

---

# Rapport de Projet

## Génie Logiciel

---

*Réalisé par :*  
Merlin BAVIERE  
Quentin VICENS

# 1 Introduction

Ce rapport présente la conception et l'implémentation d'un système de caisse de supermarché. Ce système permet aux clients de choisir des articles du magasin et de passer à la caisse pour acheter ces articles. De plus, il est possible de créer des plans de réductions qui s'appliqueront au prix payé par le client, ou encore de modifier les pricing policies des articles vendus dans le magasin et enfin il est possible de payer via un système bancaire.

## 2 Décisions de Conception

Afin d'assurer la flexibilité du code, chaque composant majeur a été conçu comme une entité indépendante : le Supermarché, la Banque, le Système d'Autorisation des Transactions (TAS) et le Terminal de Paiement (POS). Cela permet de modifier ou d'ajouter de nouvelles fonctionnalités sans impacter le reste du système et aussi de tester plus facilement le code. Les choix de conceptions suivants ont été fait pour respecter les requirements :

- R1 : On crée un "panier". C'est une liste des items que le clients va prendre en rayons dans le supermarché. Cette liste devra être vidée après le paiement du client.
- R2 : Ce dit "panier" consiste en une liste d'items, on décide d'utiliser une map afin de faire correspondre chaque items (qu'on utilise comme clé) de la liste à une quantité que le client aura choisi.
- R3 : chaque item à un prix. Pour les fruits et légumes on fait le choix de prendre poser un prix unitaire moyen. Un axe d'amélioration serait de proposer un prix pour un kilogramme d'un tel fruit ou légume.
- R4 : On choisit de faire en sorte que le système ne permette que de payer par carte. Un axe d'amélioration serait de pouvoir proposer d'autres modes de paiement, en liquide notamment. On ajoute au Customer un attribut rememberedPin. De ce fait il existe pour lui une possibilité de tester si il se trompe de code. Si le pin qu'il donne n'est pas celui de la carte alors le paiement sera annulé.
- R5 : Les clients peuvent souscrire à des programmes de réduction. Les programmes prime platinum et normal sont ajouté à la création d'un objet Supermarket.
- R5b : On crée une classe à part DiscountPlan pour gérer les plans de réductions. L'ajout d'une réduction au supermarché créera ce nouveau plan de réduction. Les clients pourront alors souscrire aux plan de réduction du supermarché en question. De plus les plan d'un supermarché sont stockés dans une map (nom, réduction) ce qui implique l'unicité des plans de réduction du supermarché.
- R6 : La pricing policies peut être modifiées directement sur chaque item du supermarché. De plus a été ajouté une fonctionnalité permettant de modifier toutes les pricing policies d'une catégorie d'item. Les catégories d'items sont soit Vegetables(fruits et légumes) soit Other.
- R6b : La pricing policy est définie à la création de l'item et elle peut ensuite être modifiée, entre 0 et 1 (0% de réduction à 100 % de réduction).
- R7 : Lors du checkout du client il lui est possible de choisir de se faire livrer ou non, si il le choisit le prix sera calculé en conséquence en tenant compte de son plan de réduction.
- R8 : on décide de choisir d'ajouter le poids\*la distance arrondi au kilomètre inférieur auquel on appliquera deliveryDiscount, le paramètre choisit lors de la création du plan de réduction.
- R8b : Par défaut les plan de réduction prime et platinum ont un deliveryDiscount de 0.5 et 1 respectivement.

## 3 Utilisation des Patterns de Conception

Dans notre système de supermarché, plusieurs patterns de conception sont appliqués pour structurer efficacement le code et améliorer la maintenabilité du système. Nous avons utilisé les patterns **Information Expert**, **Controller** et **Strategy**, pour implémenter nos décisions de conception expliquées par la partie précédente ou imposées par le sujet.

Le pattern **Information Expert** est utilisé, il attribue la responsabilité d'une tâche à la classe qui possède le plus d'informations pour l'accomplir efficacement. La classe **Customer** est responsable de la gestion de ses propres achats (**currentPurchase**) et contient la méthode **addItemToPurchase()**, évitant ainsi un couplage excessif avec d'autres classes.

Le pattern **Controller** est utilisé pour gérer les interactions entre les composants du système de paiement. La classe **Supermarket** joue ce rôle en servant de point central pour gérer les clients, les articles et les transactions. Cela permet de séparer la gestion du supermarché et la gestion des paiements. L'utilisation des classes **CashRegister**, **POS** et **TAS** permettent ensuite de décomposer le processus complexe de paiement en interface avec la Banque.

