**Département Mathématiques et Informatique**

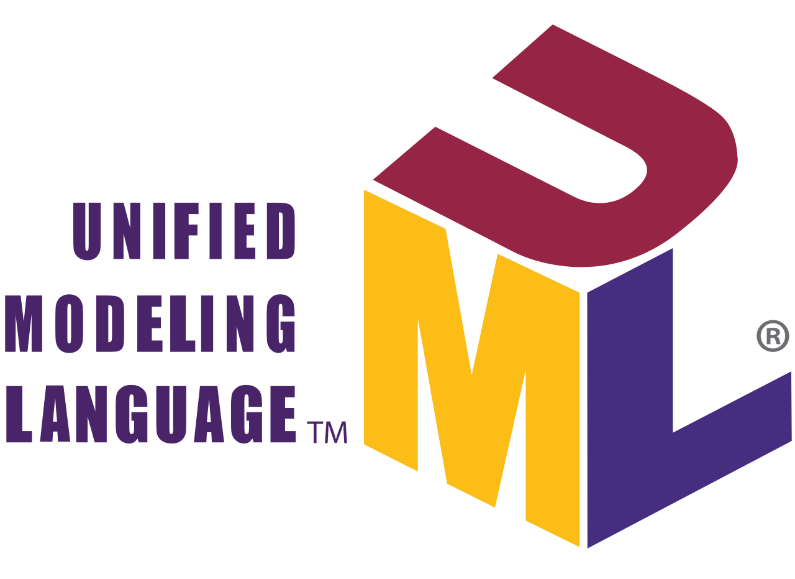
**Filière :**

**« Ingénierie Informatique : Big Data et Cloud Computing »**

**II-BDCC**

**Rapport :**

**Conception de l’application de gestion de magasin**



**CHAFI Khadija**

**DADI Anas**

**EL ASSIMI Ahmed**

**EL BADAOUI Mohammed**

**KHAIFALLAH Houda**

**NAIT CHERIF Ayman**

**OUFTOU Mohammed**

**OULAHYANE Kaoutar**

**SABAR Laila**

**Année Universitaire : 2019-2020**

Sommaire

[Introduction 4](#_Toc29132862)

[Partie I : Ingénierie et démarches 5](file:///C:\Users\khadi\Desktop\magasin.docx#_Toc29132863)

[I. Génie logiciel : 6](#_Toc29132864)

[II. Qualité logiciel : 6](#_Toc29132865)

[III. Activité et processus : 7](#_Toc29132866)

[IV. Les cycles de vie d’un logiciel : 8](#_Toc29132867)

[V. Méthode agile Scrum : 11](#_Toc29132868)

[Partie II : Architecture du projet 14](file:///C:\Users\khadi\Desktop\magasin.docx#_Toc29132869)

[I. Architecture : 15](#_Toc29132870)

[II. Le design pattern : 15](#_Toc29132871)

[Partie III : Réalisation du projet 17](file:///C:\Users\khadi\Desktop\magasin.docx#_Toc29132872)

[I. Diagramme de contexte : 18](#_Toc29132873)

[II. Diagramme de package : 18](#_Toc29132874)

[III. Module de gestion des catégories : 19](#_Toc29132875)

[A. Specification des besoins : 19](#_Toc29132876)

[B. Analyse des besoins : 20](#_Toc29132877)

[C. Conception de système de gestion des catégories : 23](#_Toc29132878)

[IV. Module de gestion des produits : 28](#_Toc29132879)

[A. Spécification de besoins : 28](#_Toc29132880)

[B. Analyse des besoins : 31](#_Toc29132881)

[C. Conception du système de gestion des produits : 32](#_Toc29132882)

[V. Module de gestion des clients : 34](#_Toc29132883)

[A. Spécification de besoins : 34](#_Toc29132884)

[B. Analyse de besoins : 35](#_Toc29132885)

[C. Conception du système de gestion des clients : 37](#_Toc29132886)

[VI. Module de gestion des ventes : 42](#_Toc29132887)

[A. Spécification de besoins : 42](#_Toc29132888)

[B. Analyse de besoins : 43](#_Toc29132889)

[C. Conception du système de gestion des ventes : 45](#_Toc29132890)

[VII. Module de gestion des paiements : 49](#_Toc29132891)

[A. Spécification de besoins : 49](#_Toc29132892)

[B. Analyse de besoins : 52](#_Toc29132893)

[C. Conception du système de gestion des paiements : 53](#_Toc29132894)

[Conclusion 57](#_Toc29132895)

[Table des figures 58](#_Toc29132896)

# Introduction

L’apparition des nouvelles technologies (WEB…) et l’augmentation des données à gérer, rend le processus de gestion en utilisant des feuilles Excel ou des outils non adaptés aux besoins inefficace. En effet, divers problèmes sont rencontrés vu que le maintien d’un inventaire précis dans Excel prend du temps, engendre des redondances et entraîne des erreurs.

De plus, avec la concurrence acharnée entre les entreprises l’entreprise se doit d’améliorer leurs processus de gestion.

Effectivement, ce logiciel de gestion de magasin permettra non seulement à l’entreprise de gérer les stocks afin d’éviter les ruptures ou le surstockage mais aussi donnera la possibilité de gérer les clients, organiser le processus de vente et de la prise des commandes jusqu’à la facturation et enfin suivre les reliquats des commandes.

Evidemment, ce logiciel doit être de qualité. Pour y parvenir, il faut prendre en considération certain éléments et suivre des démarches bien précises que nous allons introduire dans la section suivante.

Partie I

# Partie I : Ingénierie et démarches

Introduction

Les estimations de coût et de délai de production se révèlent inférieures à la réalité en travaillant de manière aléatoire et indisciplinée. Sans méthodes et démarches à suivre, les équipes de programmeurs sont confrontées à des problèmes de communication que l’encadrement peine à résoudre. Par ailleurs, le logiciel produit ne répond pas toujours aux attentes ; il faut dire que les spécifications ne capturent pas toujours les besoins de manière adéquate, faute d’outils et de méthodes adaptés.

Or, La tâche de maintenance, qui devrait normalement consister à adapter le logiciel à l’évolution des besoins, vise surtout à corriger ses fautes, sans toujours éviter d’en introduire de nouvelles.

Finalement afin de résoudre ces problèmes majeurs, un nouveau concept a été introduit : **Le concept de génie logiciel.**

## Génie logiciel :

Le terme génie logiciel (en anglais software engineering) désigne l'ensemble des méthodes, des techniques et outils concourant à la production d'un logiciel, au-delà de la seule activité de programmation.

Le choix du terme « génie » fait directement référence à celui du génie civil, désignant l'art de la construction. En effet, pour construire un ouvrage architectural, le seul fait de poser brique et ciment ne suffit pas. La construction d'un bâtiment est un tout, comprenant des activités de conception architecturale, de maçonnerie, de plomberie et d'électricité devant être coordonnées afin d'obtenir une maîtrise des délais et des budgets.

Le génie logiciel comporte donc des aspects de gestion de projet afin de produire un logiciel dans les délais prévus, avec un budget maîtrisé et donnant satisfaction au client

## Qualité logiciel :

Le génie logiciel procède en plusieurs étapes dans la fabrication d'un programme informatique. Ceci permet de garantir la qualité du produit et le respect des besoins de l'utilisateur tout en respectant les délais et les coûts fixés au départ.

