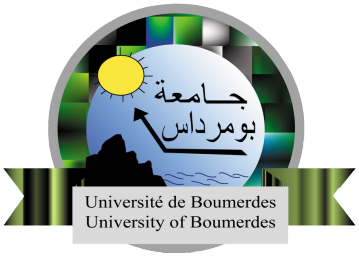
République Algérienne Démocratique et PopulaireMinistère de l'Enseignement Supérieur et de la Recherche ScientifiqueUniversité M'hamed Bougara - Boumerdès

Faculté des SciencesDépartement d'Informatique



Système de Gestion de Supermarché

avec Commandes en Ligne

Mémoire de fin d'études en vue de l'obtention du Diplôme de Licence Académique

Filière :	Informatique
Domaine :	Informatique
Spécialité :	Systèmes Informatiques
Encadrant :	IMINE Belkacem
Examinateur :	Ishak Boushaki Saida
Équipe :	
Bensalah Dhiaeddine	Debchi Nour El Houda
Teffah Adem Lahlou	Talah Melissa
Année :	2025–2026

Table des matières

1 Introduction **1**

1.1 Contexte du projet 1
1.2 Problématique 1
1.3 Objectifs du Projet 2
1.4 Justification du choix du projet 2
1.5 Acteurs interagissant avec la plateforme 2
1.6 Approche générale 3
1.7 Conclusion du chapitre 1 3

2 Modélisation UML **4**

2.1 Langages de Modélisation 4
2.1.1 Diagramme de Cas d'Utilisation 4
2.1.2 Diagramme de Classe 5
2.1.3 Diagramme d'Activité 5
2.1.4 Diagramme de Séquence 5

3 Conception **6**

3.1 Analyse des Besoins 6
3.1.1 Identification des Acteurs 6
3.1.2 Fonctionnalités Principales 6
3.1.3 Diagramme de Cas d'Utilisation 6
3.2 Conception Statique 11
3.2.1 Description des Classes 11
3.3 Conception Dynamique 12
3.3.1 Diagramme d'Activité 12
3.4 Diagramme de Séquence du Système 16

4 Réalisation et Implémentation **22**

4.1 Introduction 22
4.2 Environnement de développement 22
4.2.1 Architecture Technique de l'Application Mobile 23
4.2.2 Structure du projet 23
4.2.3 Flux des données 24
4.3 Présentation des Interfaces et Fonctionnalités 24

4.3.1 Authentification et Accès 24
4.3.2 Navigation et Catalogue (Interface "Rayons") 25
4.3.3 Gestion de la commande (Interface "Panier") 25
4.3.4 Finalisation (Interface "Checkout") 25
4.3.5 Justifications Techniques et Concepts de Développement 26
4.3.6 Introduction 29
4.3.7 Environnement de développement 29
4.3.8 Langages et Technologies utilisés 29
4.3.9 Architecture et logique de l'application 29
4.4 PROCESSUS DE COMMUNICATION (Architecture Système Mobile, Desktop & Backend) 31
4.4.1 Sécurité et Persistance des Données 32
4.5 Conclusion 33
4.5.1 Présentation des tables 33
4.5.2 Synchronisation des bases de données 34
4.6 Présentation des interfaces : 35
4.6.1 Gestion des produits : 35
4.6.2 Gestion des Commandes : 35
4.6.3 Présentation de l'application mobile 36

Table des figures

3.1 Diagramme de cas d'utilisation — Vue générale 7
3.2 Diagramme de cas d'utilisation — Vue détaillée 8
3.3 Diagramme de classes — Structure statique du système 11
3.4 Diagramme d'activité — Application Mobile (Client) 14
3.5 Diagramme d'activité — Application Desktop (Supermarché) 16
3.6 Diagramme de séquence — Application desktop 17
3.7 Diagramme de séquence — Application desktop 18
3.8 Diagramme de séquence — Application desktop 18
3.9 Diagramme de séquence — Application Mobile (Client) 19
3.10 Diagramme de séquence — Application Mobile (Client) 20

4.1 Architecture Système Mobile, Desktop & Backend 31
4.2 INTERFACE SUPERMARKET ADMIN -PRODUCTS 35
4.3 INTERFACE SUPERMARKET ADMIN -ORDERS 35
4.4 INTERFACE SUPERMARKET APP MOBILE 36

Chapitre 1

Introduction

1.1 Contexte du projet

La gestion des stocks et des commandes pose des problèmes concrets aux supermarchés, surtout depuis que les clients s'attendent à pouvoir commander depuis leur téléphone et à savoir en temps réel si un produit est disponible. C'est dans ce cadre que s'inscrit notre projet : concevoir et développer une solution informatique qui permette à un supermarché de gérer son stock tout en offrant aux clients une application mobile pour passer leurs commandes à distance. Ce travail s'inscrit dans une démarche de modernisation des processus internes, avec pour objectif d'améliorer à la fois la gestion opérationnelle et l'expérience des clients.

1.2 Problématique

Les supermarchés se heurtent régulièrement à plusieurs difficultés :

— La mise à jour manuelle du stock, source fréquente d'erreurs sur les quantités ou la disponibilité réelle des produits.

— L'absence de synchronisation entre ce qui se vend en magasin et ce qui est commandé en ligne.

— Le manque d'un outil centralisé pour suivre les mouvements de stock et les ventes.

— Une communication insuffisante entre les clients et le supermarché concernant le suivi des commandes.

La **problématique principale** est donc : 'Comment concevoir un système intégré capable de gérer le stock d'un supermarché tout en permettant aux clients de passer et suivre leurs commandes en ligne, de façon fiable et automatisée ?'

1.3 Objectifs du Projet

Le projet vise à :

— Développer une application mobile destinée aux clients, pour consulter les produits, gérer leur panier et passer commande.

— Concevoir une application desktop pour les administrateurs du supermarché, couvrant la gestion des produits, du stock et des rapports.

— Mettre en place un backend (API REST) assurant la communication entre les deux applications et la base de données.

— Assurer une mise à jour automatique du stock après chaque commande validée.

— Proposer une interface simple, accessible et sécurisée pour tous les types d'utilisateurs.

1.4 Justification du choix du projet

Ce projet répond à un besoin réel dans la distribution, un secteur où la digitalisation n'est plus optionnelle. Concrètement, il permet de :

— Réduire les erreurs dues à la saisie manuelle dans la gestion du stock.

— Accélérer le traitement des commandes.

— Améliorer l'expérience client grâce à la commande en ligne.

— Proposer une solution qui peut être adaptée à d'autres types de commerces.

1.5 Acteurs interagissant avec la plateforme

Plusieurs catégories d'utilisateurs interagissent avec le système de gestion de stock et de commandes en ligne :

— **Client (Application Mobile)**

— Parcourt le catalogue des produits.

— Ajoute des articles au panier et passe commande.

— Suit l'état de ses commandes et gère son compte utilisateur.

— **Administrateur/Employé (Application Desktop)**

— Gère le catalogue des produits et les catégories.

— Supervise le stock (entrées, sorties, alertes).

— Valide ou annule les commandes.

— Génère des rapports pour le suivi des ventes et du stock.

1.6 Approche générale

Le projet a été conduit selon une démarche structurée, organisée en six étapes :

1. **Analyse des besoins :** Identification des acteurs, des fonctionnalités attendues et des contraintes du système.
2. **Modélisation UML :** Utilisation de diagrammes pour représenter la structure et le comportement du système.
3. **Conception :** Définition de l'architecture logicielle et choix des technologies (Java, Spring Boot, Android, MySQL).
4. **Implémentation :** Développement des applications mobile et desktop, ainsi que du backend (API REST).
5. **Tests et validation :** Vérification du bon fonctionnement des fonctionnalités, de la cohérence des données et de la robustesse du système.
6. **Déploiement et perspectives :** Mise en service de la solution et identification des améliorations possibles.

1.7 Conclusion du chapitre 1

Ce premier chapitre pose les bases du projet en présentant son contexte, la problématique identifiée, les objectifs fixés, ainsi que les acteurs concernés et la démarche adoptée. Il montre pourquoi un supermarché a intérêt à se doter d'un système intégré capable de synchroniser la gestion du stock et les commandes passées en ligne.

Chapitre 2

Modélisation UML

La modélisation est une pratique centrale dans le développement logiciel. Avant d'écrire la moindre ligne de code, construire un modèle permet d'anticiper la structure du système, de détecter des incohérences tôt et de faciliter la communication entre les membres de l'équipe. Un modèle est une représentation abstraite d'un système, conçue pour en faciliter l'étude et la documentation. Plus les systèmes deviennent complexes, plus cette étape est utile, car personne ne peut tenir toute la logique d'un projet en tête sans support visuel.

Un autre avantage concret du modèle est la traçabilité : on peut partir d'un élément du système et remonter jusqu'aux besoins qui l'ont justifié, ou au contraire descendre jusqu'aux choix techniques qu'il a entraînés.

Pour modéliser ce projet, nous avons utilisé UML, qui offre un langage graphique standardisé. Une fois le modèle stabilisé, nous avons pu choisir les technologies les mieux adaptées et passer à la phase de conception.

2.1 Langages de Modélisation

Dans ce projet, UML nous a permis de modéliser le système de gestion du supermarché de façon cohérente et partagée. Il couvre à la fois la structure statique du système et son comportement dynamique, à travers différents types de diagrammes. Contrairement à un processus figé, UML s'adapte aux besoins : on peut l'utiliser pour visualiser, concevoir ou documenter, selon la phase du projet.

2.1.1 Diagramme de Cas d'Utilisation

Les cas d'utilisation délimitent ce que fait le système, pour qui et dans quelles conditions. Ils servent avant tout à exprimer les besoins : en impliquant les utilisateurs dès le début du processus, on s'assure que ce qu'on va construire correspond réellement à ce qu'ils attendent. Chaque cas d'utilisation peut être vu comme une classe de scénarios possibles.

Démarche d'utilisation des cas d'utilisation :

— Identifier les acteurs et les cas.

— Décrire les cas en écrivant des scénarios sous forme textuelle.

— Dessiner les diagrammes de cas.

— S'assurer de la cohérence d'ensemble.

2.1.2 Diagramme de Classe

Le monde réel est peuplé d'objets de toutes sortes. Le diagramme de classes sert à regrouper ces objets selon leurs caractéristiques communes, sous forme d'abstractions appelées classes. Une classe décrit la structure partagée par un ensemble d'objets : ses attributs et ses méthodes. C'est le principal outil pour représenter la structure statique d'un système en termes de données et de relations.

