la facture

### **3. Regroupement des exigences fonctionnelles par intentions des acteurs**

<b>Exigence Fonctionnelle</b>	<b>Intention d'acteur</b>	<b>Acteur</b>
Gestion des produits	Ajouter un produit	Vendeur
	Consulter les produits	Vendeur
	Modifier un produit	Vendeur
	Supprimer un produit	Vendeur
	Rechercher un produit	Vendeur
Gestion des ventes	Créer une vente	Vendeur
	Ajouter des produits à la vente	Vendeur
	Supprimer des produits de la vente	Vendeur
	Finaliser la vente et imprimer la facture	Vendeur

#### 4. Diagramme de cas d'utilisation :

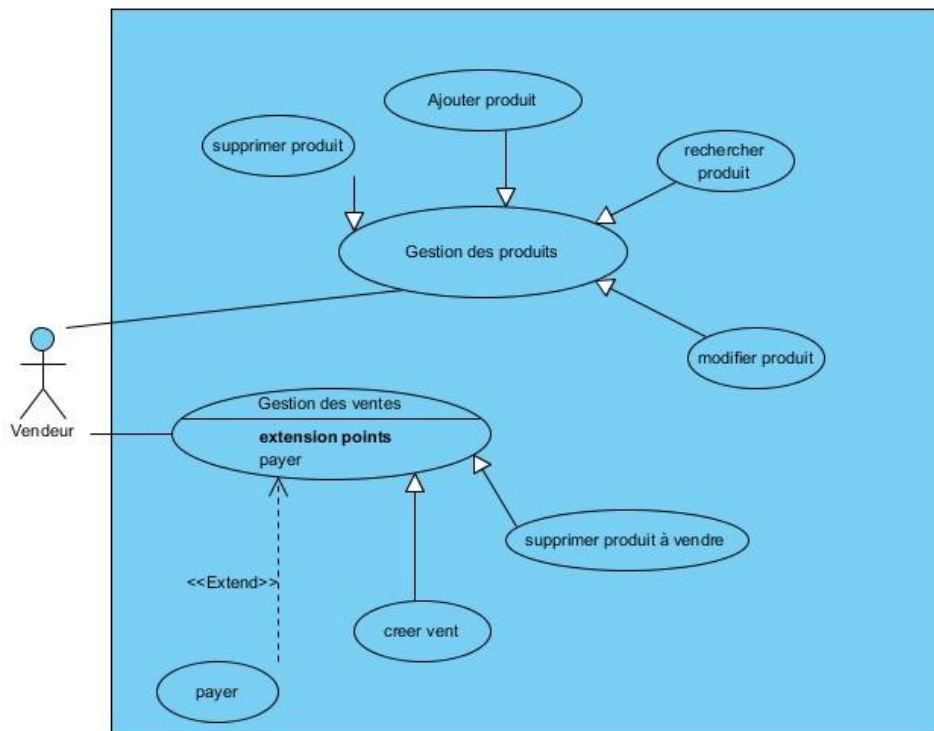


Figure 2 : use case diagramme

#### 5. Classement des cas d'utilisation

Pour classer les cas d'utilisation, il est important de décider qu'elles sont les cas les plus prioritaires et qu'elles sont les cas qui présente de grands risques techniques.

Cas d'utilisation	Priorité	Risque
UC1 : Gestion des produits	Haute	Bas
UC2 : Gestion des ventes	Haute	Moyen

#### 6. Planification du projet

Cas d'utilisation	Priorité	Risque	Itération
UC1 : Gestion des produits	Haute	Bas	1
UC2 : Gestion des ventes	Haute	Moyen	2

## IV. Itérations de développement

### 1. Itération #1 : ajouter produit

Diagramme de séquence : pour le cas ajouter produit :

#### 1.1 Flow of Events : Ajouter un produit

Nom	Ajouter produit
Objectif	Permettre à un vendeur d'ajouter un produit
ID	1
Acteurs principaux	Vendeur
Date	06/03/2025
Responsable	zakaria
Statut	Proposé
Version	V1.0
Précondition	Champs remplis
Scénario nominal	<div>4 vendeur saisit les informations du produit</div> <div>5 SYSTEM vérifie les données saisies.</div> <div>6 SYSTEM ajout le produit</div> <div>7 Fin</div>
Scénario alternatif :	3-a. SYSTEM affiche un message d'erreur

#### 1.2 Diagramme de séquence : flow of events

1. vendeur entre les informations du nouveau produit
2. <b>SYSTEM</b> vérifier les données
2.1 <b>if</b> données valid
2.1.2 <b>SYSTEM</b> ajout le produit