Après plusieurs travaux, les chercheurs ont défini la qualité d'un logiciel à partir de plusieurs facteurs, notamment:

* **La validité :** c'est la capacité du logiciel à remplir exactement les tâches énoncées lors de sa spécification.
* **La fiabilité et la robustesse :** c'est l'aptitude du logiciel à fonctionner en continu et même dans des conditions anormales.
* **La compatibilité :** le logiciel doit pouvoir être combiné très facilement à d'autres logiciels.
* **La réutilisabilité :** l'application doit pouvoir être utilisée dans son entièreté ou en partie dans un autre projet.
* **L'efficacité :** le logiciel doit pouvoir utiliser efficacement les ressources matérielles.
* **L'extensibilité :** le logiciel doit être facile à maintenir, il doit pouvoir accepter l'ajout de nouvelles fonctionnalités.
* **La portabilité :** le logiciel doit pouvoir s'exécuter sous plusieurs environnements différents.
* **L'intégrité :** le logiciel doit pouvoir protéger efficacement son code entier ou une partie de son code contre les accès non autorisés.

Ainsi, afin d’assurer une qualité logicielle, il est nécessaire de maitriser le processus de réalisation du logiciel. Afin d’y parvenir, il faut :

* Formaliser les exigences des clients
* Identifier l’objectif à atteindre
* Décrire le processus (décrire l’activité ou l’ensemble d'activités qui utilise des ressources pour convertir des éléments d'entrée en éléments de sortie possédant une valeur ajoutée)
* Définir les rôles, les missions et les objectifs de chacun
* Mesurer et contrôler les actions de chacun

Finalement la qualité du processus de développement est la garantie de la qualité du produit.

## Activité et processus :

Un processus de développement est réalisé en se basant sur le concept de diviser pour mieux régner. Il désigne l’ensemble d’activités coordonnées et régulées, en partie ordonnées, dont le but est de créer un produit.

Un processus décrit deux choses importantes :

* **Les activités (étapes ou phases) (quoi ?)**
* **L’enchainement des activités (quand ?)**

Alors qu’une activité d’un processus est souvent décrite en termes de :

* **Entrées de l’activité (matière première)**
* **Sorties de l’activité (résultat)**
* **Intervenants et rôles (qui)**
* **Description de l’activité (quoi – quel est le problème à traiter ?)**

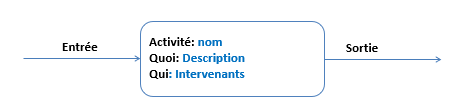


Figure 1 : activité d'un processus

Dans le cadre des projets de développement logiciel les activités sont communes.

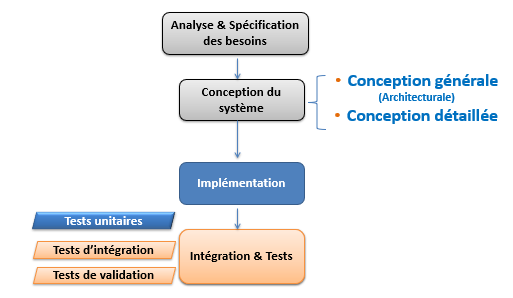


Figure 2: activités d’un processus de développement logiciel

Finalement, ces activités et processus ont donné naissance au cycle de vie d’un logiciel.

## Les cycles de vie d’un logiciel :

Afin de mettre en place un logiciel de qualité, il faut bien définir un processus de développement de qualité qui permet d’organiser les activités, guider les développeurs et fournir les moyens de développement et de maintenance pour répondre aux besoins instables du client.

Dans ce sens plusieurs cycle de vie sont proposée, dont on distingue :

1. **Cycle de vie en cascade :** ce cycle de vie est organisé séquentiellement.

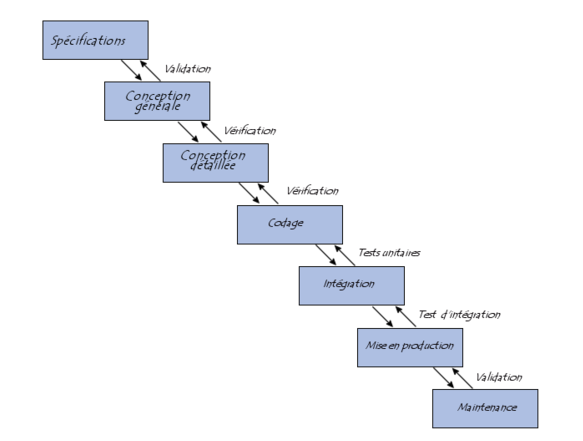


Figure 3: cycle de vie en cascade

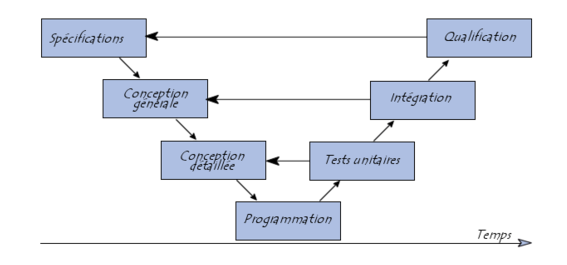
1. **Cycle de vie en V :** le modèle en V part du principe que les tests de conformité du logiciel doivent se faire dès la phase de conception.

Figure 4: cycle de vie en V

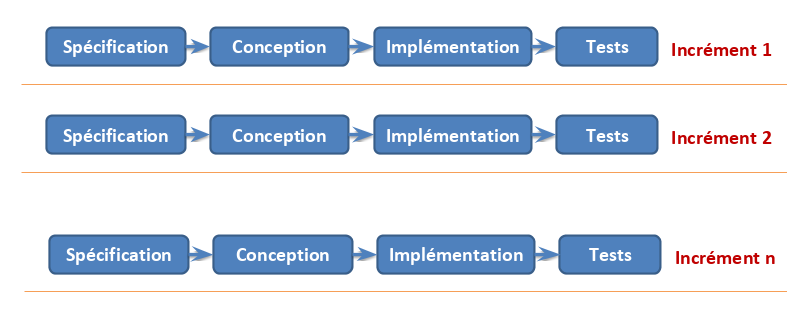
1. **Cycle de vie incrémentale :** qui se base sur l’approche « diviser pour régner » dont chaque incrément constitue un composent indépendant du logiciel.

Figure 5 : Cycle de vie incrémentale

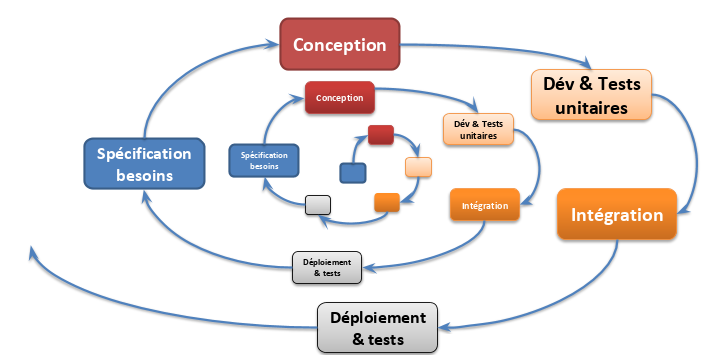
1. **Cycle de vie en spirale :** Il consiste à coder un peu (prototype), à tester puis à recommencer.