2.1.3 Diagramme d'Activité

Les diagrammes d'activité décrivent des enchaînements d'opérations sous forme de flots de contrôle. Ils s'intègrent naturellement à l'analyse des besoins car ils viennent compléter les cas d'utilisation en montrant comment les fonctionnalités s'articulent dans le temps.

2.1.4 Diagramme de Séquence

Ce type de diagramme représente les échanges entre objets ou acteurs au fil du temps. Il montre comment les messages circulent entre les composants d'un système pour réaliser un scénario donné.

Nous avons retenu les diagrammes les plus utiles pour analyser et concevoir notre système. Les autres types de diagrammes UML s'avèrent davantage pertinents dans des projets industriels ou à forte composante architecturale.

Chapitre 3

Conception

La conception UML consiste à traduire un système logiciel en diagrammes graphiques. Elle fournit un langage visuel commun qui permet de représenter, d'analyser et de structurer les différentes dimensions d'un projet avant de toucher au code.

3.1 Analyse des Besoins

3.1.1 Identification des Acteurs

Un acteur est une entité extérieure qui interagit avec le système — il peut s'agir d'un utilisateur humain, d'une organisation, d'une machine ou d'un autre système. Dans nos diagrammes, il représente un rôle particulier, pas nécessairement une personne précise.

3.1.2 Fonctionnalités Principales

Table 3.1 – Acteurs et responsabilités

Acteur	**Responsabilités**
Client	Passe des commandes via l'application mobile, consulte le catalogue, gère son panier.
Administrateur	Gère les produits, les utilisateurs, les ventes et génère des rapports.
Employee	Enregistre les ventes physiques et met à jour les stocks.

3.1.3 Diagramme de Cas d'Utilisation

Le diagramme de cas d'utilisation représente les différentes fonctions du système. Il illustre ce que les acteurs peuvent faire et comment ils interagissent avec le système pour répondre à leurs besoins.

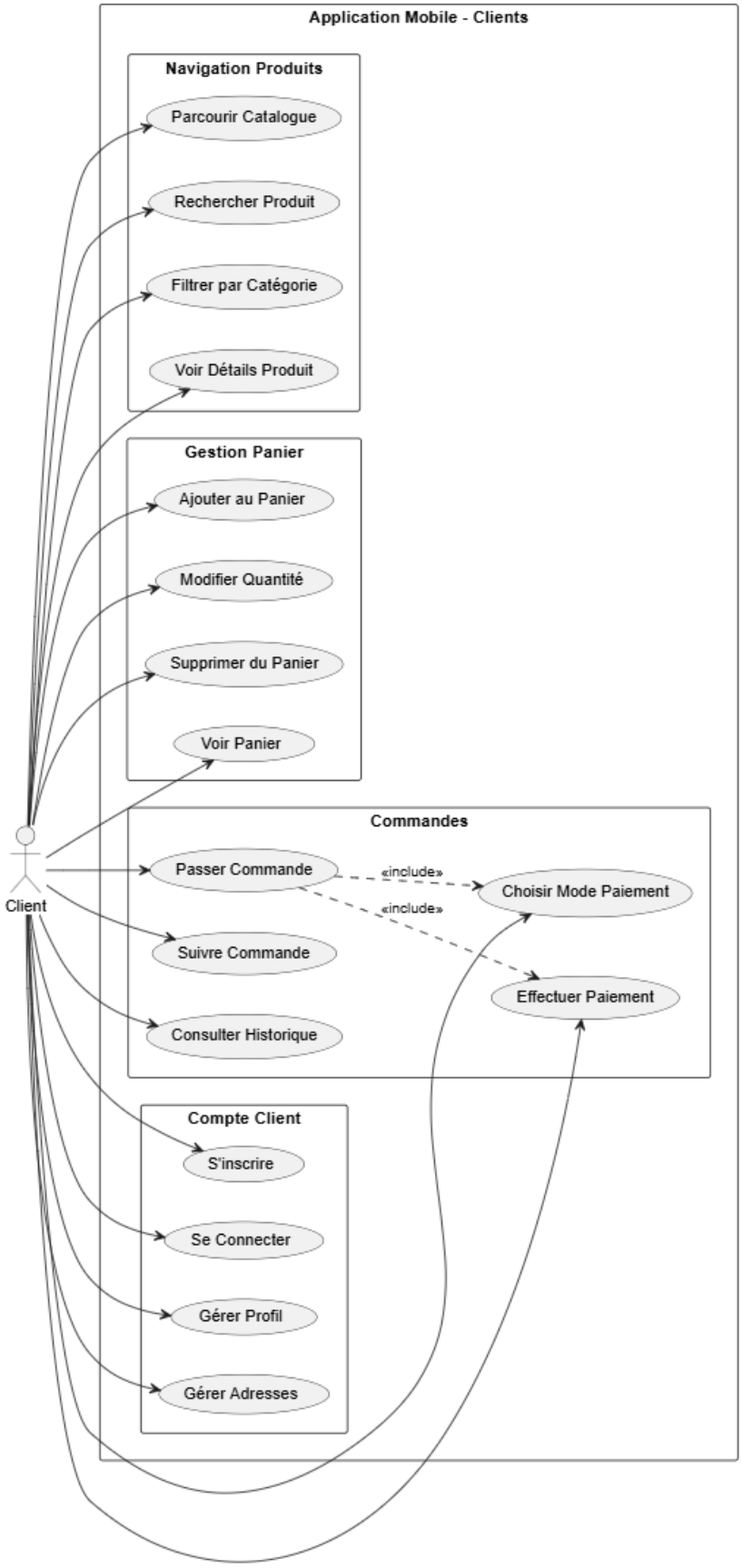


Figure 3.1 – Diagramme de cas d'utilisation — Vue générale

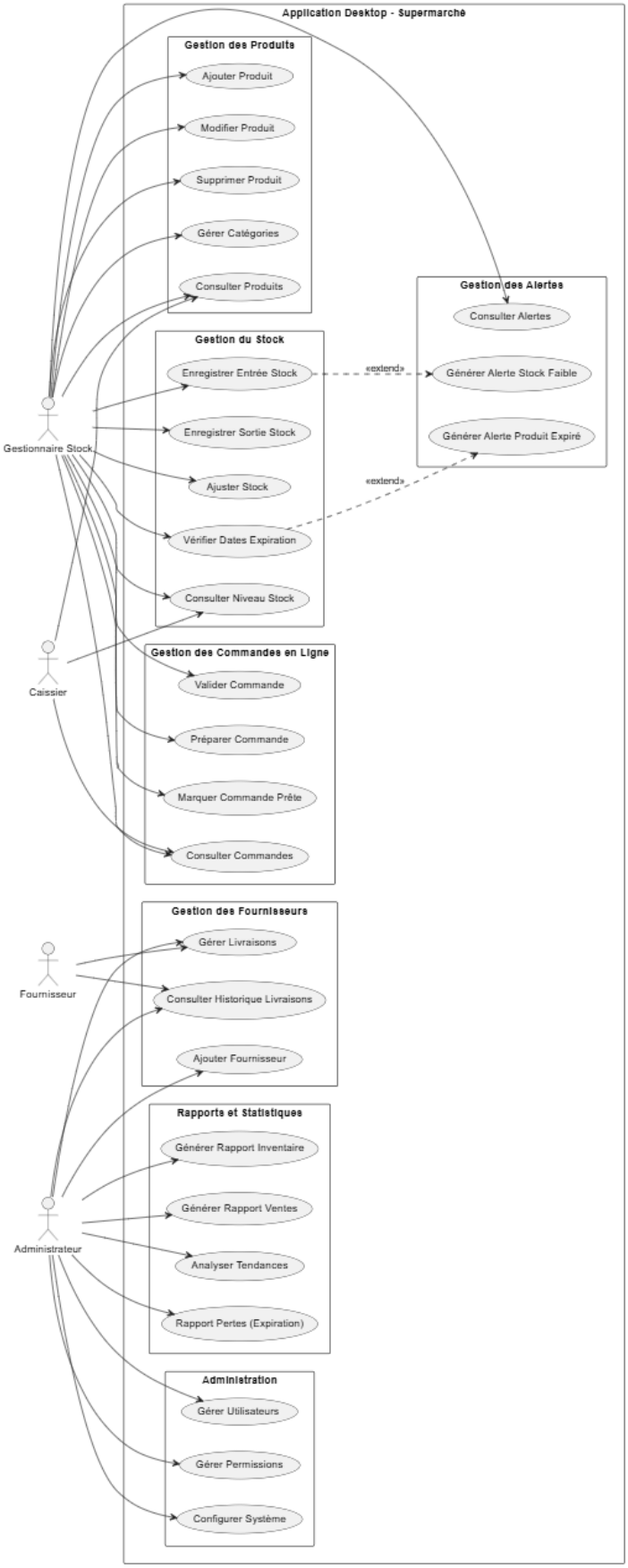


Figure 3.2 – Diagramme de cas d'utilisation — Vue détaillée

Description textuelle

Application Mobile (Client)

Le client utilise l'application mobile pour effectuer ses achats. Ses actions se regroupent en quatre catégories principales.

Navigation Produits

— **Parcourir Catalogue :** le client consulte l'ensemble des produits disponibles.

— **Rechercher Produit :** recherche ciblée par nom ou référence.

— **Filtrer par Catégorie :** restriction de l'affichage selon une catégorie (alimentaire, cosmétique, etc.).

— **Voir Détails Produit :** accès aux informations complètes d'un article (prix, description, disponibilité).

Gestion Panier

— **Ajouter au Panier :** sélection d'un produit et ajout au panier.

— **Modifier Quantité :** ajustement de la quantité d'un produit.

— **Supprimer du Panier :** suppression d'un produit du panier.

— **Voir Panier :** consultation du contenu et du montant total.

Commandes

— **Passer Commande :** transformation du panier en commande validée.

— **Suivre Commande :** suivi du statut (en attente, validée, livrée).

— **Consulter Historique :** accès aux commandes passées.

— **Choisir Mode Paiement :** sélection du mode de règlement.

— **Effectuer Paiement :** validation du paiement pour finaliser la commande.

Compte Client

— **S'inscrire :** création d'un compte utilisateur.

— **Se Connecter :** authentification pour accéder aux services personnalisés.

— **Gérer Profil :** modification des informations personnelles.

— **Gérer Adresse :** ajout ou modification des adresses de livraison.

Application Backoffice — Supermarché L'administrateur utilise l'application backoffice pour piloter les opérations internes. Ses actions s'organisent en cinq grandes catégories.

Gestion des Produits

— Ajouter / Modifier / Supprimer Catégorie : gestion des catégories de produits.