3 else

3.1 SYSTEM affiche message d'erreur

end if

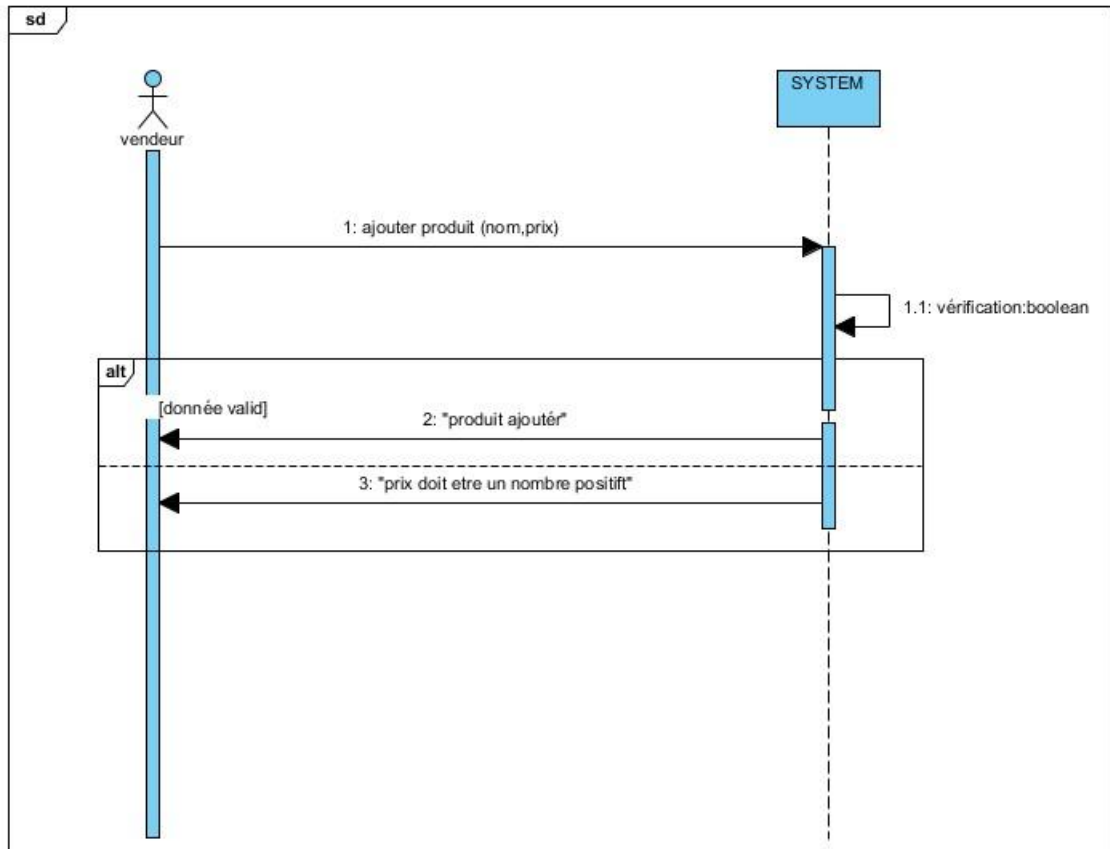
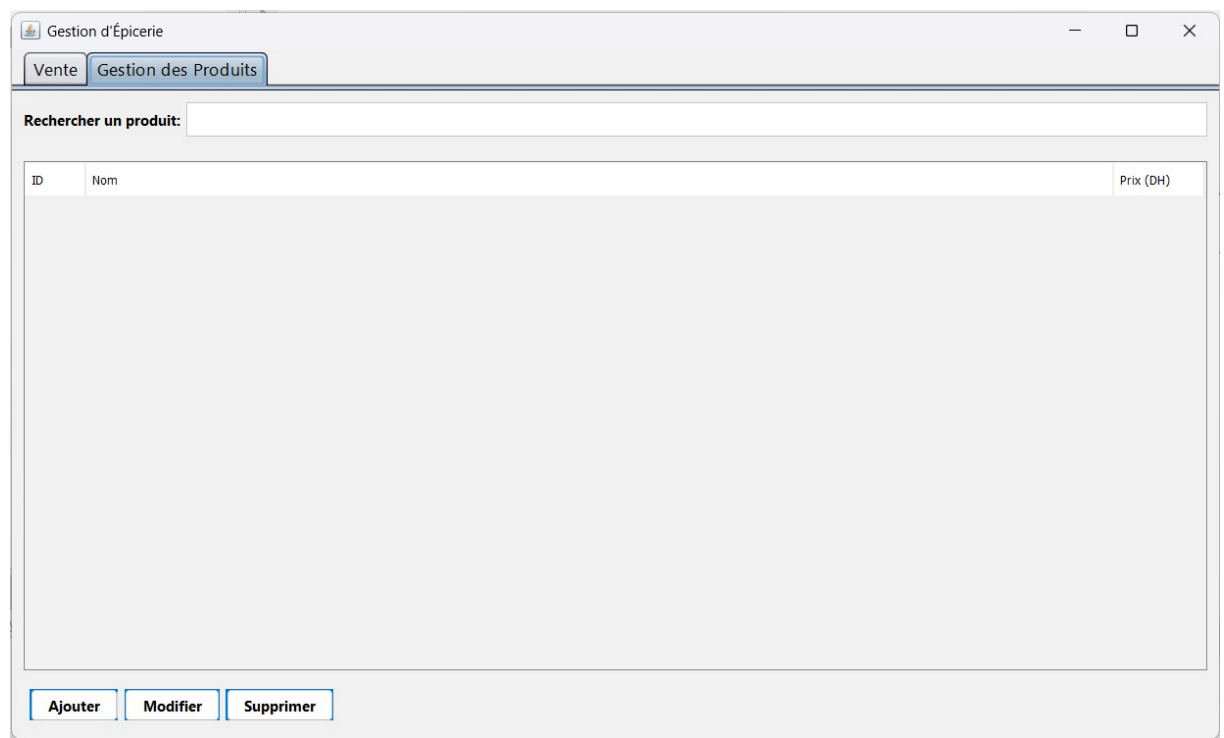
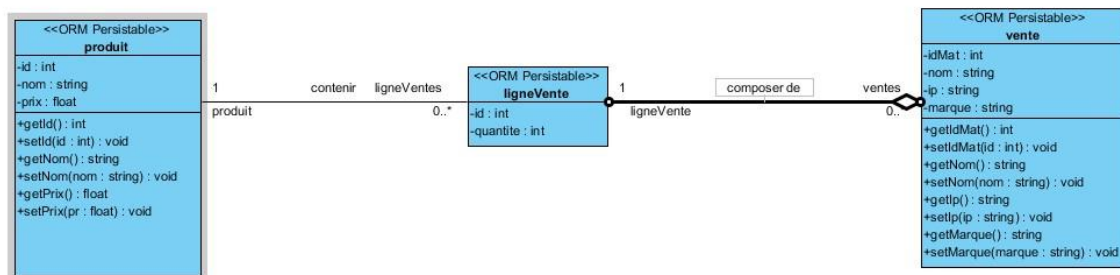


Figure 3 : SD : ajouter nouveau produit

### 1.3 Maquette :



### 1.4 Concepts du Domaine :



**Figure 4 : CD : ajouter nouveau produit**

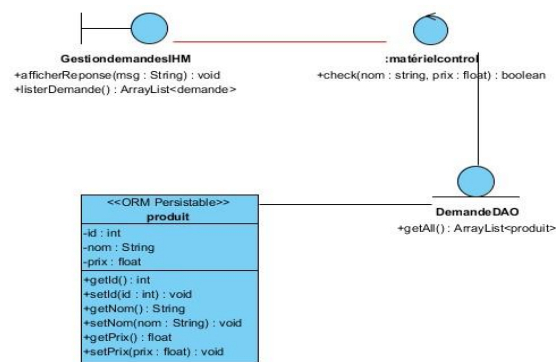
### 1.5 Diagramme de classes participantes

Dans un diagramme de classe participante, on trouve trois types de classes qui sont :

a - **Control Class** contiennent la logique métier pour gérer les interactions entre les classes entité et les classes frontières. Elles sont responsables de la coordination du flux de contrôle dans le système.

b - **ENTITY Class** représentent des objets ou des concepts du monde réel qui ont une existence et une identité persistante dans le système. Elles encapsulent généralement les données et le comportement lié à ces objets.

c - **BOUNDARY Class** agissent comme une interface entre le système et son environnement externe, elles sont responsables de la collecte des entrées de l'utilisateur et de l'affichage des résultats.