Un pattern **Strategy** est utilisé pour la gestion des plans de réduction (**DiscountPlan**). Plutôt que d'intégrer directement des réductions fixes dans les calculs, la classe **Customer** se base sur un **DiscountPlan** qu'elle gardera en attribut, ce qui permet d'ajouter facilement de nouvelles stratégies de tarification (par exemple, des réductions spéciales pour certains clients ou des promotions saisonnières) sans modifier le code existant. D'une manière similaire pour le pricing policies des items qui sont contenues dans la classe **Item**.

## 4 Analyse du Design

Notre design présente plusieurs points positifs :

- **Modularité** : Tous les composants sont bien séparés, ce qui rend le code plus clair et plus maintenable. Par exemple, les cartes bancaires sont indépendantes des comptes, qui eux-mêmes sont séparés de la banque. De la même manière, les réductions (**DiscountPlan**) sont gérées indépendamment des articles, ce qui permet d'ajouter facilement de nouvelles fonctionnalités sans impacter le reste du système.
- **Facilité de test** : Nous avons rendu certaines méthodes et attributs publics pour simplifier l'écriture des tests unitaires et vérifier que chaque classe fonctionne correctement.
- **Extensibilité** : Le système a été conçu de manière exhaustive avec des classes dédiées à chaque fonctionnalité, ce qui permettrait d'ajouter facilement de nouvelles options comme un vrai système d'inventaire ou la gestion d'autres types de réductions.

Cependant, il a aussi plusieurs limites :

- Nous n'avons pas implémenté de système d'inventaire, c'est-à-dire que on implémente seulement la présence des Item dans le supermarché sans prendre en compte leur nombre et sans en enlever quand les clients en prennent.
- Nous avons aussi seulement implémenté le paiement par carte.
- Pour les différents plans de réduction, nous avons implémenté un pourcentage de réduction ce qui pourrait être diversifié.  $x^2$
- De plus, notre implémentation ne nous permet pas de rajouter des catégories d'items durant un scénario (nous n'en avons implémenté que deux comme exemple). Nous aurions aussi pu implémenter un prix au poids pour les fruits et légumes par exemple.

## 5 Diagrammes UML

[Figure 1] représente le diagramme UML de classe du système. Dans une question de simplification les Getter et Setter ne sont pas visibles et la publicité des attributs et des méthodes non plus. [Figure 2] représente le diagramme de séquence d'un cas d'utilisation du système par un client.