— Ajouter / Modifier / Supprimer Produit : gestion du catalogue des produits.

Gestion du Stock

— **Enregistrer Stock Entrant :** ajout des nouvelles livraisons au stock.

— **Enregistrer Stock Sortant :** retrait des produits vendus ou périmés.

— **Consulter État Stock / Vérifier Dates d'Expiration :** suivi des quantités disponibles et alertes.

Gestion des Commandes

— Préparer / Valider / Marquer Commande Prête.

— **Consulter Commandes :** visualisation des commandes passées par les clients.

Gestion des Fournisseurs

— Ajouter / Modifier / Supprimer Fournisseur : gestion des partenaires fournisseurs.

Rapports et Statistiques

— Générer Rapport de Ventes, de Stock et de Commandes.

Ce diagramme de cas d'utilisation montre clairement :

— Les interactions du client avec l'application mobile pour naviguer, gérer son panier, passer commande et administrer son compte.

— Les interactions de l'administrateur du supermarché avec l'application backoffice pour gérer produits, stock, commandes, fournisseurs et rapports.

3.2 Conception Statique

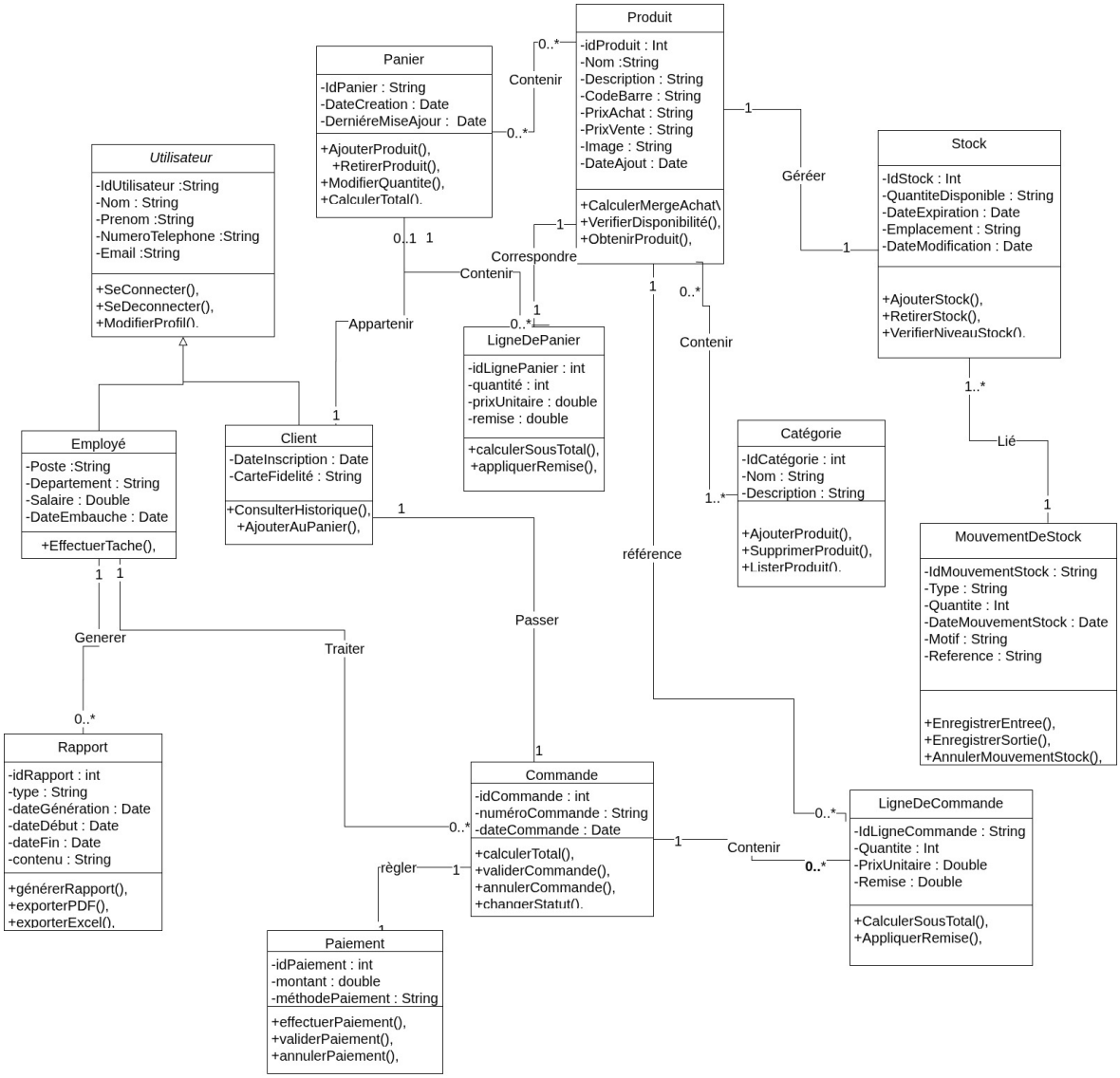


Figure 3.3 – Diagramme de classes — Structure statique du système

Cette partie représente l'organisation interne du système à travers le diagramme de classes.

3.2.1 Description des Classes

Le diagramme de classes UML décrit la structure statique du système. Il définit les entités principales, leurs attributs, leurs méthodes et les relations qui les unissent. Ce modèle a servi de référence directe lors de la conception orientée objet et de l'implémentation.

Table 3.2 – Description des Domaines et Classes

Domaine	**Classes clés**	**Objectif Technique & Métier**
Gestion des Accès	Utilisateur, Client, Employé	Modéliser la hiérarchie des utilisateurs via l'héritage. Cela permet de centraliser la sécurité tout en isolant les fonctionnalités spécifiques (fidélité pour le client, gestion pour l'employé).
Flux Transactionnel	Panier, LigneDePanier, Commande, LigneDeCommande, Paiement	Sécuriser le processus d'achat. Le passage d'un état temporaire (Panier) à un état persistant (Commande) garantit l'intégrité des prix et des quantités lors du paiement.
Catalogue	Produit, Catégorie, Stock, MouvementDeStock	Assurer la disponibilité et la traçabilité. Chaque article est classifié, et chaque flux physique est enregistré pour automatiser les alertes de stock et de péremption.
Reporting & Analyse	Rapport	Valoriser les données du système. Ce module extrait les informations des autres domaines pour générer des indicateurs de performance (ventes, état des stocks) exportables.

3.3 Conception Dynamique

La conception dynamique s'intéresse au comportement du système dans le temps : comment les composants se parlent, dans quel ordre les opérations s'enchaînent, et ce qui se passe selon les différents scénarios.

3.3.1 Diagramme d'Activité

A — Application Mobile

1. *Authentification et gestion du compte :*
   - Début → S'inscrire (création de compte).
   - Se connecter (authentification avec identifiants).
   - Gérer profil (modifier nom, email, téléphone).
   - Gérer adresse (ajouter ou modifier adresse de livraison).

— Fin si déconnexion.

2. *Navigation produits :*
   — Début → Parcourir catalogue.
   — Rechercher produit (par nom ou code-barre).
   — Filtrer par catégorie.
   — Voir détails produit (prix, description, disponibilité).
   — Décision : ajouter au panier ou quitter.
3. *Gestion du panier :*
   — Début → Ajouter au panier.
   — Modifier quantité.
   — Supprimer du panier.
   — Voir panier (contenu + montant total).
   — Décision : valider panier ou continuer achats.
4. *Commandes et paiements :*
   — Début → Passer commande (transformation du panier en commande).
   — Choisir mode de paiement (carte, espèces, etc.).
   — Effectuer paiement.
   — Si paiement validé → mise à jour du stock + génération facture.
   — Si paiement refusé → retour à la gestion du panier.
   — Suivre commande (statut : en attente, validée, livrée).
   — Consulter historique (commandes passées).
   — Fin : confirmation envoyée au client.

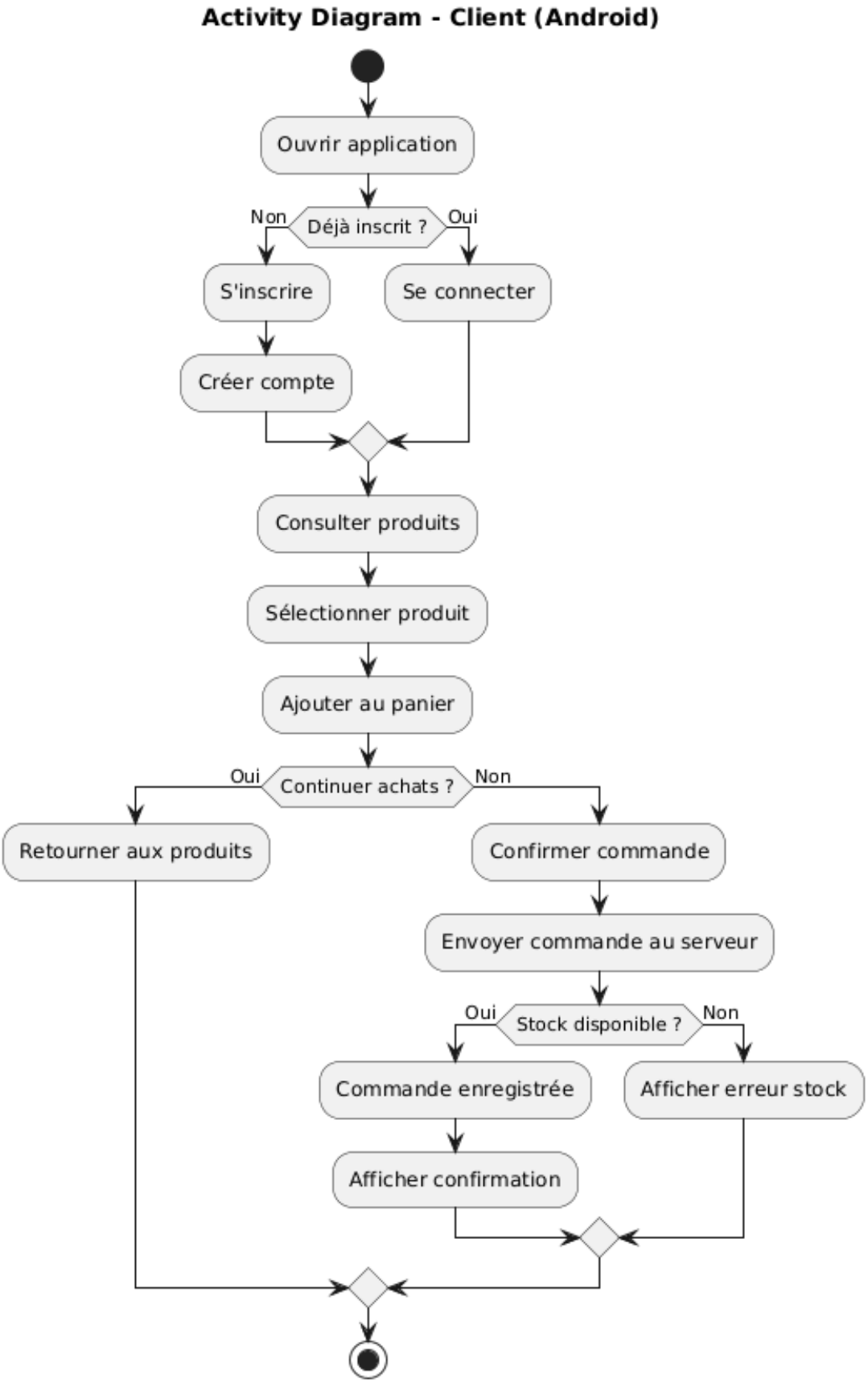


Figure 3.4 – Diagramme d'activité — Application Mobile (Client)

