



EMSI – 4IIR Groupe2

Année universitaire : 2025–2026

ProductShopping : Application de Shopping

Java Avancé / Programmation Orientée Objet

Réalisé par :

Kawtar GOUY

Encadré par :

Abderrahim Larhlimi

10 janvier 2026

Résumé

Ce rapport présente ProductShopping, une application de shopping développée en Java. Le projet couvre l'analyse des besoins, la conception UML, la conception de la base de données, l'architecture logicielle, l'implémentation, l'interface utilisateur et les tests. L'objectif est de mettre en pratique Java avancé, une approche structurée (DAO, services) et une persistance fiable avec un SGBD relationnel.

Table des matières

Résumé	1
1 Introduction générale	6
1.1 Contexte du projet	6
1.2 Problématique	6
1.3 Objectifs	6
2 Partie I : Analyse et conception	7
2.1 Spécification des besoins	7
2.1.1 Besoins fonctionnels	7
2.1.2 Besoins non fonctionnels	7
2.2 Conception UML	8
2.2.1 Diagramme de classes	8
2.3 Conception de la base de données	8
2.3.1 Modèle logique de données (MLD)	8
2.3.2 Dictionnaire de données	9
3 Partie II : Environnement technique	10
3.1 Technologies et outils	10
3.2 Extrait Maven (pom.xml)	10
4 Partie III : Architecture et implémentation	12
4.1 Architecture logicielle	12
4.2 Design patterns utilisés	12
4.2.1 Singleton	12
4.2.2 DAO	13
4.3 Extraits de code clés	13
4.3.1 Connexion et gestion des ressources	13
4.3.2 Création de commande avec transaction	14
4.3.3 Gestion des erreurs	14

5	Partie IV : Interface utilisateur et tests	16
5.1	Présentation des interfaces	16
5.2	Scénarios de test	18
5.2.1	Tests nominaux	18
5.2.2	Tests d'erreurs	19
6	Conclusion et perspectives	20
6.1	Bilan technique	20
6.2	Compétences acquises	20
6.3	Difficultés rencontrées	20
6.4	Perspectives	20
7	Webographie / Bibliographie	21
A	Annexes	23
A.1	Structure du projet	23
A.2	Script SQL (optionnel)	23

Table des figures

2.1	Diagramme de classes de ProductShopping	8
5.1	Écran Login	16
5.2	Catalogue produits	17
5.3	Panier	18

Liste des tableaux

2.1	Dictionnaire de données (exemple)	9
5.1	Scénarios nominaux	18
5.2	Scénarios d’erreur	19

Chapitre 1

Introduction générale

1.1 Contexte du projet

Dans plusieurs contextes, le suivi des produits, des commandes et des utilisateurs se fait encore de façon non centralisée. Cela entraîne des erreurs, des doublons et une perte de temps.

1.2 Problématique

L'obligation de centraliser la gestion du catalogue (produits, catégories), la gestion des utilisateurs et la gestion des commandes pour garantir la cohérence des données, la traçabilité et un accès rapide à l'information.

1.3 Objectifs

- Gérer le catalogue produits (ajout, modification, suppression, recherche).
- Gérer les utilisateurs et l'authentification.
- Gérer le panier et la commande (création, validation, historique).
- Assurer la persistance des données dans une base relationnelle.
- Structurer le code avec une architecture claire et réutilisable.

Chapitre 2

Partie I : Analyse et conception

2.1 Spécification des besoins

2.1.1 Besoins fonctionnels

- Le système doit permettre à l'administrateur de gérer les produits.
- Le système doit permettre à l'administrateur de gérer les catégories.
- Le système doit permettre à un client de créer un compte et se connecter.
- Le système doit permettre au client de consulter le catalogue et rechercher un produit.
- Le système doit permettre au client d'ajouter des produits au panier.
- Le système doit permettre au client de valider une commande.
- Le système doit afficher l'historique des commandes du client.

2.1.2 Besoins non fonctionnels

- Sécurité : mots de passe stockés de façon sécurisée (hachage + sel).
- Fiabilité : transactions lors de la création de commande.
- Performance : opérations courantes rapides (index sur clés étrangères).
- Maintenabilité : architecture en couches, code propre, exceptions gérées.
- Portabilité : exécution sur Windows et Linux avec JDK récent.