Figure 6: Cycle de vie en spirale

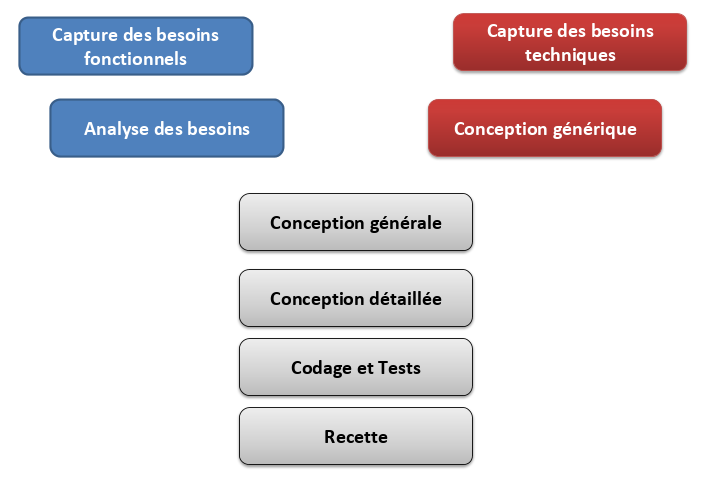
1. **Cycle de vie en Y :** qui s’articule autour de trois branches :
   1. Une branche technique : identifier les outils et les composants techniques à utiliser
   2. Une branche fonctionnelle : identifier les exigences fonctionnelles
   3. Une branche de conception et réalisation

Figure 7 : cycle de vie en Y

Selon la petite taille de logiciel de gestion de magasin qu’on vise à développer, on a choisi d’adopter un cycle de vie en Y dû à son pilotage de risque, sa nature itératif et incrémentale ainsi la simplicité de son intégration avec la méthode agile « Scrum » qui sera introduit dans la prochaine section.

## Méthode agile Scrum :

Dans le cadre de notre projet et afin d’assurer le bon déroulement des différentes phases de ce dernier, nous avons opté pour la méthode agile Scrum pour la conception et le développement de notre système pour des raisons bien déterminées, en effet le processus Scrum s’adapte parfaitement à la décomposition de notre projet, il se base sur les avantages suivants :

* Plus de souplesse et de réactivité,
* Grande capacité d’adaptation pour que l'application implémentée puisse répondre aux nouvelles exigences pendant le développement,
* Satisfaire au mieux les besoins du client,
* Itératif et incrémental, le projet sera divisé en unités incrémentales appelées itérations. Le temps de développement de chaque itération est faible (deux semaines en moyenne), fixé et strictement respecté,

Cette méthodologie agile consiste à réaliser des fonctionnalités par itération en incluant la participation du client. Chaque itération peut durer de deux à quatre semaines, à la fin de chaque sprint un produit fonctionnel doit être livré. En effet, Scrum définit trois rôles principaux :

1. **Le Product Owner** : qui est responsable du produit et représente les utilisateurs finaux, son rôle est de :

* Définir la liste des fonctionnalités du produit.
* Prioriser les fonctionnalités en fonction de leur importance et leur valeur ajoutée.
* Choisir la date de livraison des versions ainsi que leurs contenus.
* Valider les lots livrés avec l’équipe de développement
* Clarifier les besoins à l’équipe de développement si nécessaire

1. **Le Scrum master** assure les tâches suivantes :

* S’assurer que Scrum est bien appliquée et respectée.
* Encourager l’équipe à apprendre et à progresser pour qu’elle soit fonctionnelle, productive et créative durant le projet.
* Eliminer les obstacles pouvant perturber la progression du travail.

1. **L’équipe du projet** contient 9 éléments. Elle regroupe tous les rôles nécessaires au projet, il y a ceux qui se chargent de la conception, d’autres pour le test, et le reste pour le développement de l’application. Le rôle principal de l’équipe est :

* Transformer les besoins exprimés en fonctionnalités utilisables.
* Livrer régulièrement une version fonctionnelle du produit.

La figure ci-dessous présente le déroulement de la gestion de projet par Scrum.

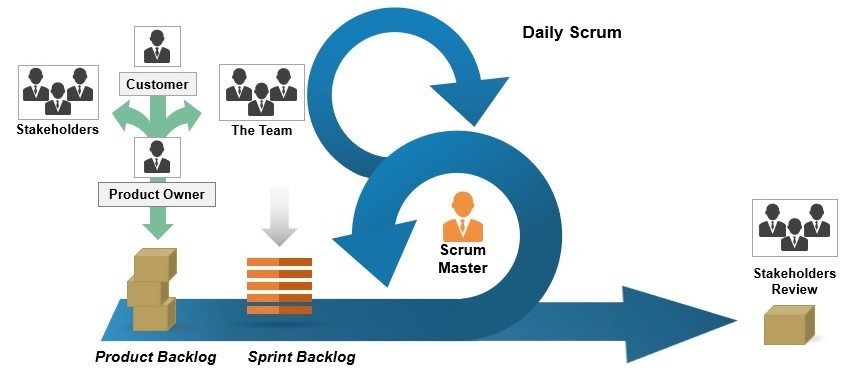


Figure 8 : méthode agile SCRUM

1. **Backlog produit** (ou catalogue des besoins)

Elaboration d’une application de gestion de magasin qui se compose des modules suivants :

* Gestion des produits
* Gestion des clients
* Gestion des ventes
* Gestion des paiements

1. **Sprint Backlog :** A chaque fois on doit se focaliser sur une partie du backlog qui est le sprint, pour notre cas on va commencer par le module de gestion des produits, après la fin du sprint en passe au module suivant, et ainsi de suite jusqu’à réalisation du dernier module.
2. **Sprint** (itération)**:**

* Développement des fonctionnalités du Backlog de sprint.
* Aucune modification du Backlog de sprint possible.
* Mêlée quotidienne (Rencontre quotidienne)
* Point de contrôle quotidien de l’équipe.

1. **Produit livrable :** livrer au Product Owner à la fin du sprint le module réalisé.

Partie II

# Partie II : Architecture du projet

## Architecture :

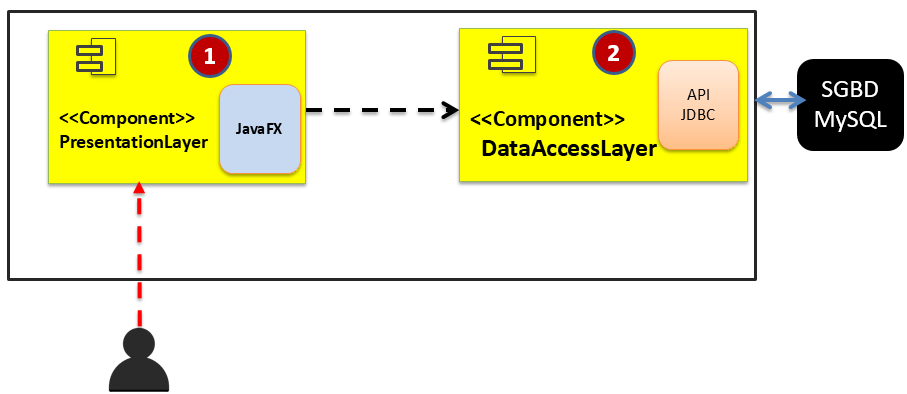
****Pour réaliser chaque module, nous allons adopter l’architecture **à 2 couches** ci-dessous qui présente de nombreux avantages, notamment la vitesse de développement, **l'évolutivité**, les **performances** et la **disponibilité**. Ainsi, en séparant les différentes couches nous pouvons les mettre à l'échelle indépendamment en fonction des besoins à tout moment. Du surcroît, la modularisation des différents niveaux de l’application donne à notre équipe de développement la possibilité de développer et d'améliorer un produit plus rapidement que de développer une base de code unique car une couche spécifique peut être mise à niveau avec un impact minimal sur les autres couches. Il peut également aider à améliorer l'efficacité du développement en permettant à notre équipe de se concentrer sur leurs compétences de base.