B — Application Desktop

1. *Gestion des produits :*
   - Début → Ajouter / Modifier / Supprimer catégorie.
   - Ajouter / Modifier / Supprimer produit.
   - Fin : catalogue mis à jour.
2. *Gestion du stock :*
   - Début → Enregistrer stock entrant (livraison fournisseur).
   - Enregistrer stock sortant (vente ou retrait produit périmé).
   - Consulter état stock (quantité disponible, dates de péremption).
   - Décision : alerte si seuil critique atteint.
   - Fin : stock mis à jour.
3. *Gestion des commandes et livraisons :*
   - Début → Consulter commandes.
   - Valider / Annuler commande.
   - Gérer livraisons (planification, suivi).
   - Fin : commande traitée.
4. *Gestion des fournisseurs :*
   - Début → Ajouter / Modifier / Supprimer fournisseur.
   - Fin : base fournisseurs mise à jour.
5. *Rapports et statistiques :*
   - Début → Générer rapport de ventes / de stock / de commandes.
   - Exporter PDF / Excel.
   - Fin : rapport disponible pour analyse.

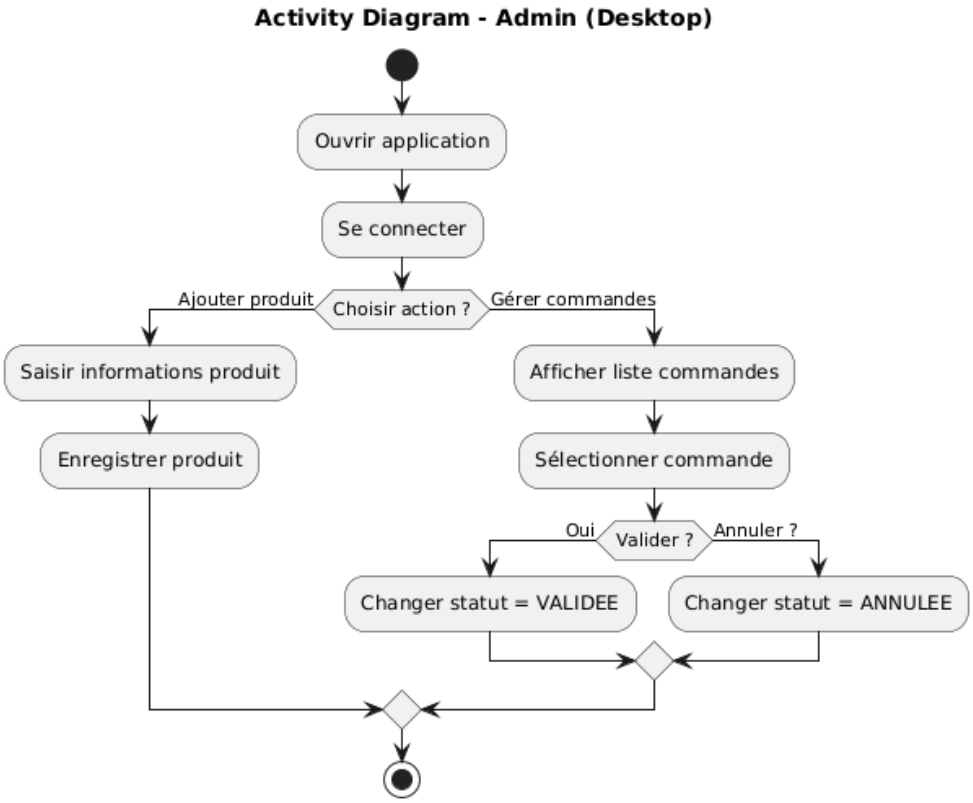


Figure 3.5 – Diagramme d'activité — Application Desktop (Supermarché)

3.4 Diagramme de Séquence du Système

But : montrer les interactions entre objets/acteurs dans le temps.

Caractéristiques :

Les acteurs/objets sont en haut, chacun avec une ligne de vie.

Les messages sont des flèches horizontales

Le temps descend verticalement.

Utilité : détaille la logique des interactions et aide à programmer les méthodes.

A.a-partie Application `desktop` :

Acteurs et composants

— **Admin** : utilisateur du système (administrateur du supermarché).

— **Desktop App** : application de bureau utilisée par l'admin.

— **API Server** : serveur applicatif qui reçoit les requêtes et les traite.

— **Database** : base de données où sont stockées les informations (produits, commandes, etc.).

1. **Ajouter Produit**

   — L'**Admin** envoie la commande *Ajouter produit* à l'**Application Desktop**.

   — L'**Application Desktop** transmet une requête `POST` */product* à l'**API Server**.

   — L'**API Server** envoie l'instruction *Insert product* à la **Database**.

   — La **Database** répond *OK* à l'**API Server**.

   — L'**API Server** renvoie une *Confirmation* à l'**Application Desktop**, qui l'affiche à l'**Admin**.

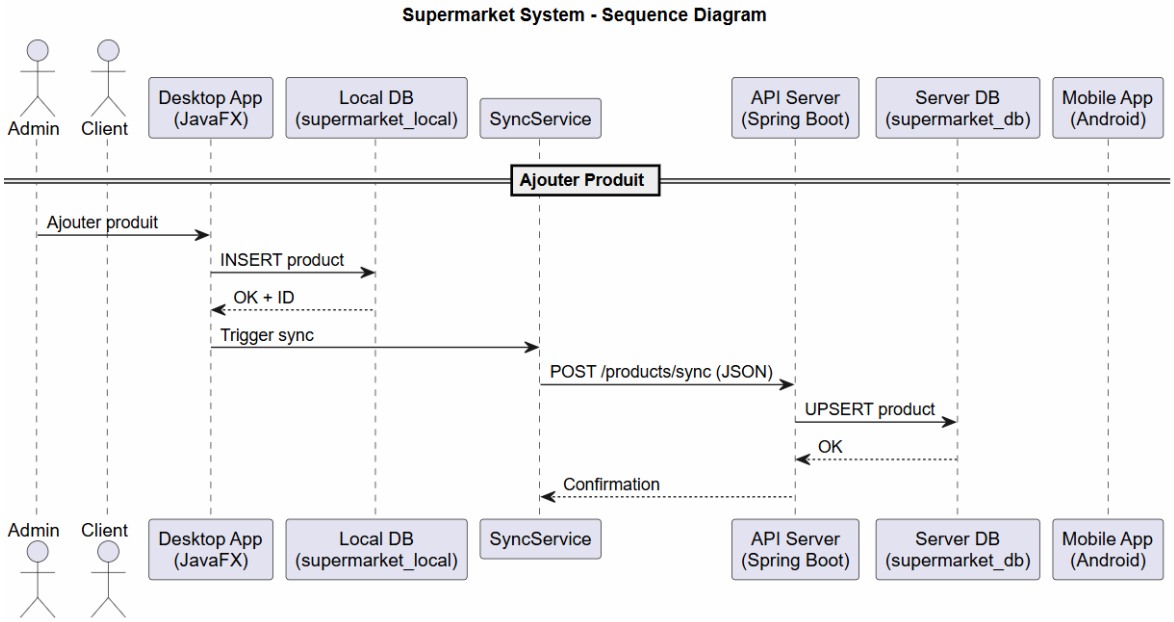


Figure 3.6 – Diagramme de séquence — Application desktop

2. **Voir Commandes**

   — L'**Admin** demande *Voir commandes* à l'**Application Desktop**.

   — L'**Application Desktop** envoie une requête `GET` */orders* à l'**API Server**.

   — L'**API Server** interroge la **Database** avec *Fetch orders*.

   — La **Database** renvoie la *Liste commandes* à l'**API Server**.

   — L'**API Server** transmet *Affichage commandes* à l'**Application Desktop**, qui affiche les résultats à l'**Admin**.

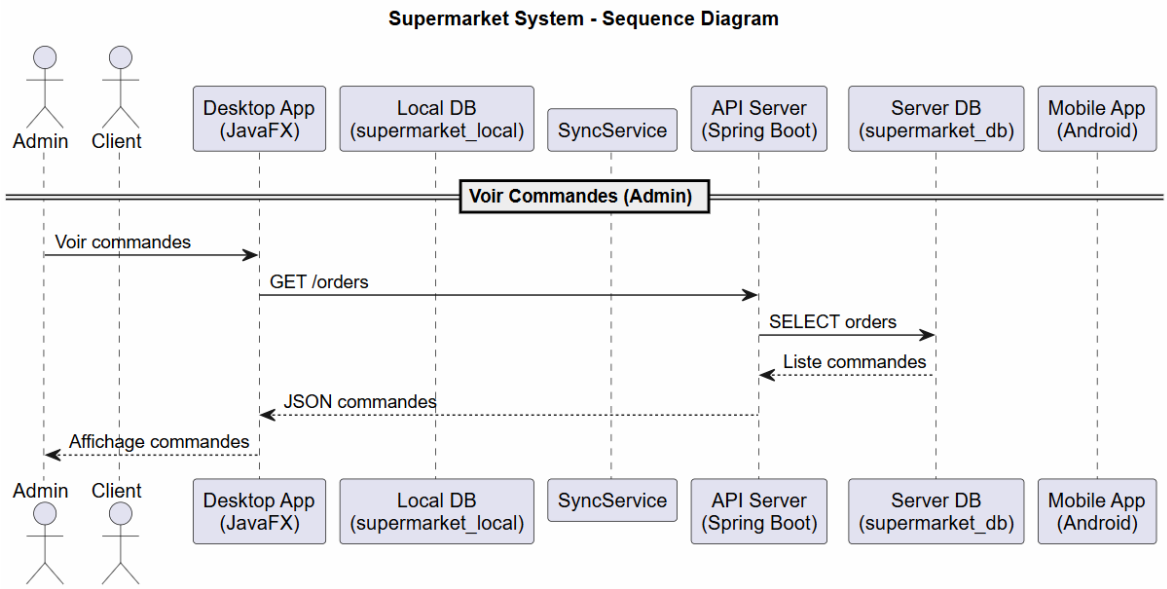


Figure 3.7 – Diagramme de séquence — Application desktop

3. **Valider Commande**

   — L'**Admin** envoie *Valider commande* à l'**Application Desktop**.