2.2 Conception UML

2.2.1 Diagramme de classes

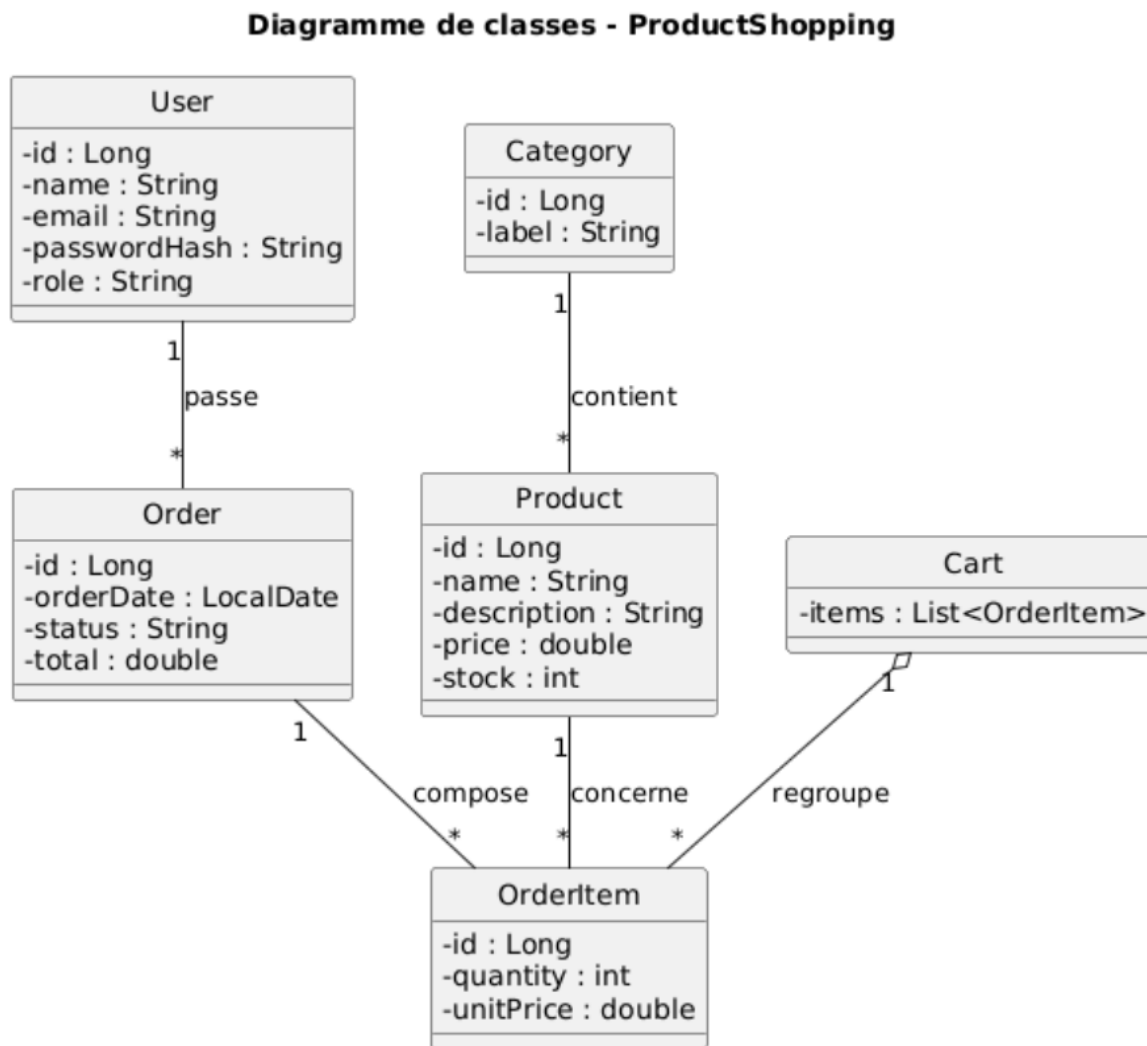


FIGURE 2.1 – Diagramme de classes de ProductShopping

- User (id, name, email, passwordHash, role)
- Product (id, name, price, stock, categoryId)
- Category (id, label)
- Order (id, userId, date, status, total)