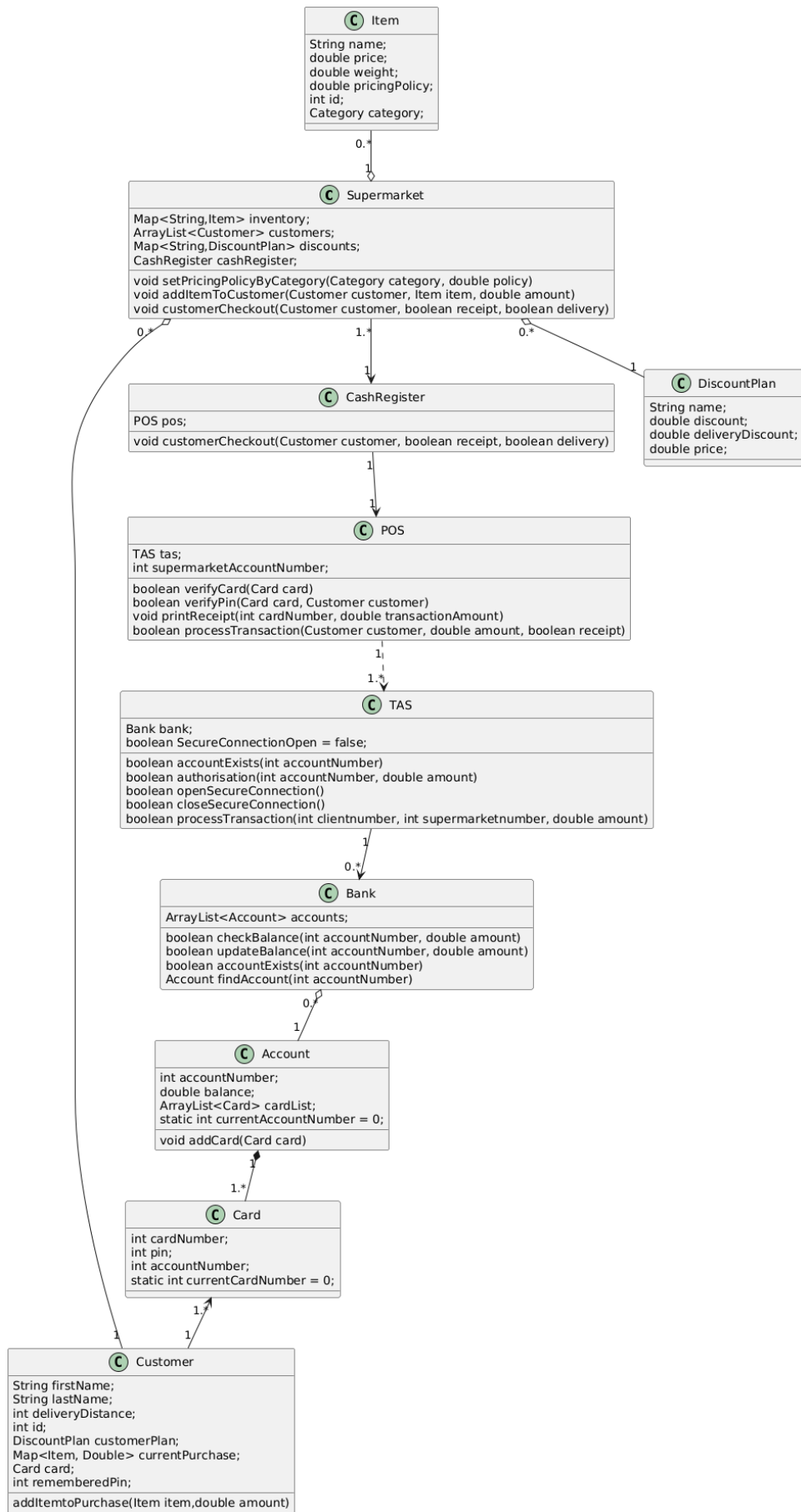


FIGURE 1 – Diagramme de classes

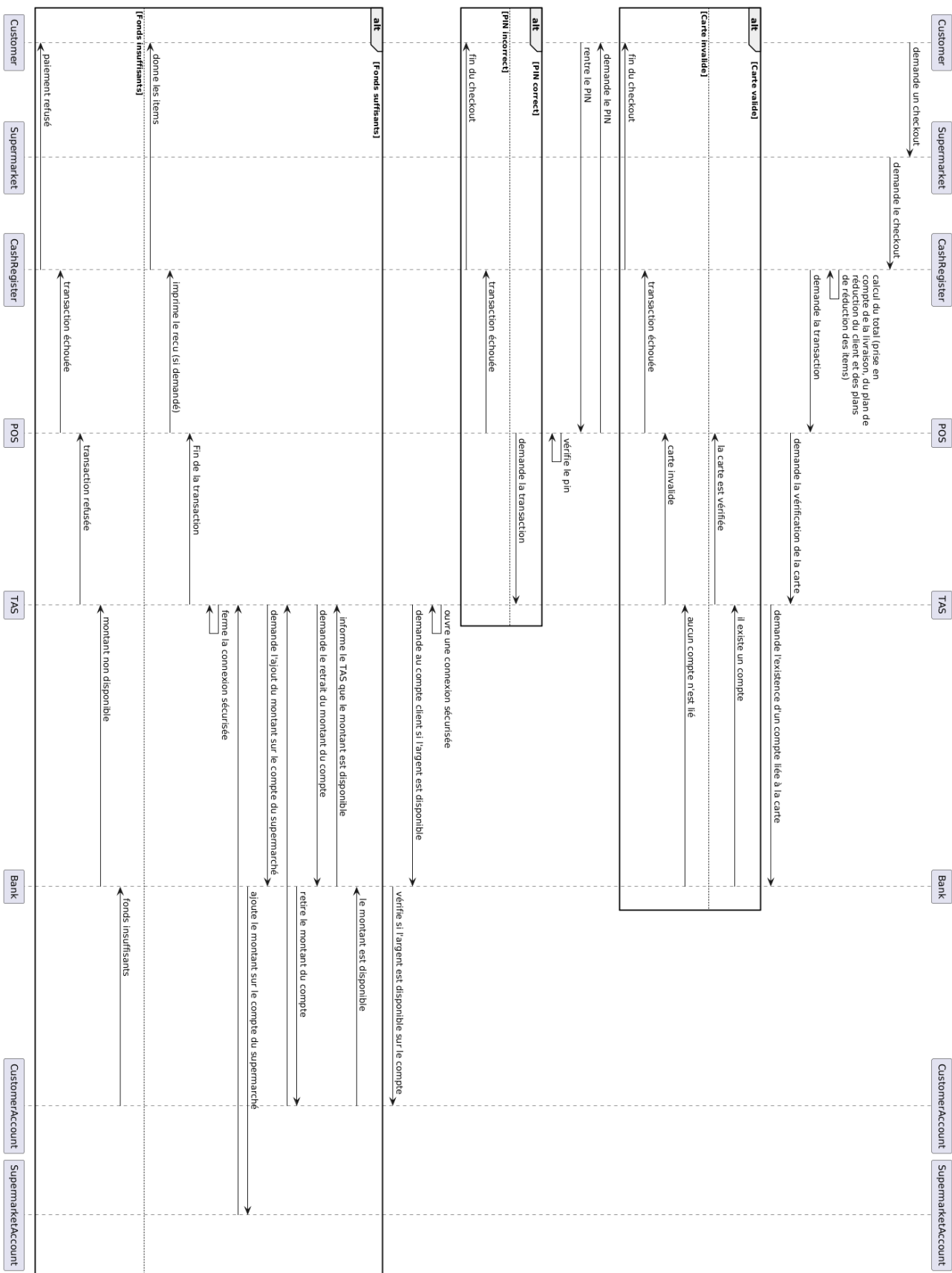


FIGURE 2 – Diagramme de séquence de l'utilisation du système

## 6 Scénarios de Test

Le scénario de test de notre système de supermarché suit plusieurs étapes permettant de simuler un processus d'achat complet, depuis la création des comptes jusqu'à la finalisation du paiement.

1. **Création des comptes et des moyens de paiement** : Un compte bancaire est créé avec un solde initial, et une carte bancaire associée est générée. Ce compte est ensuite ajouté à la banque du système.
2. **Initialisation du système** : Un terminal de paiement est mis en place et associé à la banque pour gérer les transactions. Une caisse enregistreuse est également créée et liée au système de paiement.
3. **Création du supermarché et ajout des articles** : Un supermarché est instancié avec un inventaire d'articles contenant leurs prix et leurs catégories. Ces articles sont ajoutés au stock disponible.
4. **Ajout d'un client et définition de son plan de réduction** : Un client est créé avec un plan de réduction applicable à ses achats.
5. **Ajout d'articles au panier du client** : Le client sélectionne plusieurs articles, qui sont alors ajoutés à son panier avec les quantités souhaitées.
6. **Passage en caisse et vérification du paiement** : Le client procède au paiement. La carte est vérifiée, le code PIN est validé, et la transaction est autorisée en fonction du solde disponible sur le compte.

Le scénario de test est disponible dans le fichier Main. De plus, nous avons mis en place une campagne de tests unitaires dans le fichier SupermarketTest. Nous avons aussi effectué des tests relatifs à chaque requirement dans le fichier SupermarketRequirementTest.

## 7 Comment tester notre réalisation

Dans cette partie, nous allons détailler comment tester notre réalisation, c'est-à-dire les étapes à faire pour réaliser un scénario et comment tester les différents ajouts demandés.

### 7.1 Initialisation du Système

Pour commencer, on commence par créer le supermarché en créant une banque, un compte associé au supermarché, un POS, un TAS et un cashRegister :