   — L'**Application Desktop** transmet une requête `PUT` */order/{id}* à l'**API Server**.

   — L'**API Server** envoie *Update statut = VALIDÉE* à la **Database**.

   — La **Database** répond *OK* à l'**API Server**.

   — L'**API Server** renvoie une *Confirmation* à l'**Application Desktop**, qui l'affiche à l'**Admin**.

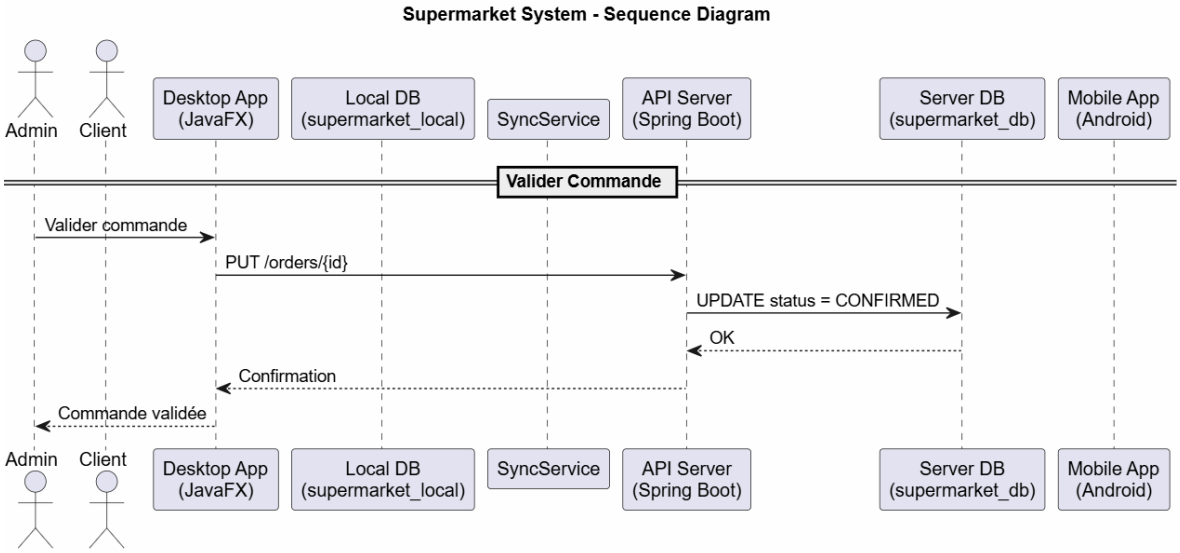


Figure 3.8 – Diagramme de séquence — Application desktop

Ce diagramme de séquence met en évidence :

— Le rôle de l'**Admin** qui initie les actions.

— Le rôle de l'**Application Desktop** comme interface utilisateur.

— Le rôle de l'**API Server** comme intermédiaire logique.

A.b – Application Mobile :

1. **Inscription (S'inscrire)**
   - Le **Client** envoie la demande *S'inscrire* via l'**Android App**.
   - L'**Android App** transmet une requête `POST` */register* à l'**API Server**.
   - L'**API Server** enregistre le nouvel utilisateur dans la **Database** (*Save User*).
   - La **Database** confirme avec *OK*.
   - L'**API Server** renvoie *Compte créé* à l'**Android App**.
   - L'**Android App** affiche la *Confirmation* au Client.
2. **Consulter Produits (Voir Produits)**
   - Le **Client** demande *Voir produits* à l'**Android App**.
   - L'**Android App** envoie une requête `GET` */products* à l'**API Server**.
   - L'**API Server** interroge la **Database** (*Fetch products*).
   - La **Database** renvoie la *Liste produits*.
   - L'**API Server** transmet les données (*JSON produits*) à l'**Android App**.
   - L'**Android App** affiche les produits au Client.

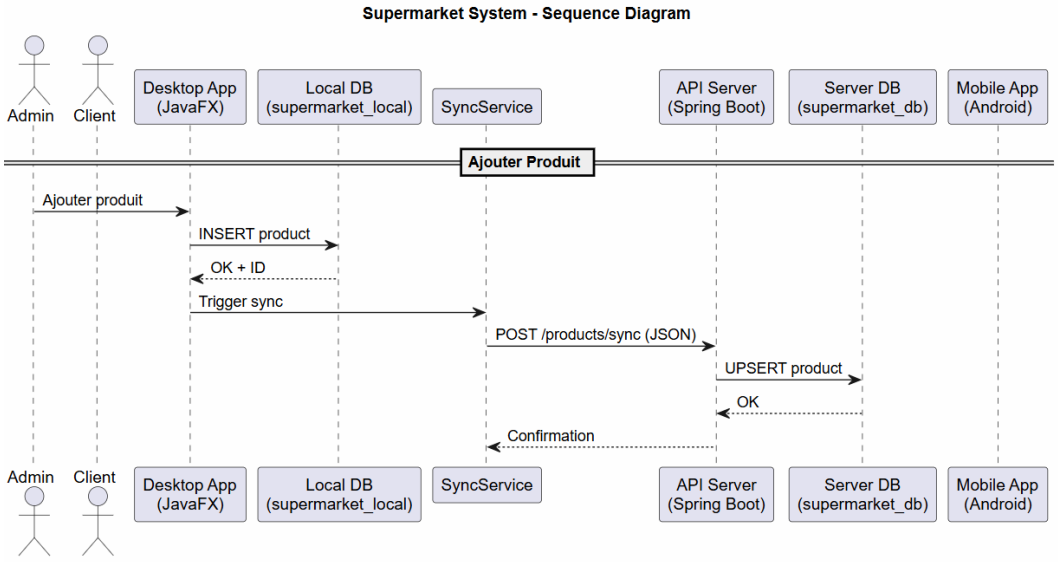


Figure 3.9 – Diagramme de séquence — Application Mobile (Client)

3. **Passer Commande (Confirmer commande)**

— Le **Client** envoie *Confirmer commande* à l'**Android App**.

— L'**Android App** transmet une requête `POST` */order* à l'**API Server**.

— L'**API Server** demande à la **Database** de *Vérifier stock*.

Cas 1 – Stock disponible :

— La **Database** répond *Stock OK*.

— L'**API Server** envoie *Diminuer stock* à la **Database**.

— L'**API Server** enregistre la commande (*Enregistrer commande*).

— La **Database** confirme avec *OK*.

— L'**API Server** renvoie *Confirmation commande* à l'**Android App**.

— L'**Android App** affiche *Commande envoyée* au Client.

Cas 2 – Stock insuffisant :

— La **Database** renvoie *Erreur stock*.

— L'**API Server** transmet un *Message d'erreur* à l'**Android App**.

— L'**Android App** transmet un Message d'erreur à l'**Android App.**

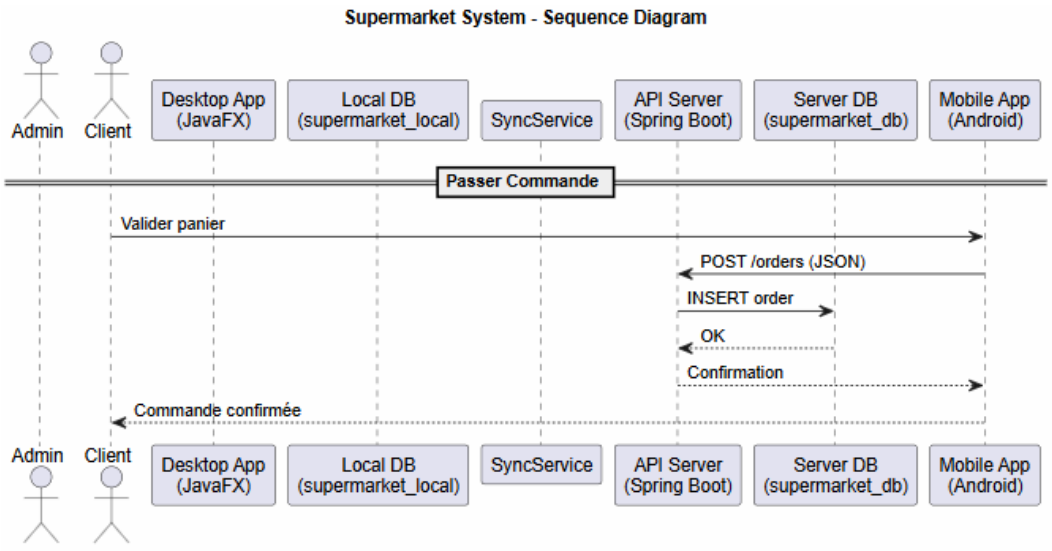


Figure 3.10 – Diagramme de séquence — Application Mobile (Client)

Ce diagramme de séquence côté **Client** couvre :

— La création de compte (inscription).

— La consultation du catalogue de produits.

— Le passage de commande avec vérification du stock.

Conclusion (partie modélisation) :

La phase de modélisation nous a permis de clarifier l'architecture du système avant d'écrire une seule ligne de code. En travaillant avec UML, nous avons pu traduire des besoins métier parfois imprécis en une architecture technique cohérente et vérifiable.

-Analyse fonctionnelle : (Diagramme de cas utilisation) : a délimité avec précision les responsabilités de l'administrateur sur l'application `desktop` et les fonctionnalités proposées aux clients sur l'application mobile.

-La conception statique : (Diagramme de classe) : a servi de base directe à la création de la base de données et à la définition de nos entités principales (produit, commande, stock).

-La conception dynamique : (Diagramme de séquence et Activités) :

A permis d'anticiper les flux de données et les interactions entre le `front end` et l'`API Spring Boot`, ce qui a réduit le risque d'erreurs lors du codage.