2.3 Conception de la base de données

2.3.1 Modèle logique de données (MLD)

schéma relationnel :

- USERS(id, name, email, password_hash, role)

- CATEGORIES(id, label)
- PRODUCTS(id, name, description, price, stock, **category_id** FK)
- ORDERS(id, **user_id** FK, order_date, status, total)
- ORDER_ITEMS(id, **order_id** FK, **product_id** FK, quantity, unit_price)

2.3.2 Dictionnaire de données

TABLE 2.1 – Dictionnaire de données (exemple)

Table	Champ	Type	Contraintes / Description
USERS	id	BIGINT	PK, auto-incrément
USERS	email	VARCHAR(150)	UNIQUE, NOT NULL
USERS	password_hash	VARCHAR(255)	NOT NULL
PRODUCTS	price	DECIMAL(10,2)	NOT NULL, ≥ 0
PRODUCTS	stock	INT	NOT NULL, ≥ 0
ORDERS	total	DECIMAL(10,2)	NOT NULL

Chapitre 3

Partie II : Environnement technique

3.1 Technologies et outils

- Langage : Java (JDK 25).
- IDE : IntelliJ IDEA
- Build : Maven (gestion dépendances + packaging).
- SGBD : MySQL 8.0
- ORM : Hibernate et JDBC
- Modélisation : Draw.io
- Gestion de version : Git + GitHub.

3.2 Extrait Maven (pom.xml)

```
1 <dependencies>
2   <!-- Driver MySQL -->
3   <dependency>
4     <groupId>com.mysql</groupId>
5     <artifactId>mysql-connector-j</artifactId>
6     <version>8.4.0</version>
7   </dependency>
8
9   <!-- Hibernate (si utilis ) -->
10  <dependency>
11    <groupId>org.hibernate.orm</groupId>
12    <artifactId>hibernate-core</artifactId>
13    <version>6.5.2.Final</version>
14  </dependency>
15 </dependencies>
```

Listing 3.1 – Extrait pom.xml (exemple)

Chapitre 4

Partie III : Architecture et implémentation

4.1 Architecture logicielle

- `ma.emsi.productshopping.model` : entités métier (Product, Order, User).
- `ma.emsi.productshopping.dao` : accès aux données (ProductDAO, OrderDAO).
- `ma.emsi.productshopping.service` : logique métier (CartService, OrderService).
- `ma.emsi.productshopping.ui` : interface (JavaFX ou console).
- `ma.emsi.productshopping.util` : utilitaires (HibernateUtil, DBUtil, validators).

4.2 Design patterns utilisés

4.2.1 Singleton

```
1 public final class DbConfig {
2     private static final DbConfig INSTANCE = new DbConfig();
3     private final String url = "jdbc:mysql://localhost:3306/
4         productshopping";
5     private final String user = "root";
6     private final String password = "root";
7
8     private DbConfig() {}
9
10    public static DbConfig getInstance() {
11        return INSTANCE;
12    }
```

```
13     public String getUrl() { return url; }
14     public String getUser() { return user; }
15     public String getPassword() { return password; }
16 }
```

Listing 4.1 – Singleton (exemple simple)

4.2.2 DAO

Le pattern DAO isole l'accès à la base. Il rend le code testable et plus maintenable.

```
1 public class ProductDAO {
2
3     public List<Product> findAll(Connection cn) throws SQLException
4     {
5         String sql = "SELECT id, name, price, stock FROM products";
6         try (PreparedStatement ps = cn.prepareStatement(sql);
7             ResultSet rs = ps.executeQuery()) {
8
9             List<Product> out = new ArrayList<>();
10            while (rs.next()) {
11                Product p = new Product();
12                p.setId(rs.getLong("id"));
13                p.setName(rs.getString("name"));
14                p.setPrice(rs.getBigDecimal("price"));
15                p.setStock(rs.getInt("stock"));
16                out.add(p);
17            }
18            return out;
19        }
20 }
```