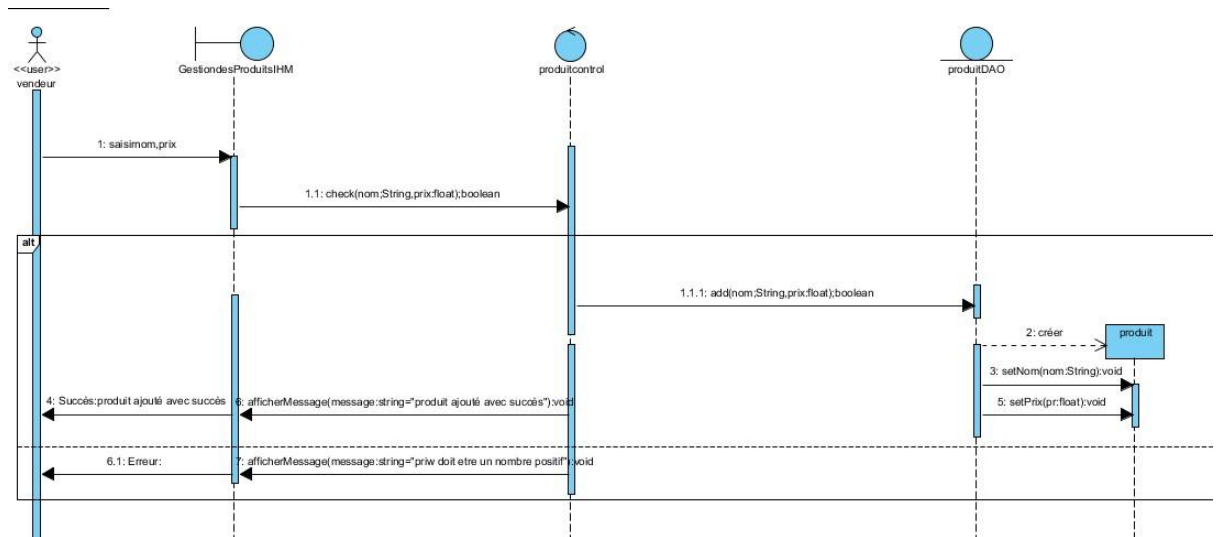


**Figure 5 : CP : ajouter produit**

## 1.6 Diagramme d'interaction

Après avoir trouvé les classes participantes, nous allons établir un diagramme d'interaction :

Le diagramme d'interaction pour le UC1 se présente comme suit :



## 2. Itération #2 : Modifier produit

### 2.1 Description Textuelle

Nom	Modifier produit
Objectif	Permettre à un vendeur de modifier un produit
ID	1
Acteurs principaux	Vendeur
Date	02/03/2025
Responsable	zakaria
Statut	Proposé
Version	V1.0
Précondition	Produit à modifier est sélectionner

Scénario nominal	<ol style="list-style-type: none"> <li>1. Vendeur saisit les nouvelles informations du produit</li> <li>2. SYSTEM vérifie les données saisies.</li> <li>3. SYSTEM modifie le produit</li> <li>4. Fin</li> </ol>
Scénario alternatif :	3-a. SYSTEM affiche un message d'erreur
Post condition :	Produit modifié.

## 2.2 Diagramme de séquence : flow of events

1. vendeur sélectionne le produit à modifier
2. vendeur entre les nouvelles informations
3. <b>SYSTEM</b> vérifier les données
3.1 <b>if</b> données valid
3.1.2 <b>SYSTEM</b> modifier le produit
5 <b>else</b>
5.1 <b>SYSTEM</b> affiche message d'erreur
<b>end if</b>



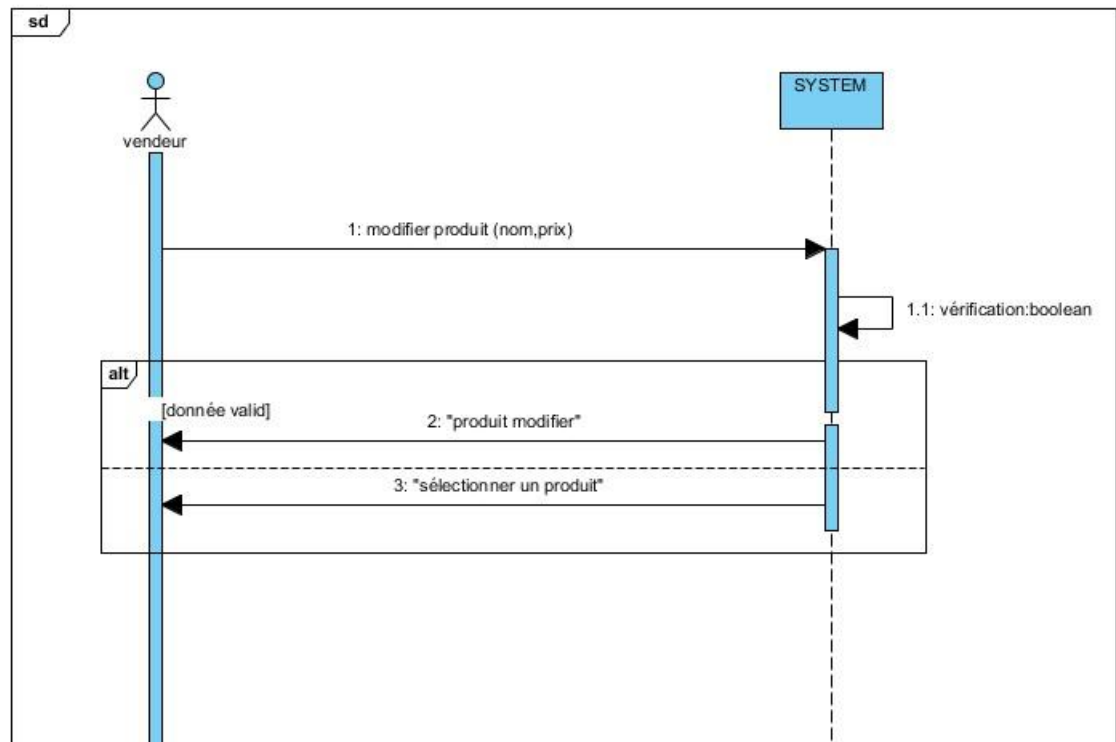
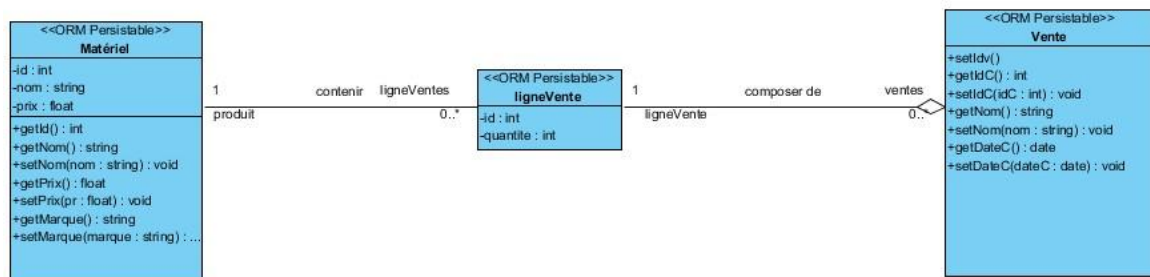