Chapitre 4

Réalisation et Implémentation

4.1 Introduction

Ce chapitre décrit la phase de réalisation du système de gestion de supermarché avec commande en ligne. Nous y présentons comment les modèles UML ont été traduits en code : développement de l'application mobile, du `backend` (API REST) et de l'application `desktop`, avec leurs interactions.

Nous y précisons aussi les outils retenus, les choix technologiques et les principales interfaces développées.

4.2 Environnement de développement

Le développement du projet a mobilisé plusieurs outils complémentaires, choisis pour leur adéquation avec les technologies utilisées :

- **IDE (Environnement de Développement Intégré)**
  - **Android Studio** : Utilisé pour développer l'application mobile cliente. Il intègre un débogueur, un éditeur XML pour les interfaces et un émulateur `Android`.
  - **IntelliJ IDEA** : Utilisé pour le développement du `backend` avec `Spring Boot` et de l'application `desktop` (JavaFX).
- **Langages de programmation**
  - **Java** : Langage principal pour le `backend` (`Spring Boot`) et l'application `Android`. Sa structure orientée objet (POO) facilite l'organisation du code.
  - **XML** : Utilisé pour concevoir les interfaces utilisateur (*layouts Android*).

- **Frameworks et bibliothèques**
  - **Spring Boot** : *Framework* utilisé pour développer l'API REST qui assure la communication entre l'application mobile et le serveur.
  - **Retrofit** : Bibliothèque intégrée dans l'application mobile pour effectuer les requêtes HTTP vers l'API.
  - **Android Intents** : Gèrent la navigation entre les activités et le transfert de données d'un écran à l'autre.
  - **RecyclerView** : Composant Android utilisé pour afficher les listes de produits de façon performante.
  - **Material Design** : Utilisé pour concevoir une interface claire et cohérente.

- **Gestionnaire de dépendances**
  - **Gradle** : Utilisé pour gérer les dépendances et automatiser la compilation du projet Android.

- **Formats de données**
  - **JSON** : Utilisé pour l'échange de données entre l'API REST (Spring Boot) et l'application mobile Android. Il permet de structurer les réponses et requêtes HTTP de manière lisible et légère.

A – Partie Application Mobile

4.2.1 Architecture Technique de l'Application Mobile

L'application mobile suit une architecture modulaire inspirée du modèle MVC (Model-View-Controller). Ce découpage sépare clairement la logique métier, l'interface utilisateur et la gestion des données.

Cette organisation facilite la maintenance du code et son évolution future.

4.2.2 Structure du projet

Le projet est organisé en plusieurs `packages` principaux :

- **Activities** : Gèrent le cycle de vie des écrans (`LoginActivity`, `HomeActivity`, `CartActivity`). Elles forment la couche vue et contrôleur.
- **Adapters** : Assurent la liaison entre les données (liste de produits) et les composants visuels comme le `RecyclerView`.
- **Models** : Contiennent les classes métier (ex : `Product`, `Order`) correspondant aux données échangées avec l'API.
- **Managers** : Comme le `CartManager`, chargé de gérer l'état du panier. Il est implémenté avec le pattern Singleton pour garantir une instance unique accessible depuis toutes les activités.
- **Network (API)** : Contient les classes liées à la communication avec le backend :
  - **ApiService** (définition des *endpoints*)
  - **RetrofitClient** (configuration des requêtes HTTP)

4.2.3 Flux des données

Le fonctionnement de l'application repose sur le flux suivant :

1. L'application mobile envoie une requête HTTP via Retrofit.
2. Le serveur renvoie une réponse JSON.
3. Les données sont converties en objets Java (`Models`).
4. Les Adapters affichent les données dans le RecyclerView.

4.3 Présentation des Interfaces et Fonctionnalités

Les interfaces ont été conçues en tenant compte des principes d'UX et d'UI, pour offrir une navigation claire et des actions accessibles sans effort particulier.

4.3.1 Authentification et Accès

- **Interface de Connexion (LoginActivity) :** Permet au client de s'authentifier et donne accès à l'écran d'inscription pour les nouveaux utilisateurs.
- **Interface d'Inscription (SignUpActivity) :** Permet aux nouveaux utilisateurs de créer un compte.

4.3.2 Navigation et Catalogue (Interface "Rayons")

L'écran principal (HomeActivity) affiche les produits disponibles.

- Les données sont récupérées depuis le serveur via une requête **GET /products** grâce à Retrofit.
- Les produits sont affichés à l'aide d'un RecyclerView associé à un Adapter.

Fonctionnalités :

- **Filtrage par catégorie** : via un HorizontalScrollView, permettant de filtrer dynamiquement les produits.
- **Recherche** : possibilité de rechercher un produit par nom.
- **Ajout au panier** : chaque produit possède un bouton permettant de l'ajouter au panier via le CartManager.

4.3.3 Gestion de la commande (Interface "Panier")

L'écran **CartActivity** permet au client de consulter les produits sélectionnés. *Fonctionnalités :*

- Affichage dynamique des articles ajoutés.
- Calcul automatique du total en Dinar Algérien (DA).
- Modification du panier (ajout / suppression).

Le panier est géré via le CartManager, implémenté en `Singleton` pour conserver les données entre les écrans.

4.3.4 Finalisation (Interface "Checkout")

Cet écran représente la dernière étape du processus d'achat. *Fonctionnalités :*

- Saisie des informations client (nom, adresse).
- Choix du mode de paiement (cash ou carte).
- Validation de la commande.

Lors de la validation :

- Une requête **POST /orders** est envoyée au serveur via Retrofit.
- La commande est enregistrée dans la base de données.
- Un message de confirmation est affiché à l'utilisateur.

4.3.5 Justifications Techniques et Concepts de Développement

Plusieurs choix techniques ont guidé la réalisation de l'application mobile, avec pour objectifs la performance, la maintenabilité et la qualité de l'expérience utilisateur.

- **Adapter (Pattern Design)**

Nous avons implémenté un ProductAdapter pour relier les données (liste d'objets Product) à leur représentation visuelle dans le RecyclerView. *Ce choix permet :*

- de séparer la logique de présentation des données.
- de faciliter la mise à jour dynamique de l'interface.
- d'optimiser les performances grâce au recyclage des vues.

- **Singleton (CartManager)**

Le CartManager a été implémenté avec le pattern Singleton, garantissant une seule instance partagée dans toute l'application. *Ce choix permet :*

- de maintenir l'état du panier entre les différentes activités.
- d'éviter la duplication des données.
- d'assurer une cohérence globale des informations du panier.

- **Retrofit (Communication API)**

La bibliothèque Retrofit a été utilisée pour gérer la communication entre l'application mobile et le `backend`. *Ce choix permet :*

- de simplifier l'envoi de requêtes HTTP (GET, POST).
- de gérer automatiquement la conversion JSON $\leftrightarrow$ objets Java.
- de structurer clairement les appels API via une interface (`ApiService`).

- **Architecture Modulaire**

L'application a été organisée en plusieurs packages (Activities, Models, Adapters, Network), ce qui permet :

- une séparation des responsabilités.
- une meilleure maintenabilité du code.
- une facilité d'évolution et d'ajout de nouvelles fonctionnalités.

- **Material Design**

Les composants **Material Design** ont été utilisés pour concevoir l'interface utilisateur. *Ce choix permet :*

- d'assurer une cohérence visuelle.
- d'améliorer l'expérience utilisateur (UX).
- de respecter les standards modernes des applications Android.

Conclusion de la phase implémentation mobile

Le développement de l'application mobile sous Android Studio a concrétisé les besoins fonctionnels identifiés pour les clients du supermarché. En s'appuyant sur Java et les composants Material Design, nous avons obtenu une interface à la fois réactive et agréable à utiliser.

Le recours aux patrons de conception — Singleton pour la persistance du panier et Adapter pour la fluidité des listes — donne à l'application une base stable. Cette étape valide la faisabilité technique du projet : le client peut parcourir les rayons, gérer son panier en temps réel et finaliser sa commande sans friction.

B - Implémentation du Backend (Spring Boot)

Le `backend` constitue le serveur central du système. Il sert d'intermédiaire entre l'application mobile, l'application `desktop` et la base de données distante. Il reçoit les requêtes HTTP, applique la logique métier et s'assure que les données sont correctement persistées.

Il centralise aussi les informations accessibles aux clients mobiles, tout en restant synchronisé avec la base locale du `desktop`.

Architecture et Frameworks

Nous avons choisi **Spring Boot** pour développer une API REST maintenable et extensible.

L'architecture du backend repose sur quatre couches distinctes, avec une séparation claire des responsabilités :

- **Couche Entité (JPA / Hibernate)**
  Cette couche définit la structure des données sous forme de classes Java annotées avec `@Entity`. Par exemple, la classe `CustomerOrder` correspond à la table des

commandes dans la base de données. Grâce à **JPA (Java Persistence API)** et **Hibernate**, les objets Java sont automatiquement traduits en enregistrements SQL (mapping objet-relationnel).

- **Couche Repository**
  Cette couche repose sur des interfaces comme `OrderRepository`, qui héritent de JpaRepository. Elle permet :
  - d'accéder aux données.
  - d'effectuer des opérations CRUD (Create, Read, Update, Delete).
  - sans écrire de requêtes SQL manuellement.
- **Couche Service**
  La couche Service (ex : `OrderService`) contient la **logique métier de l'application**. Elle permet :
  - de traiter les données (calculs, validations).
  - d'appliquer les règles métier (ex : définir le statut initial d'une commande à PENDING).
  - de faire le lien entre Controller et Repository.
- **Couche Controller (API REST)**
  La couche Controller (ex : `OrderController`) est la porte d'entrée du `backend`. Elle expose des endpoints REST accessibles via HTTP :
  - `GET /products` → récupérer les produits.
  - `POST /orders` → créer une commande.
  - `PUT /orders/{id}` → mettre à jour une commande.
- **Tests et Validation (Postman)**
  Postman a été utilisé pour tester et valider le comportement de l'API REST. Il a permis de vérifier :
  - **Simulation HTTP** → envoyer des requêtes (GET, POST, PUT).
  - **Validation JSON** → vérifier la structure des données reçues.
  - **Débogage** → identifier les erreurs avant l'intégration.

Les échanges entre le client et le serveur se font au format **JSON (JavaScript Object Notation)**, un format léger et bien supporté par l'ensemble des composants du système.

C – Partie Desktop

4.3.6 Introduction

Ce chapitre présente la réalisation technique de l'application `desktop` de gestion de stock du supermarché. À l'issue de la phase de conception UML, nous avons développé une solution fonctionnelle permettant à l'administrateur de gérer les produits et de superviser les commandes.