Listing 4.2 – DAO (exemple minimal JDBC)

4.3 Extraits de code clés

4.3.1 Connexion et gestion des ressources

```
1 public final class DBUtil {
2
3     private DBUtil() {}
4 }
```

```
5     public static Connection open() throws SQLException {
6         DbConfig cfg = DbConfig.getInstance();
7         return DriverManager.getConnection(cfg.getUrl(), cfg.
            getUser(), cfg.getPassword());
8     }
9 }
```

Listing 4.3 – Connexion JDBC (exemple)

4.3.2 Création de commande avec transaction

Objectif : aucune commande partielle en cas d'erreur.

```
1 public long createOrder(long userId, List<CartLine> cart) throws
   SQLException {
2     try (Connection cn = DBUtil.open()) {
3         cn.setAutoCommit(false);
4
5         try {
6             long orderId = insertOrder(cn, userId, cart);
7             for (CartLine line : cart) {
8                 insertOrderItem(cn, orderId, line);
9                 decrementStock(cn, line.getProductId(), line.
                    getQuantity());
10            }
11
12            cn.commit();
13            return orderId;
14        } catch (Exception ex) {
15            cn.rollback();
16            throw ex;
17        } finally {
18            cn.setAutoCommit(true);
19        }
20    }
21 }
```

Listing 4.4 – Transaction : créer une commande (pseudo-code Java)

4.3.3 Gestion des erreurs

— Exceptions techniques : SQLException, erreurs de connexion.

- Exceptions métier : stock insuffisant, panier vide, utilisateur non trouvé.
- Messages clairs côté UI : alertes, logs, validations.