Figure 6 : SD : modifier produit

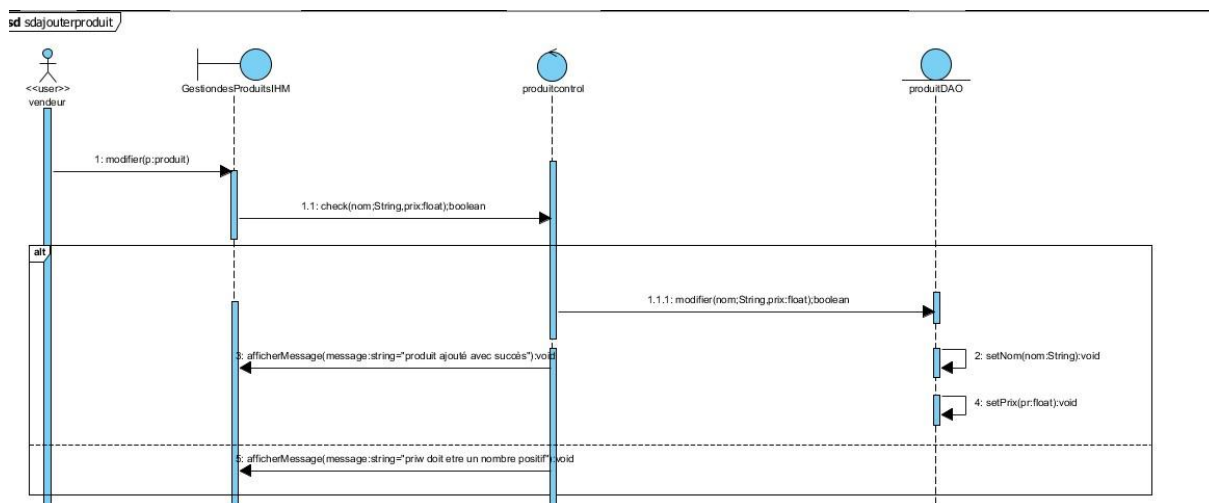
## 2.3 Trouver le concept du domaine



## 2.4 Diagramme des classes participantes



## 2.5 Diagramme d'interaction



## 3. Itération #3 : rechercher un produit

### 3.1 Description Textuelle :

Nom	Rechercher un matériel
Objectif	Permettre à un vendeur de rechercher un produit
ID	1
Acteurs principaux	Vendeur
Date	05/03/2025

Responsable	zakaria
Statut	Proposé
Version	V1.0
Précondition	Vendeur saisit Nom du prout
Scénario nominal	<ol style="list-style-type: none"> <li>1. Admin saisit les informations du matériel</li> <li>2. SYSTEM vérifie les données saisies.</li> <li>3. SYSTEM ajout la demande</li> <li>4. Fin</li> </ol>
Scénario alternatif :	---
Post condition :	Résultat fournie

### 3.2 Diagramme de séquence : flow of events

1. vendeur entre nom du produit
2. SYSTEM vérifier les données
4.1 if produit trouvé
4.1.2 SYSTEM affiche les informations du produit
5 else
5.1 SYSTEM affiche produit introuvable
end if