4.3.7 Environnement de développement

Le développement de l'application `desktop` s'appuie sur les outils suivants :

- **IDE : IntelliJ IDEA**
  Utilisé pour le développement, le débogage et la gestion du projet.
- **Base de données locale : MySQL**
  Utilisée pour stocker les données du supermarché (produits, stock) directement sur la machine de l'administrateur.
- **Gestionnaire de projet : Apache Maven**
  Permet de gérer les dépendances et d'automatiser la compilation.

4.3.8 Langages et Technologies utilisés

- **Java :** Langage principal de l'application, utilisé pour implémenter la logique métier et assurer la robustesse du système.
- **JavaFX :** Framework graphique utilisé pour concevoir l'interface utilisateur (GUI). Il permet de créer des interfaces avec des composants comme :
  - TableView (affichage des produits)
  - Formulaires de saisie
  - Boutons d'action (ajout, modification, suppression)
- **SQL / MySQL :** Utilisé pour la gestion et la persistance des données locales.

4.3.9 Architecture et logique de l'application

L'application `desktop` suit une architecture structurée qui distingue interface utilisateur, logique métier et accès aux données.

Connectivité (JDBC)

La communication avec la base de données locale repose sur **JDBC (Java Database Connectivity)**. Cette API permet d'exécuter des requêtes SQL depuis le code Java :

- INSERT (ajout de produit)
- UPDATE (modification)
- DELETE (suppression)
- SELECT (lecture des données)

Logique métier

La logique métier est au cœur de l'application desktop. Elle assure plusieurs fonctions :

- Validation des données saisies (formats, champs obligatoires).
- Gestion du stock (quantité, alertes de stock faible).
- Traitement des opérations CRUD (Create, Read, Update, Delete).
- Préparation des données à synchroniser avec le serveur distant.

Synchronisation avec le serveur

Pour rendre les données accessibles depuis l'application mobile, un mécanisme de synchronisation a été mis en place :

- Lorsqu'un produit est ajouté ou modifié localement :
  - L'application `desktop` enregistre d'abord les données dans la base locale.
  - Ensuite, elle envoie une requête HTTP vers l'API distante.
- Le serveur met à jour la base de données distante, utilisée par l'application mobile.

Ce mécanisme garantit que les deux applications travaillent sur les mêmes données.

- Une performance rapide en local.
- Une disponibilité des données à distance.

4.4 PROCESSUS DE COMMUNICATION (Architecture Système Mobile, Desktop & Backend)

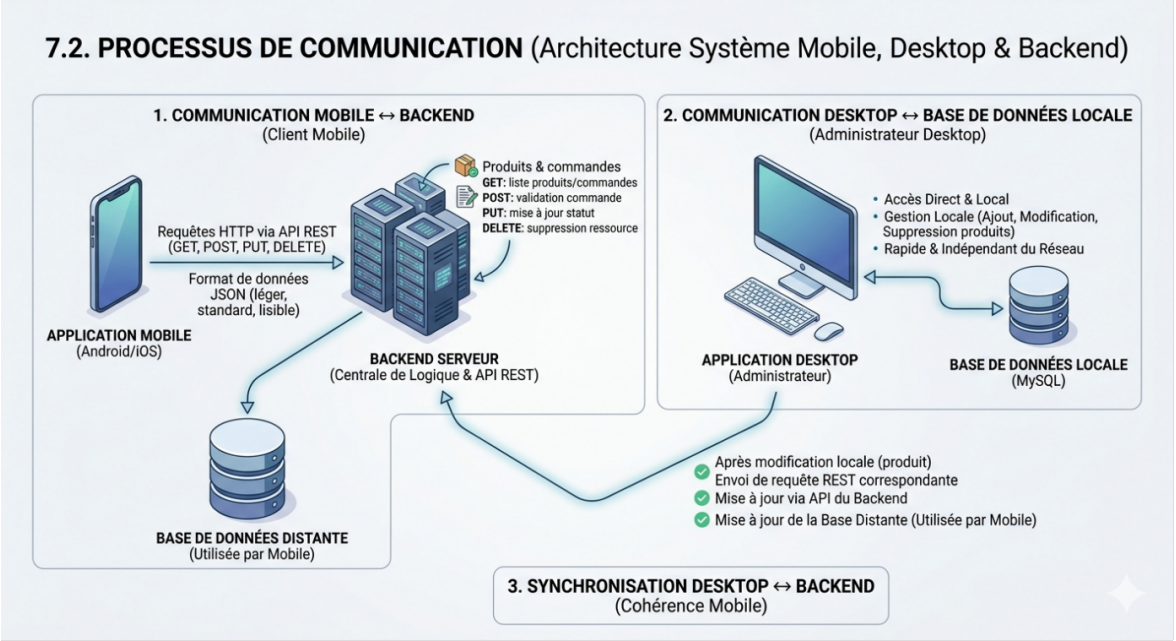


Figure 4.1 – Architecture Système Mobile, Desktop & Backend

La communication entre les composantes du système repose sur le protocole HTTP et l'échange de données au format **JSON (JavaScript Object Notation)**, un standard léger et bien adapté aux échanges entre applications hétérogènes.

- **Communication Mobile ↔ Backend**

L'application mobile communique directement avec le serveur via l'API REST exposée par le `backend`. Les opérations principales sont :

- **Requête GET** : récupère des données depuis le serveur, notamment la liste des produits et les commandes.
- **Requête POST** : envoie des données vers le serveur, par exemple lors de la validation d'une commande.
- **Requête PUT** : met à jour des informations existantes, comme le statut d'une commande (PENDING → CONFIRMED → DONE).
- **Requête DELETE** : supprime des ressources si nécessaire.
- **Communication Desktop ↔ Base de Données Locale**

L'application `desktop` interagit directement avec une base de données MySQL locale.

Toutes les opérations de gestion (ajout, modification, suppression) sont exécutées en local pour garantir de bonnes performances, indépendamment de la connexion réseau.

- **Synchronisation Desktop ↔ Backend**

Pour maintenir la cohérence entre la gestion interne (`desktop`) et les données accessibles en ligne (mobile), un mécanisme de synchronisation est utilisé :

- Lorsqu'un produit est ajouté, modifié ou supprimé en local, l'application `desktop` envoie une requête correspondante à l'API REST.
- Le serveur met à jour la base distante utilisée par l'application mobile.
- Ainsi, les clients consultent toujours des données à jour.

4.4.1 Sécurité et Persistance des Données

- **Base de Données :**
  Nous avons utilisé MySQL pour le stockage permanent des données. Le système repose sur deux bases distinctes :
  - Une **base locale** utilisée par l'application `desktop` pour des performances rapides et une disponibilité hors-ligne.
  - Une **base distante (serveur)** pilotée par l'API pour donner accès aux données depuis l'application mobile.
- **Synchronisation des Données :**
  Un mécanisme de synchronisation a été mis en place entre la base locale et la base distante. Les modifications effectuées dans l'application desktop (ajout, mise à jour) sont transmises au serveur afin que les données restent cohérentes pour l'application mobile.
- **Intégrité des Données :**
  Le backend contrôle les données reçues (format, cohérence, disponibilité du stock) avant tout enregistrement, ce qui renforce la fiabilité globale du système.
- **Interopérabilité :**
  Grâce à une API REST basée sur HTTP et JSON, le système assure une communication fiable entre :
  - L'application mobile.
  - L'application `desktop`.
  - Le serveur distant.

Cette architecture offre une bonne flexibilité et facilite les évolutions futures du système.

4.5 Conclusion

Cette phase d'implémentation illustre la mise en place d'une architecture reposant sur les **API REST**, qui permet une communication fiable entre les différentes parties du système.

L'application mobile offre une expérience utilisateur fluide côté client, tandis que le backend, via Spring Boot, garantit la cohérence et la sécurité des données. L'usage des patrons de conception (Singleton pour le panier, Repository pour les données) et des technologies choisies (Java 17, Spring Data JPA) donne au projet une base solide, stable et adaptée aux besoins réels d'un supermarché.

Conception de la base de données

La base de données est le cœur du système. Elle stocke les informations relatives aux produits et aux commandes, et assure la synchronisation entre la base locale (application `desktop`) et la base distante (serveur).

4.5.1 Présentation des tables

Notre base de données MySQL est organisée de façon à ce que l'instance locale regroupe toutes les tables décrites ci-dessous, tandis que l'instance serveur ne comporte que quatre tables : Products, Orders, Payments et Client.

Les tables principales :

- **Table Produit (Products)** : contient toutes les informations relatives aux produits disponibles dans le supermarché, telles que le nom, le prix, la quantité en stock, le code-barres et la catégorie associée.
- **Table Commande (Oreders)** : enregistre les commandes passées par les clients via l'application mobile ou desktop, avec les détails concernant les produits commandés, le montant total, le statut de la commande et la date de création.
- **Table Paiement** : gère les informations liées aux paiements des commandes, notamment le montant payé, le mode de paiement, le statut de la transaction et les références associées.
- **Table Client** : stocke les informations relatives aux clients, comme leur identifiant, leur date d'inscription et les données associées à leur carte de fidélité

La base de données comprend également un ensemble de tables secondaires qui assurent des fonctions de soutien et de gestion. Elles permettent d'affiner le niveau de détail des informations et de préserver la cohérence entre les entités principales.

On peut notamment citer :

- **Table Utilisateur** : gère les informations de base de tous les utilisateurs, qu'il s'agisse de clients ou d'employés.
- **Table Employé** : spécialisation de la table Utilisateur, dédiée aux données propres aux employés (poste, département, rémunération, etc.).
- **Table Panier** : représente l'espace virtuel d'un client, destiné à regrouper les produits sélectionnés avant la validation d'une commande.
- **Table LigneDePanier** : détaille les articles d'un panier, en indiquant la quantité choisie et les éventuelles remises.
- **Table LigneDeCommande** : décrit les produits inclus dans une commande, avec leurs caractéristiques transactionnelles (quantité, prix, remise).
- **Table Categorie** : classe les produits en groupes homogènes pour faciliter leur identification et leur gestion.
- **Table Stock** : suit les niveaux de disponibilité des produits, avec des informations sur la quantité, les dates de péremption et les mises à jour.
- **Table MouvementDeStock** : enregistre les flux de stock (approvisionnement, vente, retour) pour assurer la traçabilité et l'intégrité des données.
- **Table Rapport** : permet aux employés de générer et d'archiver des documents analytiques (ventes, stocks, activité) pour le suivi et la prise de décision.

4.5.2 Synchronisation des bases de données

L'architecture du système distingue deux types de stockage :

- Une **base locale** utilisée par l'application `desktop` pour des accès rapides.
- Une **base distante** hébergée sur un serveur et accessible via API REST pour l'application mobile.