Figure 9: Architecture de l'application de gestion de magasin

Où **PresentationLayer** représente la partie IHM (Interface-Homme-Machine) de l’application. Pour réaliser cette partie nous allons utiliser **JAVAFX**.

Et **DataAccessLayer** représente la partie d’accès aux données. Pour réaliser cette partie nous allons utiliser le design pattern **DAO** (Data Access Object) qui constitue une bonne pratique d’accès aux données.

## Le design pattern :

En effet, **DAO** est un modèle qui sépare la logique métier de haut niveau des opérations d'accès aux données. DAO nous permet d'écrire le code pour travailler avec les données de la base de données. Il réduit la complexité de notre code par rapport à l'écriture conjointe de la logique métier et des opérations d'accès aux données. Le pattern Data Access Object peut avoir les participants suivants :

* Une **interface** : une interface DAO qui peut déclarer les opérations importantes requises pour s’effectuer sur des objets de modèle.
* Une **classe** : une classe concrète DAOIIMPL qui peut implémenter l'interface DAO ou définir les opérations requises pour s’effectuer sur des objets de modèle. Cette classe joue le rôle principal pour obtenir les données de la source de données.
* **Objet de modèle** : l'objet de modèle peut être appelé une classe POJO (Plain old Java object ou bon vieil objet Java en français) qui a les membres de données et leurs méthodes setter / getter pour stocker les données récupérées via la classe DAOIIMP. Dans notre projets les objets de modèle sont représentés par les classes : Produit, Catégorie, Client, Vente, ligne Commande et paiement.

Partie III

# Partie III : Réalisation du projet

## Diagramme de contexte :

Le diagramme de contexte est un diagramme UML qui constitue une étape intermédiaire entre le cahier des charges et la construction des premiers cas d’utilisation. Il présente le système à modéliser, en général sous la forme d’une **boîte noire** et les différents acteurs qui interagissent avec ce système.

Dans notre cas, le système de gestion du magasin, a principalement quatre acteurs, dont trois acteurs principaux : le magasinier, le vendeur et le caissier, de plus d’un serveur des paiements comme un acteur secondaire.

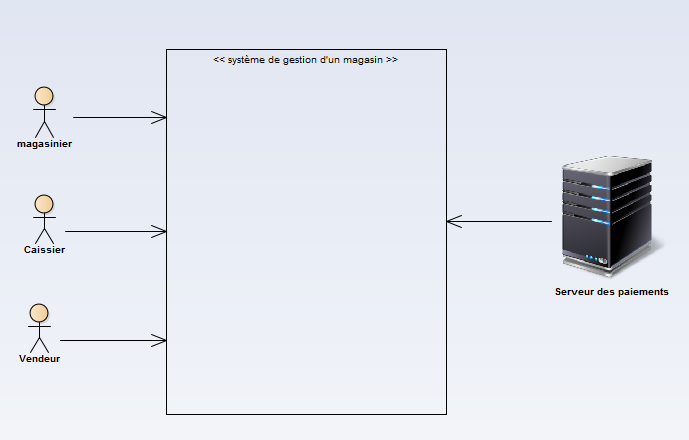


Figure 10: diagramme de contexte de gestion de magasin

## Diagramme de package :

Le diagramme de paquetages est un diagramme UML, qui donne une représentation graphique de haut niveau des relations existant entre les packages composant un système.

Notre système est constitué de cinq packages ; d’une part, la gestion des produits et la gestion des catégories seront faites par le magasinier, d’autre part, la gestion des clients et la gestion des ventes sont concernées par le vendeur, et pour la gestion des paiements, elle est associée à un caissier avec l’intervention d’un serveur des paiements.

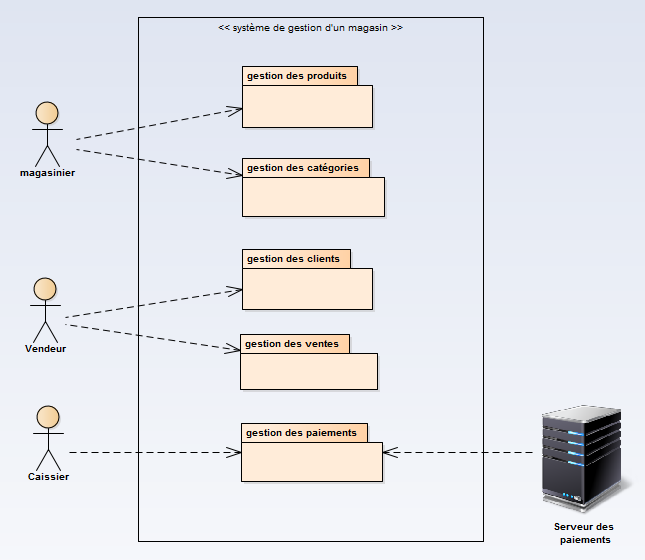


Figure 11: diagramme de package

## Module de gestion des catégories :

### Specification des besoins :

Le module de gestion des catégories doit permettre à ces utilisateurs d’ajouter de nouvelle catégorie à la base de données, de rechercher et afficher, de modifier et de supprimer des catégories. Une catégorie est définie par un nom et un id à générer automatiquement par le système Une catégorie peut avoir plusieurs produits.

1. Diagramme de cas d’utilisation :

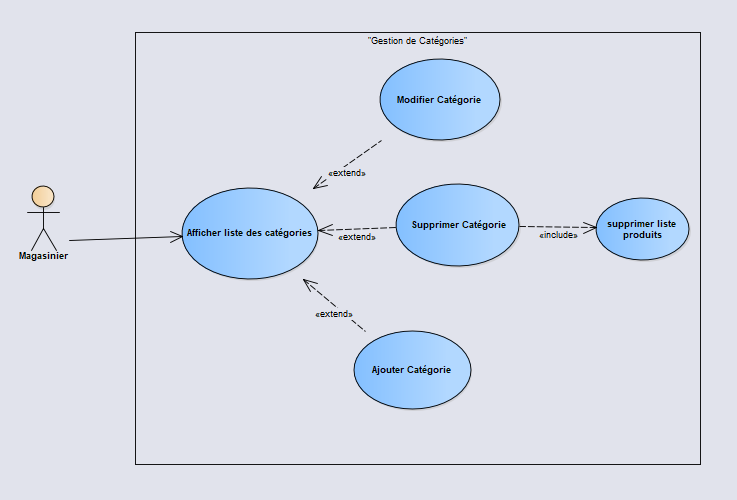


Figure 12: diagramme de cas d'utilisation pour la gestion des catégories

### Analyse des besoins :

Le diagramme de cas d’utilisation ci-dessus « gestion des catégories » présente les ensembles des cas d’utilisation. Dans ce module, le système doit donner la possibilité au magasinier d’afficher la liste des catégories et à la base de ce cas il a la possibilité de créer, modifier ou supprimer une catégorie. Pour le cas de suppression d’une catégorie il est évident de supprimer tous les produits associer à la catégorie que l’on souhaite supprimer.