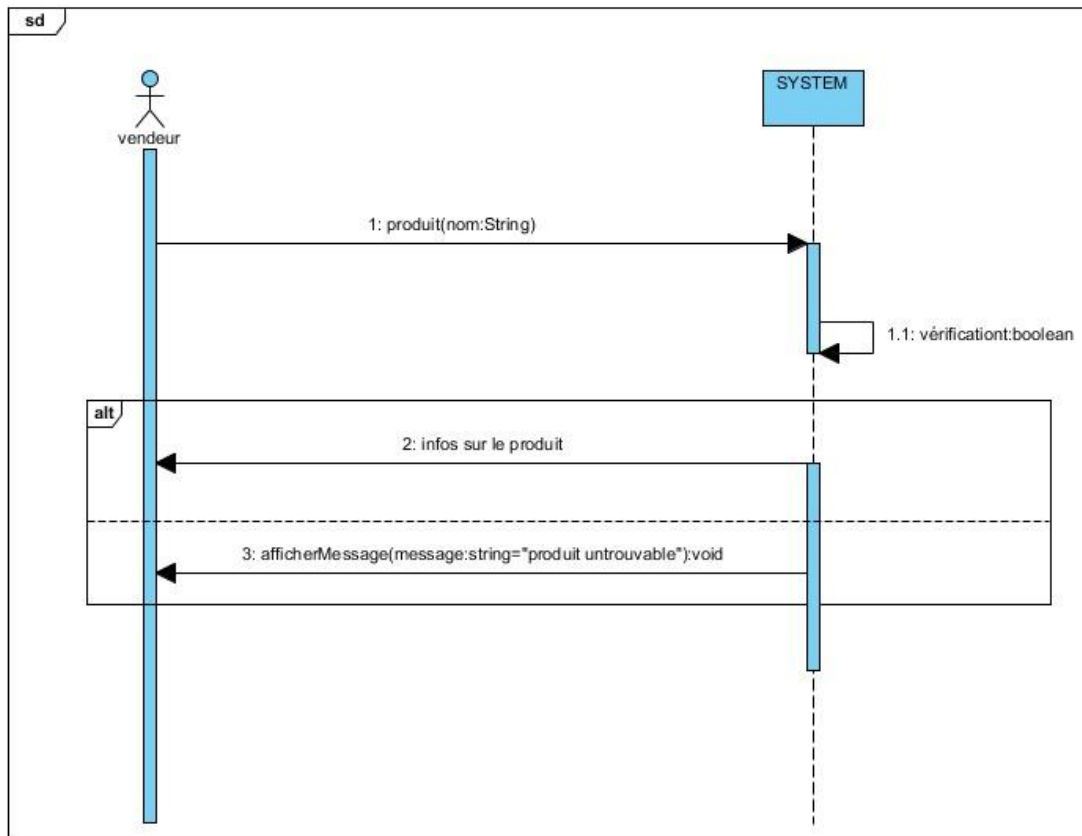
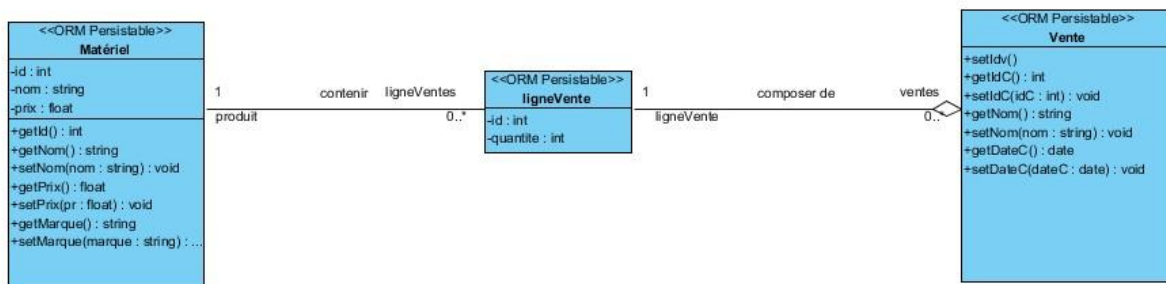
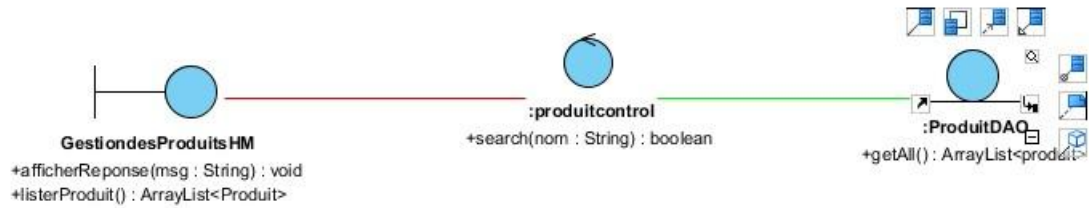