Chapitre 5

Partie IV : Interface utilisateur et tests

5.1 Présentation des interfaces

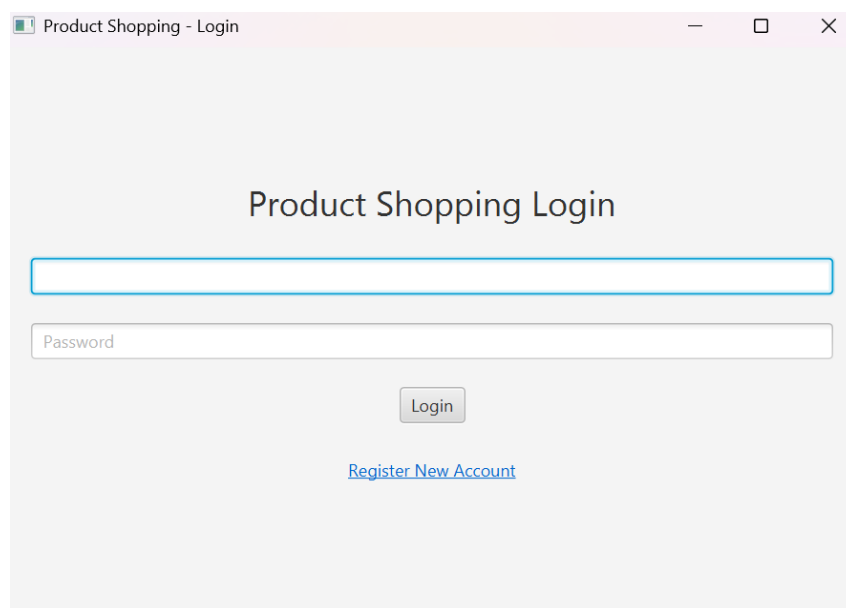


FIGURE 5.1 – Écran Login

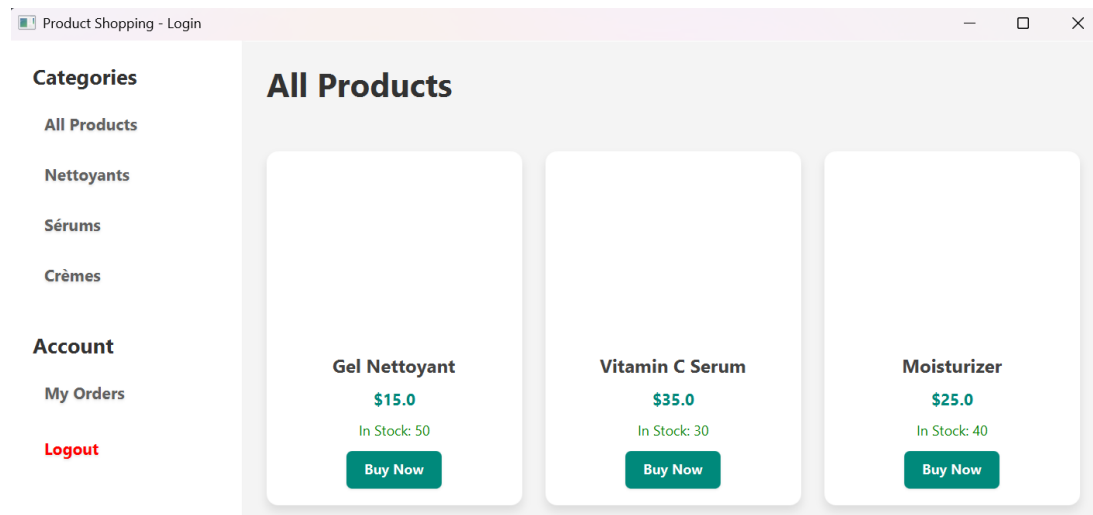


FIGURE 5.2 – Catalogue produits

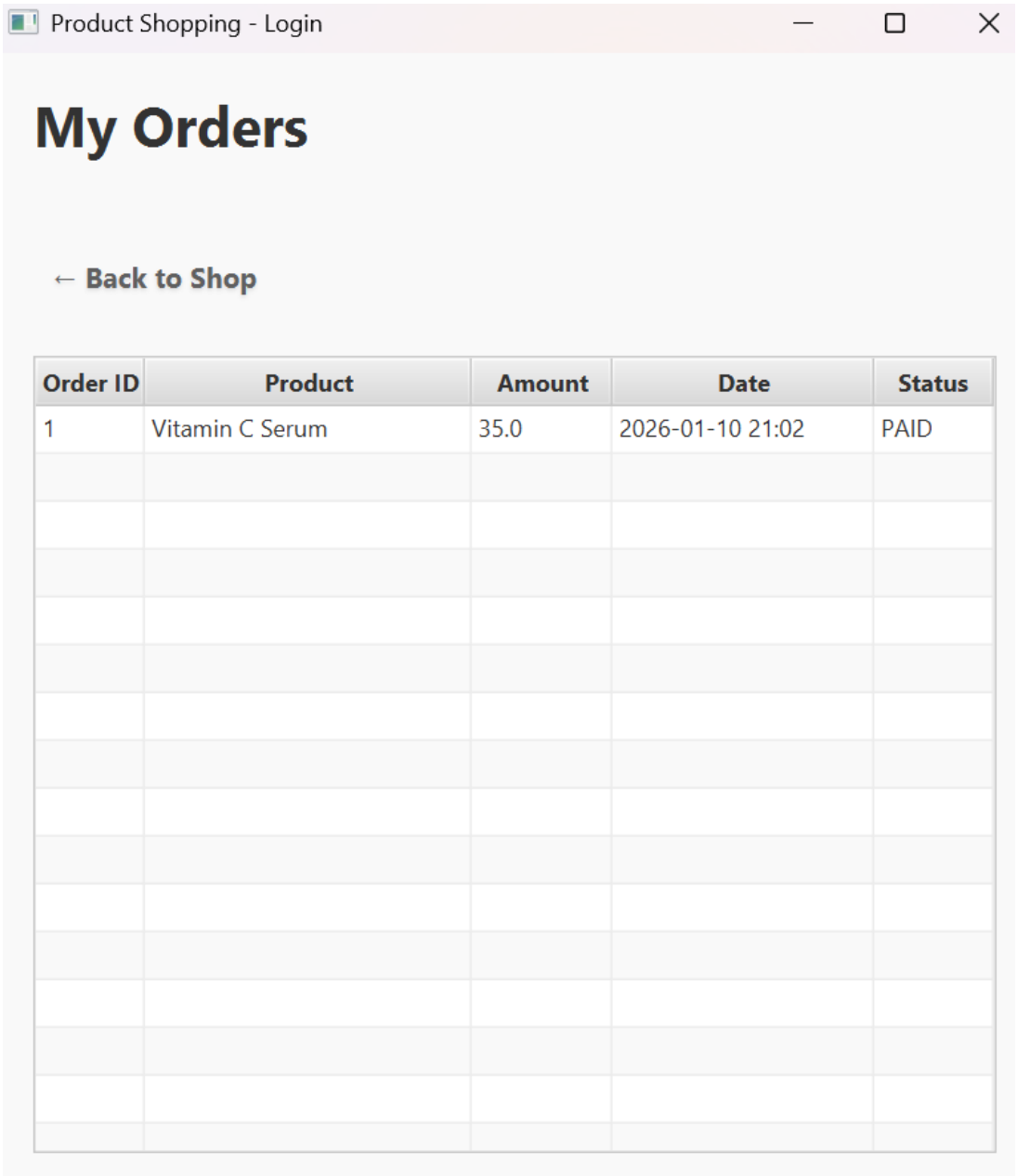


FIGURE 5.3 – Panier

5.2 Scénarios de test

5.2.1 Tests nominaux

TABLE 5.1 – Scénarios nominaux

ID	Description	Résultat attendu
TN-01	Ajout d'un produit (Admin)	Produit ajouté et affiché

ID	Description	Résultat attendu
TN-02	Connexion client valide	Accès au catalogue
TN-03	Ajouter au panier puis valider	Commande créée + stock mis à jour

5.2.2 Tests d'erreurs

TABLE 5.2 – Scénarios d'erreur

ID	Description	Résultat attendu
TE-01	Mot de passe incorrect	Message d'erreur clair
TE-02	Stock insuffisant à la validation	Commande refusée, aucune écriture partielle
TE-03	Champs vides dans formulaire produit	Validation côté UI + blocage sauvegarde

Chapitre 6

Conclusion et perspectives

6.1 Bilan technique

Le projet respecte les fonctionnalités principales : gestion du catalogue, authentification, panier et commandes. L'utilisation d'une architecture en couches rend le code plus lisible et testable.

6.2 Compétences acquises

- Structuration d'un projet Java (packages, responsabilités).
- Persistance avec base relationnelle et transactions.
- Patterns : DAO, Singleton, services métier.
- Gestion des exceptions et validations.