1. Diagramme de classe métier :

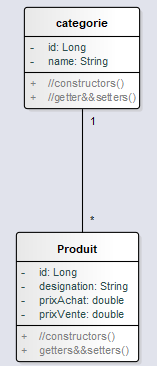


Figure 13: diagramme de classe pour la gestion des catégories

Le diagramme de classe ci-dessus représente les deux principales classes **Produit** et **Catégorie**, que nous aurons besoin pour réaliser ce module. Un produit peut appartenir à une et une seule catégorie, et une catégorie peut contenir plusieurs produits.

1. Diagramme de séquence boite noire :

Pour bien détailler le diagramme de cas d’utilisation nous allons présenter le diagramme de classe boite noire pour montrer comment nous allons réaliser les cas d’utilisation de ce module.

Nous allons présenter les 2 diagrammes de séquences

* Diagramme de séquence de cas d’utilisation afficher catégorie

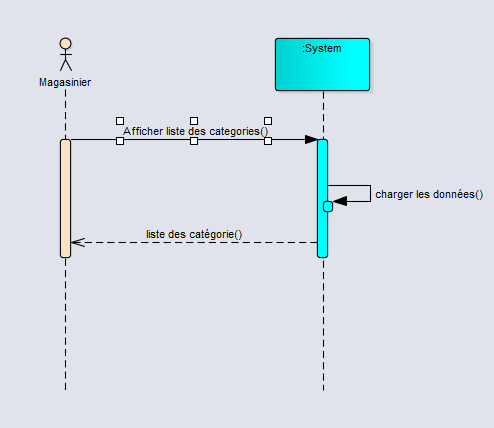


Figure 14: Diagramme de séquence de cas d’utilisation afficher catégorie

* Diagramme de séquence de cas d’utilisation Ajouter catégorie

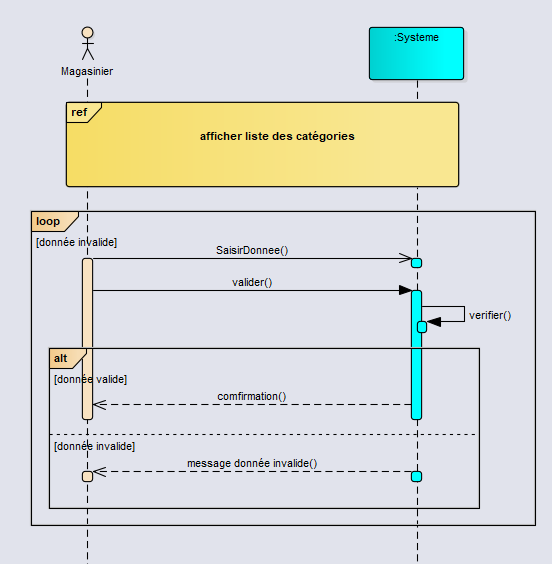


Figure 15 : Diagramme de séquence de cas d’utilisation Ajouter catégorie

### Conception de système de gestion des catégories :

Pour réussir à mettre en place un diagramme de conception détaillée nous allons le fragmenter en trois sous diagramme de classe :

* Diagramme de classe DAO (data access object)
* Diagramme de classe CategorieIHM (catégorie interaction homme-machine)
* Diagramme de classe CategorieDAO (catégorie data access object)

1. Diagramme de classe DAO :

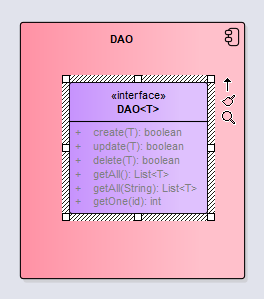


Figure 16: diagramme de classe DAO

Le diagramme ci-dessus représente l’interface de type générique T qui vas nous générer toutes les méthodes que nous aurons besoin au cour notre implémentation.

1. Diagramme de classe CategorieIHM :

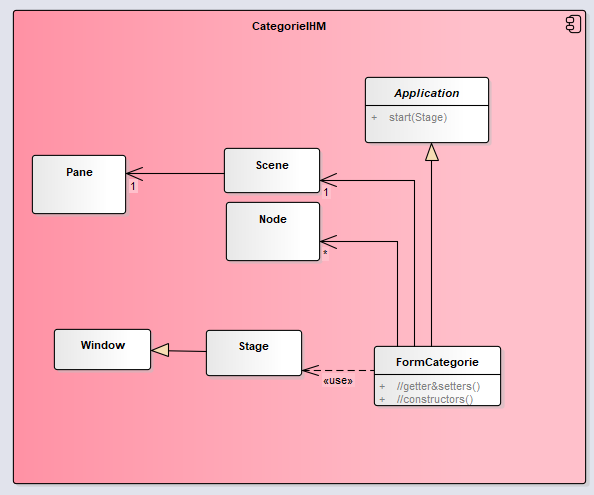


Figure 17: categorieIHM

Ce diagramme ci-dessus « Diagramme de classe CategorieIHM » représente l’ensemble des classes qui interagissent afin de réaliser une interface graphique facile à utiliser pour l’utilisateur. Nous avons utilisé les classes **Stage**, **Scene**, **Node** du **javafx**.

1. Diagramme de classe CategorieDAO :

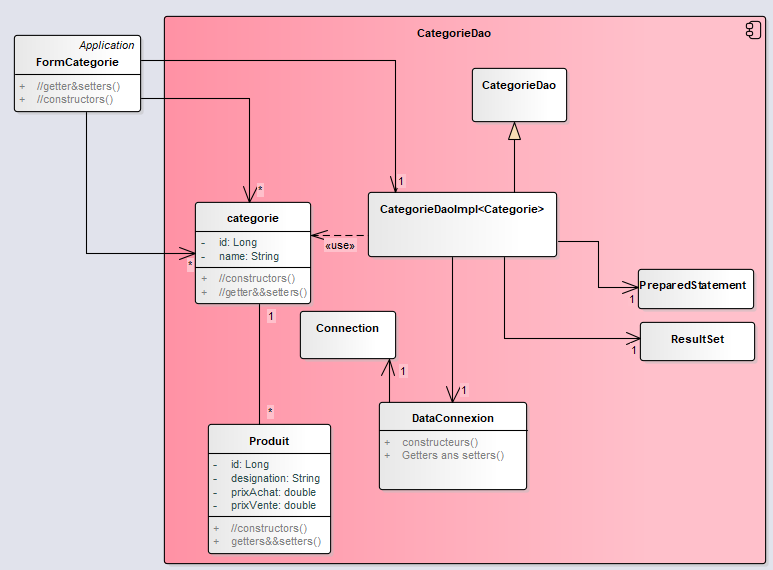


Figure 18: Diagramme de classe CategorieDAO

Le diagramme ci-dessus présente l’ensemble des classes qui interagissent pour montre comment se connecter à la base donnée en utilisant la classe **dataConnection** et aussi la relation entre la classe **FormCategrie** du composant **categorieIHM** avec la classe **catégorie** et **CategorieDaoImp** qui implémente l’interface **CategorieDao.**

1. Diagramme de composant :

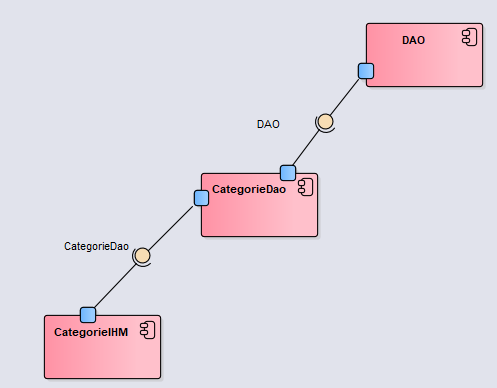


Figure 19 : diagramme de composant

Le diagramme de composant ci-dessus présente les différents composants du système de ce module, et montre les interactions. Le composant DAO fourni l’interface **DAO** au composant **CategorieDAO** qui fournit lui-même l’interface **CategorieDao** au composant **CategorieIHM.**

1. Diagrammes de séquences détaillés

Pour bien détailler et monter la communication entre les composants pour réaliser un cas, nous allons présenter les diagrammes de séquence boite blanche.