Figure 10 : SD : rechercher produit

### 3.3 Concept de Domaine

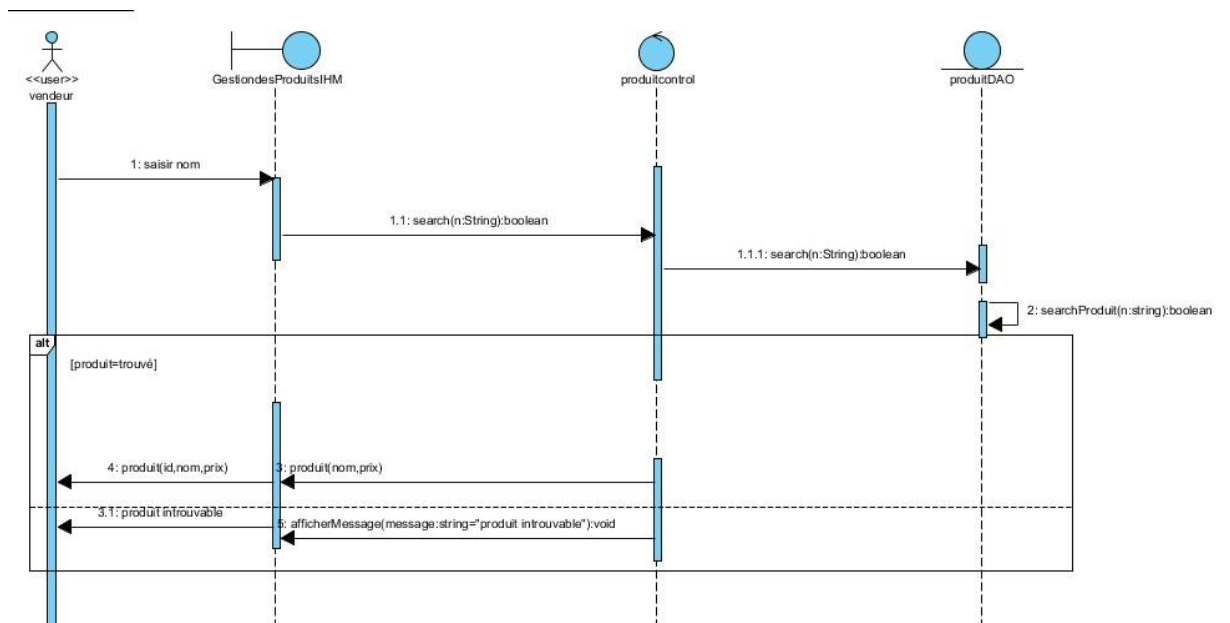


### 3.4 Diagrammes des Classes participantes :



### 3.5 Diagramme d'interaction

set



## 4. Itération #4 : supprimer produit à vendre

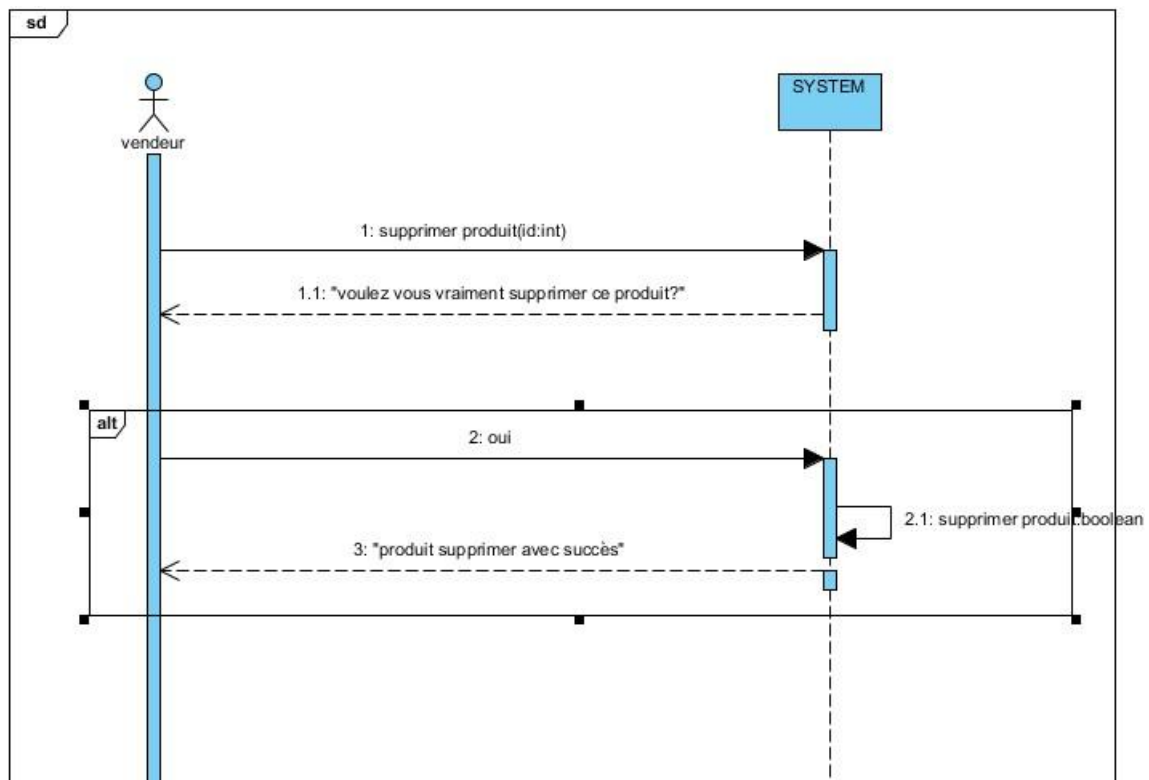
### 4.1 Description Textuelle

Nom	Supprimer produit à vendre
Objectif	Permettre à un vendeur de supprimer un produit à vendre
ID	1
Acteurs principaux	Vendeur

Date	01/03/2025
Responsable	zakaria
Statut	Proposé
Version	V1.0
Précondition	Produit sélectionné
Scénario nominal	<ol style="list-style-type: none"> <li>1. Vendeur sélectionne le produit</li> <li>2. SYSTEM affiche alerte</li> <li>3. Vendeur confirme son choix</li> <li>4. SYSTEM affiche message d'opération</li> <li>5. Fin</li> </ol>
Scénario alternatif :	3-afficher message d'erreur
Post condition :	Réponse fournie

## 4.2 Diagramme de séquence : Flow of events

1. Vendeur sélectionne un produit à supprimer
2. Vendeur clique supprimer
4.SYSTEM affiche alerte de confirmation de choix
4.1 if choix== oui
4.1.2 SYSTEM supprime produit et affiche message de confirmation
end if



## 4.3 Maquette

Gestion d'Épicerie

Vente Gestion des Produits

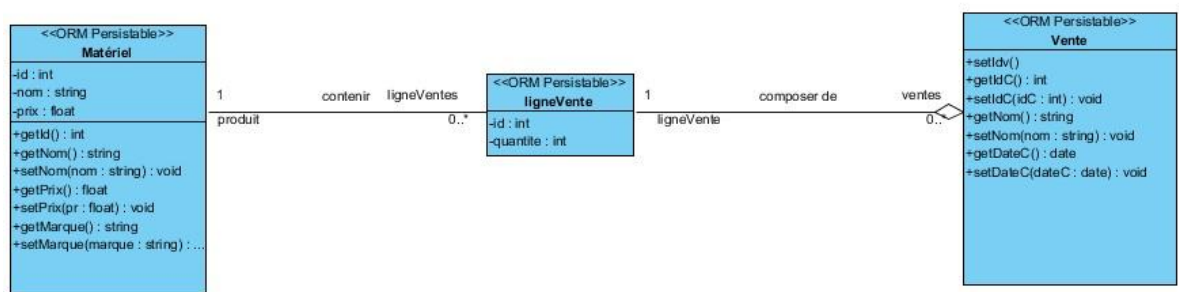
Chercher un produit:

ID	Produit	Prix (DH)
<div>---&gt;</div> <div>&lt;---</div> <div>Effacer</div> <div>Payer</div>		