6.3 Difficultés rencontrées

- Configuration des dépendances Maven.
- Problèmes de mapping (si Hibernate).
- Gestion correcte des transactions.

6.4 Perspectives

- Ajouter un système de rôles complet (Admin, Client, Manager).
- Ajouter un paiement simulé (statuts : pending, paid, canceled).
- Ajouter des rapports (ventes par période, top produits).
- Migrer vers une version Web (Spring Boot + REST).

Chapitre 7

Webographie / Bibliographie

Bibliographie

- [1] Documentation Java SE, Oracle.
- [2] Documentation MySQL.
- [3] Documentation Hibernate ORM.
- [4] Documentation Apache Maven.

Annexe A

Annexes

A.1 Structure du projet

```
1 ProductShopping/  
2   src/main/java/ma/emsi/productshopping/  
3     model/  
4     dao/  
5     service/  
6     ui/  
7     util/  
8   src/main/resources/  
9   images/  
10  pom.xml
```

Listing A.1 – Arborescence (exemple)

A.2 Script SQL (optionnel)

```
1 CREATE TABLE categories (  
2   id BIGINT PRIMARY KEY AUTO_INCREMENT,  
3   label VARCHAR(100) NOT NULL  
4 );  
5  
6 CREATE TABLE products (  
7   id BIGINT PRIMARY KEY AUTO_INCREMENT,  
8   name VARCHAR(150) NOT NULL,  
9   description TEXT,  
10  price DECIMAL(10,2) NOT NULL,  
11  stock INT NOT NULL,
```

```
12  category_id BIGINT,  
13  CONSTRAINT fk_prod_cat FOREIGN KEY (category_id) REFERENCES  
    categories(id)  
14 );
```

Listing A.2 – DDL (exemple)