Nous allons présenter les 2 diagrammes de séquences

* Diagramme de séquence afficher catégorie boite blanche

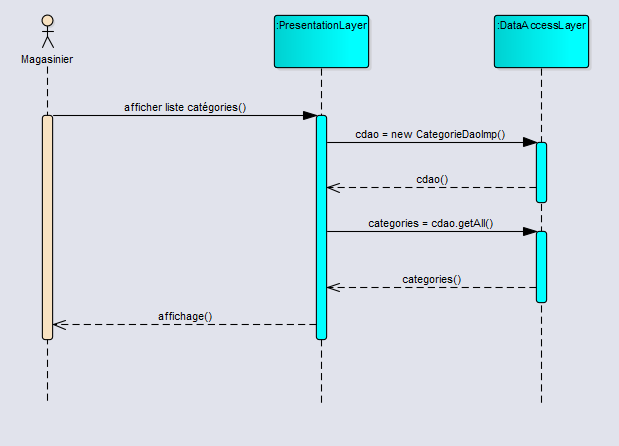


Figure 20: diagramme de séquence afficher catégorie boite blanche

* Diagramme de séquence Ajouter catégorie boite blanche

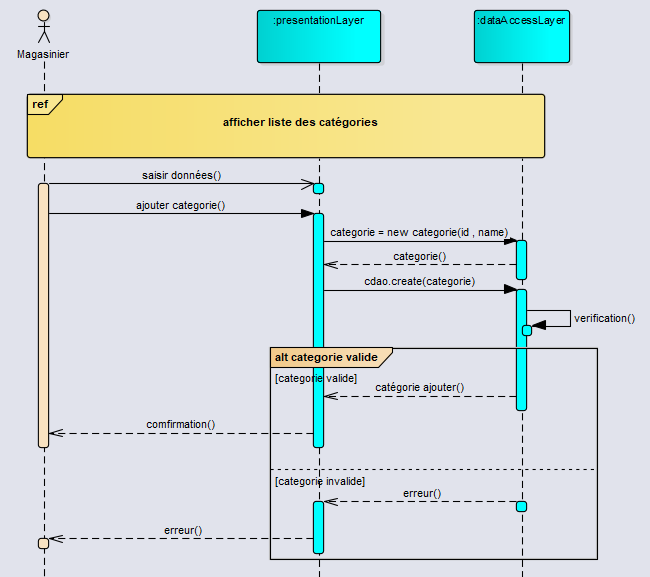


Figure 21:diagramme de séquence Ajouter catégorie boite blanche

## Module de gestion des produits :

Après la réalisation du module de gestion des catégories. Dans cette partie nous allons aborder en détails le module de gestion des produits. Pour réaliser ce module nous allons commencer par la définition les besoins fonctionnelles et techniques exprimés par le client ensuite nous allons modéliser ces besoins en utilisant le langage UML.

### Spécification de besoins :

Le module à concevoir est régi par une application qui fonctionnera sur un poste et sera accessible par le magasinier, qui en sera l'administrateur principal de ce module, alors après avoir rencontré le magasinier et comprendre ses attentes de notre système nous avons décidé de mettre en place un système de qualité qui répond aux besoins du client.

Notre application doit permettre aux utilisateurs de :

* Créer un produit
* Afficher les détails d’un produit
* Afficher la liste des produits
* Modifier un produit
* Supprimer un produit
* Rechercher un produit par sa référence
* Rechercher les produits par leur catégorie

Ce module est géré par le magasinier qui est le responsable principal de ce module. Pour sécuriser l’accès du magasinier a ce module nous allons adopter un système d’authentification.

1. Diagramme de cas d’utilisation :

**Acteur :** Le module de gestion des produits est géré par le magasinier.

**Description des cas d’utilisations :** les cas d’utilisation assuré par ce module sont :

* **L’authentification :** assure un accès sécuriser au système par un système de vérification du compte
* **Créer un produit :** ce cas permet d’ajouter des nouveaux produits à la base de données cela suite à un achat auprès du fournisseurs
* **Afficher les détails d’un produit**
* **Afficher la liste des produits**
* **Modifier un produit :** permet de modifier les informations d’un produit (sa désignation, son prix d’achat ou son prix de vente)
* **Supprimer un produit :** la suppression d’un produit de la base de données.
* **Rechercher un produit par sa référence**
* **Rechercher des produits par leur catégorie :** permet d’afficher les produits d’une catégorie choisie.

Le diagramme de cas d’utilisation ci-dessus visualise les fonctionnalités offertes par le module de gestion des produits :

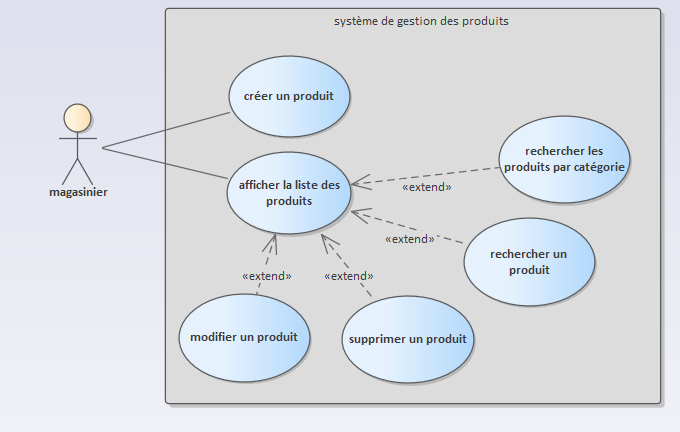


Figure 22: diagramme de cas d'utilisation de système de gestion des produits

1. Diagramme de séquence boite noire pour la création d’un nouveau produit :

Le diagramme de séquence boite noire visualise l’interaction d’un acteur avec le système pour notre cas l’acteur c’est le magasinier et le système c’est le module de gestion des produits.

Dans le module de gestion des produits le cas le plus important c’est la création d’un nouveau produit, le diagramme de séquence ci-dessus décrit en détails le processus de la création d’un nouveau produit.

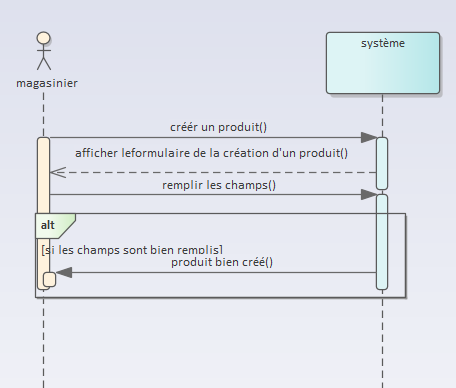


Figure 23: diagramme de séquence boite noire pour la création d'un produit

### Analyse des besoins :

Le module de gestion des produits doit permettre à ces utilisateurs d’ajouter de nouveaux produits à la base de données, de rechercher et afficher des produits, de modifier et de supprimer des produits. Un produit est défini par un id à générer automatiquement par le système, une désignation, un prix d’achat, un prix de vente et une catégorie. Cette catégorie est gérée par le module de gestion des catégories.