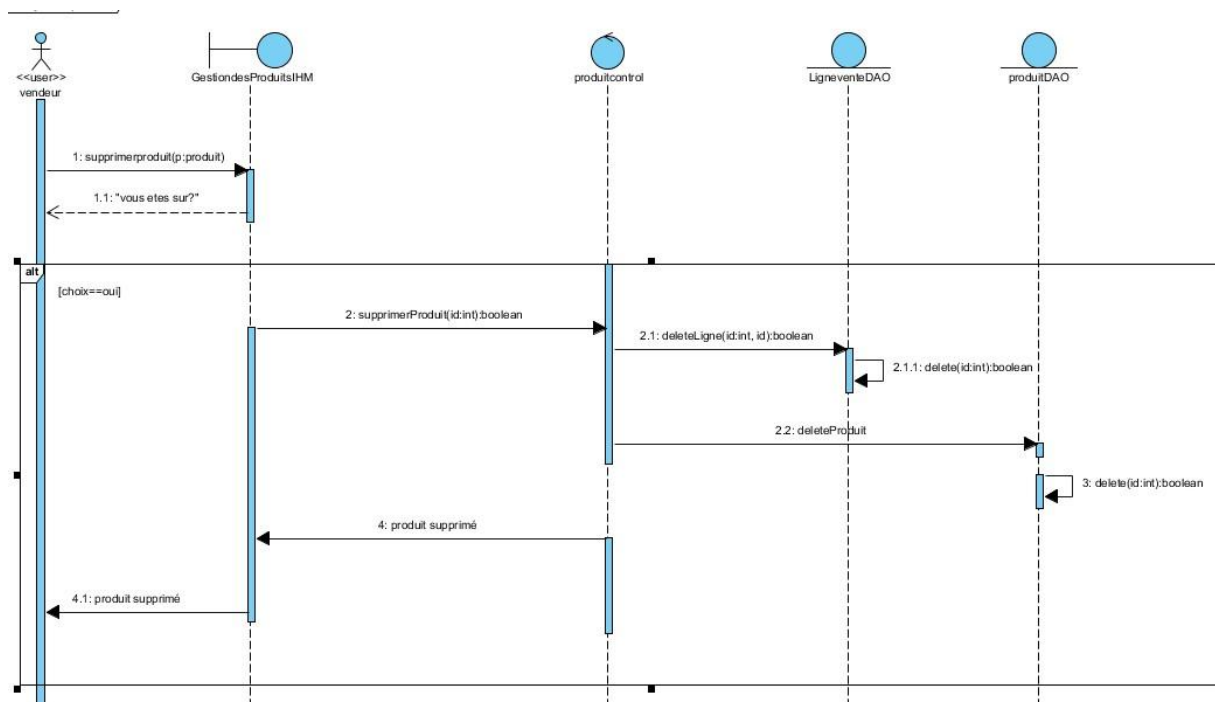
ID	Produit	Prix (DH)	Qté	Total (DH)

Total: 0.00 DH

## 4.4 Concept du domaine



## 4.5 Diagramme d'interaction



## V. Tests et Validation

### 1. Introduction

Les tests constituent une étape cruciale dans tout projet logiciel, car ils permettent de garantir la fiabilité, la robustesse et la stabilité de l'application. Dans le cadre de notre projet de gestion



d'épicerie, nous avons mis en place des **tests unitaires** pour valider les différentes fonctionnalités offertes par l'application.

## 2. Outils utilisés

Pour effectuer les tests, nous avons utilisé les outils suivants :

- **JUnit 5** : Framework de test pour Java, intégré dans notre projet Maven.
- **Mockito** : Framework permettant de créer des objets simulés (mocks) pour isoler les composants testés.
- **JaCoCo (Java Code Coverage)** : Outil de génération de rapports de couverture de code, permettant de mesurer quelles portions du code sont effectivement exécutées lors des tests.