- **Création d'une banque et d'un système de transaction (TAS)**

```
Bank bank = new Bank();
TAS tas = new TAS(bank);
Account supermarketAccount = new Account(initialBalance);
bank.addAccount(supermarketAccount);
POS pos = new POS(tas, supermarketAccount.getAccountNumber());
CashRegister cashRegister = new CashRegister(pos);
Supermarket supermarket = new Supermarket(cashRegister);
```

### 7.2 Gestion des Produits et des Clients

Une fois le supermarché initialisé, on peut commencer à ajouter des clients et des items :

- **Création et ajout d'un client** :

```
Account customerAccount = new Account(initialBalance);
bank.addAccount(customerAccount);
Card customerCard = new Card(Pin);
customerAccount.addCard(customerCard);
Customer customer = new Customer(Nom, Prenom, Distancetothesupermarket,
    customer.discounts.get(DiscountName) //customerPlan, customerCard, RememberedPin);
supermarket.addCustomer(customer);
```

- **Ajout d'un item au magasin** :

```
Item apple = new Item("Apple", prix, poids, pricingPolicy(entre 0 et 1), id, Category);
supermarket.addToInventory(apple);
```

### 7.3 Ajout de pricing policies, discountPlan,

- Ajout d'articles au panier et passage en caisse :  
`supermarket.addItemToCustomer(customer, item, numberofitems);`  
`supermarket.customerCheckout(customer, Wants a receipt(bool), delivery(bool));`
- Création de nouveaux customerPlan :  
`supermarket.addDiscount(name,discout(sur les articles ((entre 0 et 1),`  
`discount(sur la livraison ((entre 0 et 1), prix) ;`
- Mise en place de nouvelles pricing policies :  
`//pour un item`  
`supermarket.setPricingpolicy(item,policy(entre 0 et 1));`  
`//pour une catégorie`  
`supermarket.setPricingPolicyByCategory(category,policy(entre 0 et 1));`

## 8 Répartition du Travail

Le tableau suivant indique la répartition des tâches entre les membres de l'équipe :

Membre	Design	Code	Tests
Merlin Baviere	Diagramme de classes	Gestion des paiements	Tests unitaires JUnit
Quentin Vicens	Diagramme de séquence	Gestion du supermarché	Scénarios test

TABLE 1 – Répartition des tâches dans le projet