Le module de gestion des produits est lié au module de gestion des catégories :

* Chaque produit a une catégorie
* Chaque catégorie contient une liste des produits

Dans la phase d’analyse des besoins le plus important c’est la structure et le type des données géré par notre système et pour modéliser ces données nous allons nous baser sur les diagrammes structurés du langage UML

1. Diagramme de classe métier :

Le diagramme de classe métier de ce module se compose de deux classes principales une classe qui représente les produits et une classe qui représente les catégories.

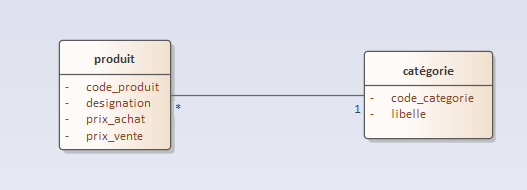


Figure 24: diagramme de classe métier de gestion des produits

### Conception du système de gestion des produits :

Pour la réalisation de ce module nous allons utiliser le design pattern DAO qui permet de séparer la couche modèle d'une application en deux sous-couches distinctes :

* Une couche qui gère les traitements métiers appliqués aux données.
* Une couche qui gère le stockage des données nommée couche de données. Il s'agit là des opérations classiques de stockage : la création, l’affichage, la modification, la suppression et la recherche.

1. Diagramme de composant :

Dans le même sens que le modèle design pattern DAO, notre diagramme de classe est composé d’une couche métier présentée par la classe produit et la classe catégorie et une couche de données contenant les interfaces DAO et la classe d’implémentation. Cela est représenté par le diagramme de composant suivant :

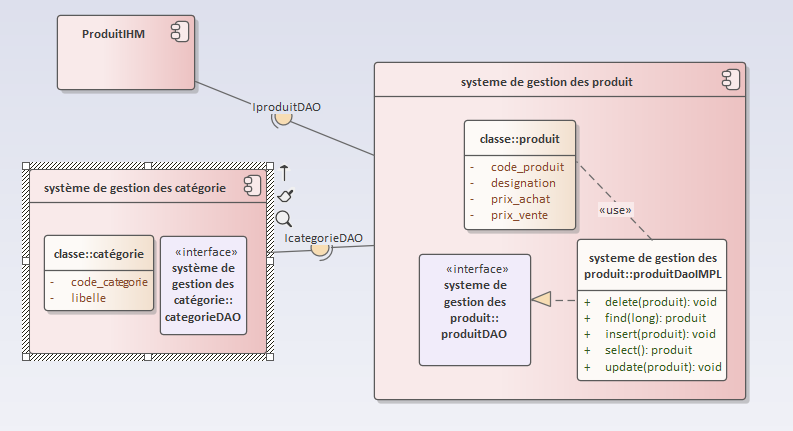


Figure 25: diagramme de composants de gestion des produits

1. Diagramme de séquence :

Pour bien détailler et monter la communication entre les composants pour réaliser un cas, nous allons présenter les diagrammes de séquence boite blanche.

Nous allons présenter le diagramme de séquence pour le cas d’utilisation créer un produit.

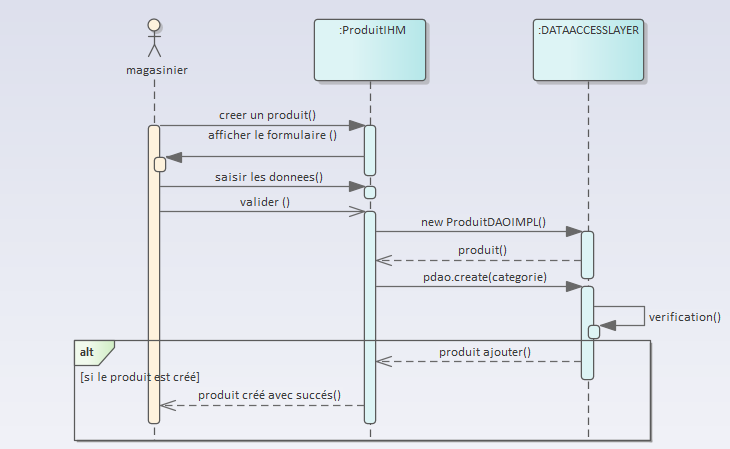


Figure 26: diagramme de séquence pour le cas créer un produit

## Module de gestion des clients :

### Spécification de besoins :

Le module de gestion des clients doit permettre à ces utilisateurs d’ajouter de nouveau client à la base de données, rechercher, afficher, modifier et de supprimer des clients. Un client est défini par un nom, prénom, téléphone, adresse et id à générer automatiquement par le système.

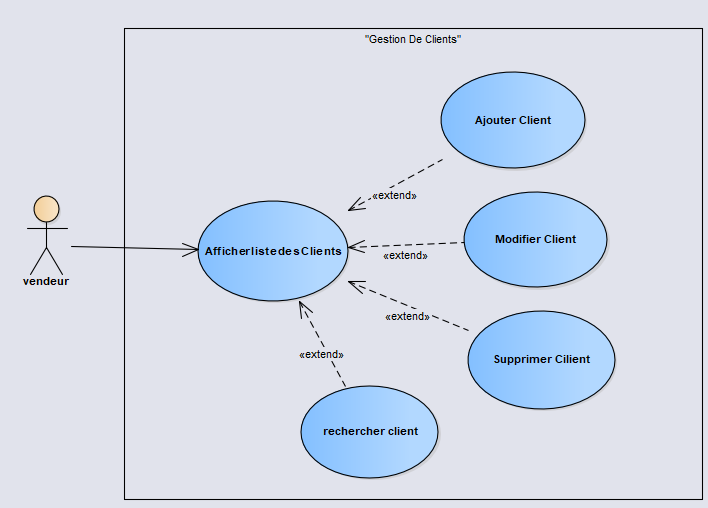


Figure 27: diagramme de cas d'utilisation pour la gestion des clients

Le diagramme de cas d’utilisation ci-dessus « gestions des clients » présentes les ensembles cas d’utilisation. Dans ce module, le système doit donner la possibilité au magasinier d’afficher un client et à la base de ce cas il a la possibilité de créer, supprimer ou ajouter un client.

### Analyse de besoins :

1. Diagramme de classe métier :



Figure 28 : diagramme de classe métier

Le diagramme de classe ci-dessus représente le principale classe **Client**, que l’on a besoin pour réaliser ce module. Un client est caractérisé par son id, nom, téléphone, adresse et aussi ses opérations.

1. Diagramme de séquence boite noire :

Pour bien détaillé le diagramme de cas d’utilisation nous allons présenter le diagramme de classe boite noire pour montrer comment nous allons réaliser les cas d’utilisation de ce module.