## 3. Configuration Maven

Le fichier pom.xml contient la configuration nécessaire pour utiliser JaCoCo et les autres dépendances de test.

```
<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <version>0.8.8</version>
  <executions>
    <execution>
      <id>prepare-agent</id>
      <goals>
        <goal>prepare-agent</goal>
      </goals>
    </execution>
    <execution>
      <id>report</id>
      <phase>verify</phase>
      <goals>
        <goal>report</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Une fois la configuration faite, les tests sont lancés via :

- mvn clean test
- mvn jacoco:report

Un rapport HTML est généré dans le dossier target/site/jacoco

#### 4. Classes de tests

Voici une présentation des classes de test organisée en tableaux séparés avec des descriptions détaillées pour chaque classe :

##### 4.1 Classe de test : DBConnectionTest.java

Cette classe est responsable de tester la gestion des connexions à la base de données.

Elle utilise Mockito pour simuler les connexions et vérifier leur comportement.

Méthode de test	Description
testGetConnectionWithCustomConnection()	Vérifie que la méthode getConnection() renvoie correctement une connexion personnalisée qui a été préalablement définie.
testCloseConnection()	Teste la fermeture d'une connexion active, en s'assurant que la méthode close() est bien appelée.
testCloseConnectionAlreadyClosed()	Vérifie que la méthode de fermeture gère correctement le cas où la connexion est déjà fermée (ne tente pas de la fermer à nouveau).
testCloseConnectionNull()	S'assure que la méthode closeConnection() gère correctement le cas d'une connexion null sans lever d'exception.

#### 4.2 Classe de test : LigneVenteDAOTest.java

Cette classe teste les fonctionnalités d'accès aux données pour les lignes de vente. Elle utilise une base de données en mémoire H2 pour simuler les interactions avec la base de données.

Méthode de test	Description
testAjouterLigneVente()	Vérifie l'ajout d'une ligne de vente dans la base de données et s'assure que l'ID est correctement généré.
testTrouverParVente()	Teste la récupération des lignes de vente associées à une vente spécifique et vérifie que les données récupérées sont correctes.
testSupprimerLigneVente()	Vérifie la suppression d'une ligne de vente spécifique et confirme qu'elle n'existe plus dans la base de données.
testSupprimerParVente()	Teste la suppression de toutes les lignes de vente associées à une vente spécifique.

#### 4.3 Classe de test : LigneVenteTest.java

Cette classe teste le modèle de données LigneVente, en vérifiant ses constructeurs, ses getters/setters et ses méthodes de calcul.

Méthode de test	Description
testConstructeurSansParametres()	Vérifie que le constructeur par défaut initialise correctement les attributs.
testConstructeurAvecProduitEtQuantite()	Teste le constructeur qui prend un produit et une quantité en paramètres.

testConstructeurAvecIdProduitEtQuantite()	Vérifie le constructeur complet avec ID, produit et quantité.
testSettersEtGetters()	S'assure que tous les accesseurs et mutateurs fonctionnent correctement.
testGetSousTotal()	Vérifie que la méthode de calcul du sous-total renvoie le bon montant (prix × quantité).
testToString()	Teste que la représentation sous forme de chaîne de caractères est formatée correctement.

#### 4.4 Classe de test : ProduitDAOTest.java

Cette classe teste les opérations CRUD sur les produits dans la base de données, en utilisant des mocks pour simuler les connexions et les requêtes.

Méthode de test	Description
testAjouterProduit()	Vérifie l'ajout d'un produit dans la base de données et s'assure que l'ID généré est correctement assigné.
testAjouterProduitCatchSQLException()	Teste la gestion des exceptions SQL lors de l'ajout d'un produit.
testTrouverParId()	Vérifie la récupération d'un produit par son ID et contrôle l'exactitude des données.
testTrouverParIdCatchSQLException()	Teste la gestion des exceptions SQL lors de la recherche d'un produit.

testTrouverTous()	Vérifie que tous les produits sont correctement récupérés et triés par nom.
testTrouverTousCatchSQLException()	Teste la gestion des exceptions SQL lors de la récupération de tous les produits.
testRechercherParNom()	Vérifie la recherche de produits par nom et contrôle les résultats.
testRechercherParNomCatchSQLException()	Teste la gestion des exceptions SQL lors de la recherche par nom.
testModifier()	Vérifie la mise à jour des informations d'un produit existant.
testModifierCatchSQLException()	Teste la gestion des exceptions SQL lors de la modification d'un produit.
testSupprimer()	Vérifie la suppression d'un produit par son ID.
testSupprimerCatchSQLException()	Teste la gestion des exceptions SQL lors de la suppression d'un produit.

#### 4.5 Classe de test : ProduitTest.java

Cette classe teste le modèle de données Produit, en vérifiant ses constructeurs, ses getters/setters et sa méthode toString.

Méthode de test	Description
-----------------	-------------

testConstructeurSansParametres()	Vérifie que le constructeur par défaut initialise correctement les attributs.
testConstructeurAvecIdNomPrix()	Teste le constructeur complet avec ID, nom et prix.
testConstructeurAvecNomPrix()	Vérifie le constructeur qui prend un nom et un prix en paramètres.
testSettersEtGetters()	S'assure que tous les accesseurs et mutateurs fonctionnent correctement.
testToString()	Teste que la représentation sous forme de chaîne de caractères est formatée correctement.

#### 4.6 Classe de test : VenteDAOTest.java

Cette classe teste les opérations d'accès aux données pour les ventes, en utilisant des mocks pour simuler les interactions avec la base de données.

Méthode de test	Description
testAjouterVente()	Vérifie l'ajout d'une vente dans la base de données et s'assure que l'ID généré est correctement assigné.
testTrouverParId()	Teste la récupération d'une vente par son ID et contrôle l'exactitude des données.
testTrouverTous()	Vérifie que toutes les ventes sont correctement récupérées et triées par date.

testSupprimerVente()	Teste la suppression d'une vente par son ID.
testAjouterVenteCatchSQLException()	Vérifie la gestion des exceptions SQL lors de l'ajout d'une vente.
testTrouverParIdCatchSQLException()	Teste la gestion des exceptions SQL lors de la recherche d'une vente.
testTrouverTousCatchSQLException()	Vérifie la gestion des exceptions SQL lors de la récupération de toutes les ventes.
testSupprimerVenteCatchSQLException()	Teste la gestion des exceptions SQL lors de la suppression d'une vente.

#### 4.7 Classe de test : VenteTest.java

Cette classe teste le modèle de données Vente, en vérifiant ses constructeurs, ses getters/setters et ses méthodes de gestion des lignes de vente.

Méthode de test	Description
testConstructeurSansParametres()	Vérifie que le constructeur par défaut initialise correctement les attributs.
testConstructeurAvecParametres()	Teste le constructeur qui prend ID, date et montant total en paramètres.
testSettersEtGetters()	S'assure que tous les accesseurs et mutateurs fonctionnent correctement.

testAjouterLigneVente()	Vérifie l'ajout d'une ligne de vente et son impact sur le montant total.
testRetirerLigneVente()	Teste la suppression d'une ligne de vente et la mise à jour du montant total.
testCalculerTotal()	Vérifie que le calcul du montant total de la vente est correct en additionnant les sous-totaux des lignes.

## 5. Rapport de couverture de code (JaCoCo)





L'exécution des tests a permis de produire un rapport de couverture, indiquant le pourcentage du code testé. Voici un aperçu des résultats obtenus :

Classe	Couverture des lignes
Vente.java	100%
LigneVente.java	100%
Produit.java	100%
ProduitDAO.java	98%
VenteDAO.java	98%
LigneVenteDAO.java	98%
DBConnection.java	47%





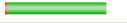
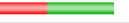


Gestion-epicerie > com.epicerie.model

## com.epicerie.model

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Vente		100 %		100 %	0	14	0	34	0	13	0	1
LigneVente		100 %		n/a	0	11	0	23	0	11	0	1
Produit		100 %		n/a	0	10	0	21	0	10	0	1
Total	0 of 221	100 %	0 of 2	100 %	0	35	0	78	0	34	0	3

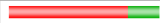





Gestion-epicerie > com.epicerie.dao

## com.epicerie.dao

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ProduitDAO		98 %		59 %	9	18	2	83	0	7	0	1
VenteDAO		98 %		55 %	8	14	2	55	0	5	0	1
LigneVenteDAO		98 %		56 %	7	13	1	55	0	5	0	1
Total	10 of 655	98 %	24 of 56	57 %	24	45	5	193	0	17	0	3

Gestion-epicerie > com.epicerie.util > DBConnection

## DBConnection

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
getConnection()		21 %		50 %	1	2	4	6	0	1
closeConnection(Connection)		60 %		100 %	0	3	2	7	0	1
setConnection(Connection)		100 %		n/a	0	1	0	2	0	1
static {...}		100 %		n/a	0	1	0	1	0	1
Total	21 of 40	47 %	1 of 6	83 %	1	7	6	16	0	4

## VI. Conclusion

En conclusion, ce projet d'application de gestion d'épicerie a permis de développer une solution simple et efficace pour répondre aux besoins des petits commerces. Grâce à une conception rigoureuse, des itérations bien planifiées et des tests approfondis, l'application offre des fonctionnalités essentielles comme la gestion des produits et des ventes, tout en garantissant une interface intuitive et des performances optimales. Ce travail démontre l'importance de la méthodologie et des bonnes pratiques dans le développement logiciel pour aboutir à un outil fiable et adapté aux utilisateurs finaux.