La synchronisation s'effectue de manière asynchrone :

- Les produits ajoutés ou modifiés en local sont transmis vers le serveur.
- Les commandes créées via l'application mobile sont récupérées par l'application `desktop`.

4.6 Présentation des interfaces :

4.6.1 Gestion des produits :

Panneau principal pour gérer l'inventaire.

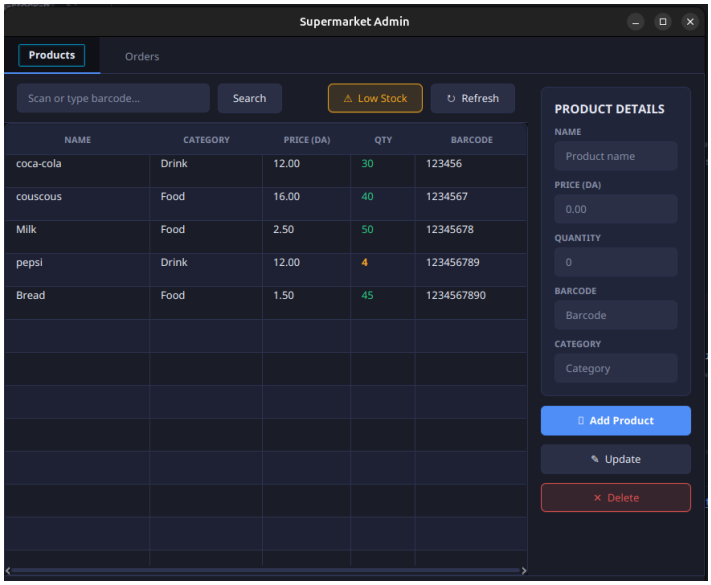


Figure 4.2 – INTERFACE SUPERMARKET ADMIN -PRODUCTS

4.6.2 Gestion des Commandes :

Permet de classer les produits (Aliments, Boissons, etc.)

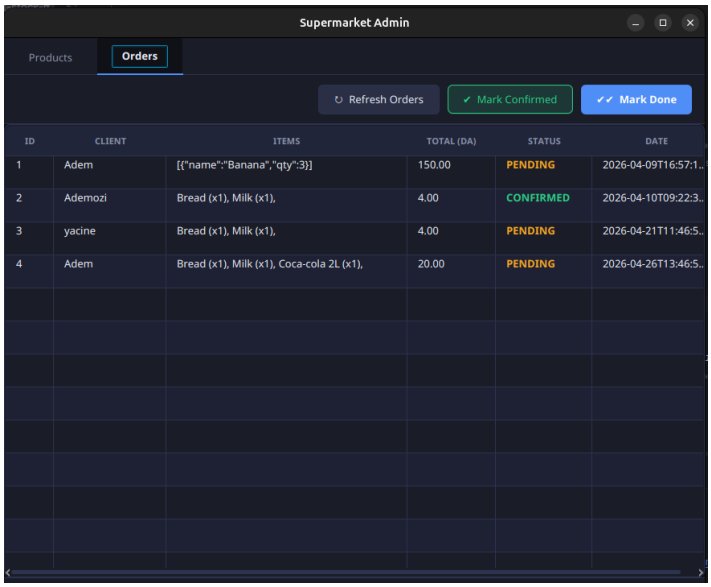


Figure 4.3 – INTERFACE SUPERMARKET ADMIN -ORDERS

4.6.3 Présentation de l'application mobile

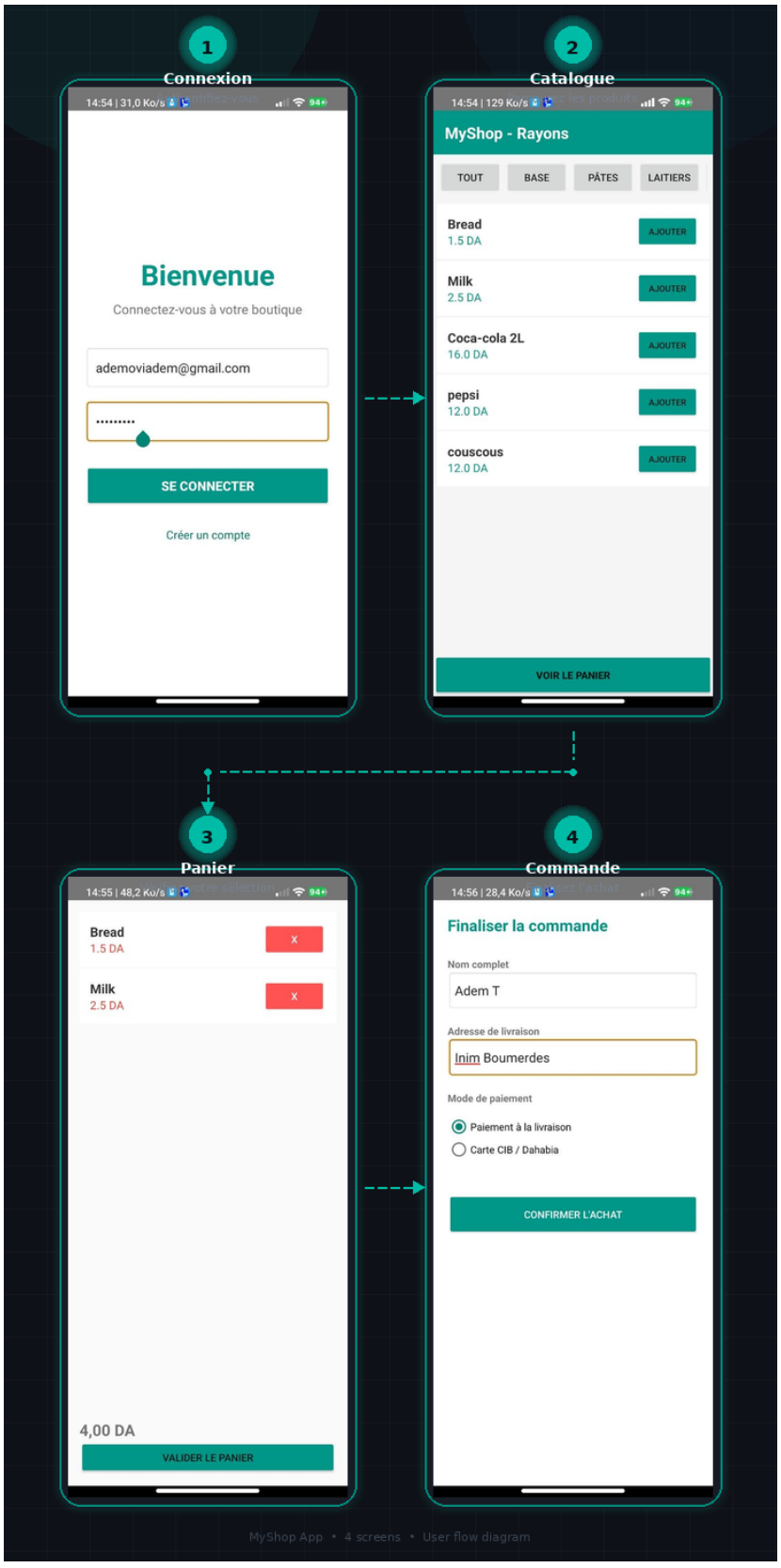


Figure 4.4 – INTERFACE SUPERMARKET APP MOBILE

Conclusion du chapitre 4

Ce chapitre a présenté la phase de réalisation du système, en mettant en évidence les choix techniques et architecturaux adoptés. L'utilisation de JDBC assure une interaction efficace et rapide avec la base de données locale pour les opérations CRUD.

La logique métier joue un rôle central dans la validation des données, la gestion des stocks et la préparation de la synchronisation.

L'architecture repose sur une séparation claire entre l'application desktop, l'application mobile et le backend, communiquant via une API REST (HTTP/JSON) pour garantir une interopérabilité fluide. De plus, un mécanisme de synchronisation entre les bases locale et distante maintient la cohérence des données, alliant rapidité locale et accessibilité à distance.

Enfin, la conception de la base de données et la présentation des interfaces illustrent la structure globale du système, confirmant la robustesse de cette solution pour un supermarché moderne.

Conclusion Générale :

Ce mémoire a présenté la conception, le développement et la réalisation d'un système de gestion de supermarché basé sur une architecture multi-plateforme (desktop, mobile et backend). L'objectif principal était de proposer une solution moderne, efficace et cohérente pour la gestion des produits, des commandes et des clients.

Le travail a commencé par l'analyse des besoins et la conception du système, notamment à travers la modélisation de la base de données et l'architecture globale. Ensuite, la phase d'implémentation a permis de mettre en œuvre les différentes fonctionnalités en utilisant des technologies adaptées telles que Java, Spring Boot, JDBC et MySQL.

L'architecture proposée, basée sur une API REST et une synchronisation entre base locale et distante, garantit à la fois performance, interopérabilité et cohérence des données entre les différentes applications. Cette approche permet de combiner l'efficacité d'un traitement local avec l'accessibilité des données à distance.

Enfin, ce projet a permis de consolider des compétences techniques en développement logiciel, en conception de bases de données et en architecture distribuée. Il ouvre également la voie à plusieurs améliorations futures, notamment l'optimisation de la synchronisation, l'ajout de fonctionnalités avancées et l'amélioration de l'expérience utilisateur.

Bibliographie

[1] C. Solnon, Cours de Conception Orientée Objet et UML, INSA de Lyon, Département Informatique (3IF).

[2] Université Joseph Kasa-Vubu, Supports de cours de Modélisation et Systèmes d'Information, Faculté des Sciences Informatiques, République Démocratique du Congo (support interne de cours).

[3] P. Roques, UML 2 par la pratique : Études de cas et exercices corrigés, 7e édition, Eyrolles.

[4] Oracle Corporation, Java SE Documentation. Disponible sur :`https://docs.oracle.com/en/java/`.

[5] Oracle Corporation, MySQL Documentation. Disponible sur : `https://dev.mysql.com/doc/`.

[6] Spring Team, Documentation officielle Spring Boot. Disponible sur : `https://spring.io/projects/spring-boot`.

[7] OpenJFX Community, Documentation JavaFX. Disponible sur : `https://openjfx.io/`.

[8] Google Developers, Android Studio & Java Development Documentation. Disponible sur : `https://developer.android.com/studio`.

[9] Postman, Postman API Platform Documentation. Disponible sur : `https://www.postman.com/`.

[10] GitHub, GitHub Documentation. Disponible sur : `https://docs.github.com/`.