Nous allons présenter les 3 diagrammes de séquences

* Diagramme de séquence afficher client

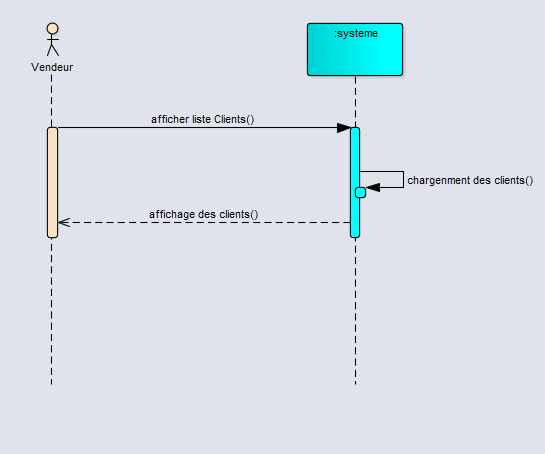


Figure 29: Diagramme de séquence afficher client

* Diagramme de séquence Ajouter client

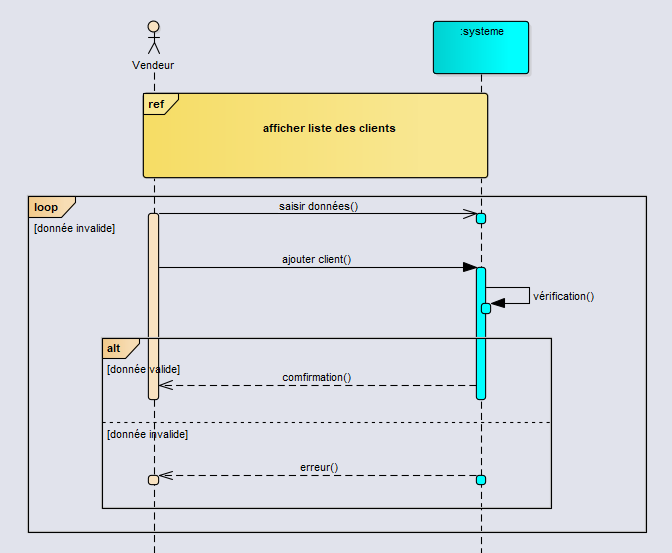


Figure 30: Diagramme de séquence Ajouter client

### Conception du système de gestion des clients :

Pour réussir à mettre en place un diagramme de conception détaillée complète nous allons le fragmenter en trois sous diagramme de classe :

* Diagramme de classe DAO (data Access Object)
* Diagramme de classe ClientIHM (client interaction homme-machine)
* Diagramme de classe ClientDAO (client data Access Object)

1. Diagramme de classe ClientIHM (client interaction homme-machine) :

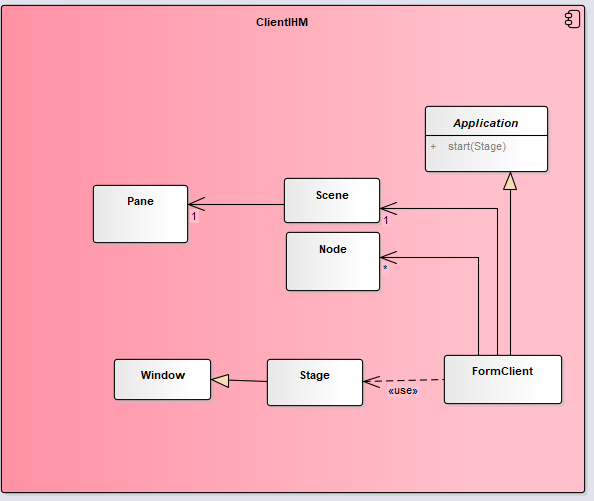


Figure 31: clientIHM

Ce diagramme ci-dessus « Diagramme de classe ClientIHM » représente l’ensemble des classes qui interagissent afin de réaliser une interface graphique facile à utiliser pour l’utilisateur. Nous avons utilisé les classes **Stage**, **Scene**, **Node** du **javafx**.

1. Diagramme de classe ClientDAO (client data Access Object) :

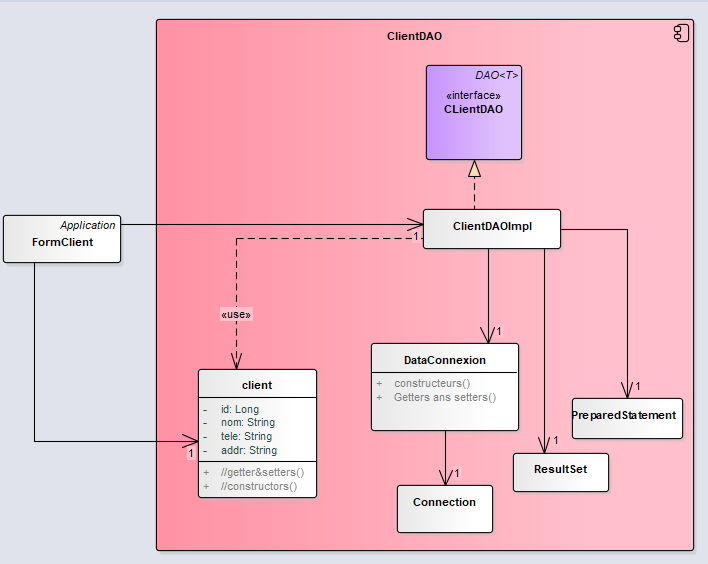


Figure 32: ClientDao

Le diagramme ci-dessus « Diagramme de classe ClientDAO » présente l’ensemble des classes qui interagissent pour montre comment se connecter à la base donnée utilisant la classe **dataConnection** à la fois et aussi la relation class **FormClient** du composant **ClientIHM** avec la classe **Client, ClientDaoImp** qui implémente l’interface **ClientDao.**

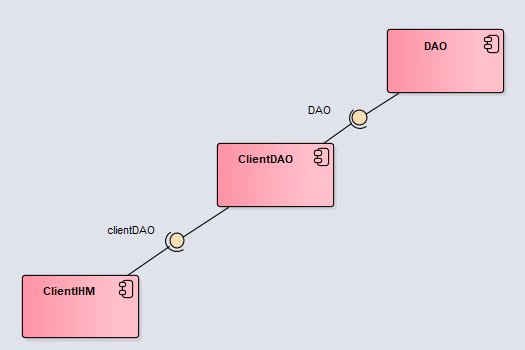


Figure 33: diagramme de composant pour la gestion des clients

Le diagramme de composant ci-dessus présente les déférents composants de système de ce module, et monte les interactions faites. Notamment le composant DAO fourni l’interface **DAO** a le composant **ClientDAO** qui fournit lui-même l’interface **ClientDao** au composant **ClientIHM.**

1. Diagrammes de séquences détaillés

Pour bien détaillé et monter la communication des composants pour réaliser un cas nous allons présenter les diagrammes de classe boite blanche.

Nous allons présenter les 2 diagrammes de séquences

* Diagramme de séquence afficher clients boite blanche

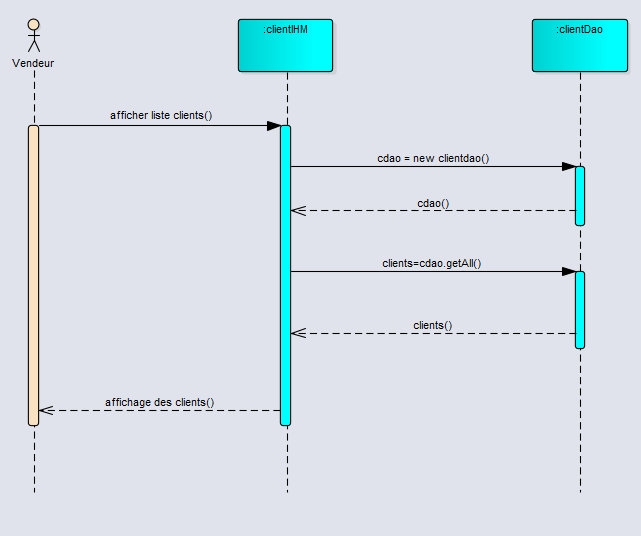


Figure 34:Diagramme de séquence afficher clients boite blanche

* Diagramme de séquence Ajouter client boite blanche

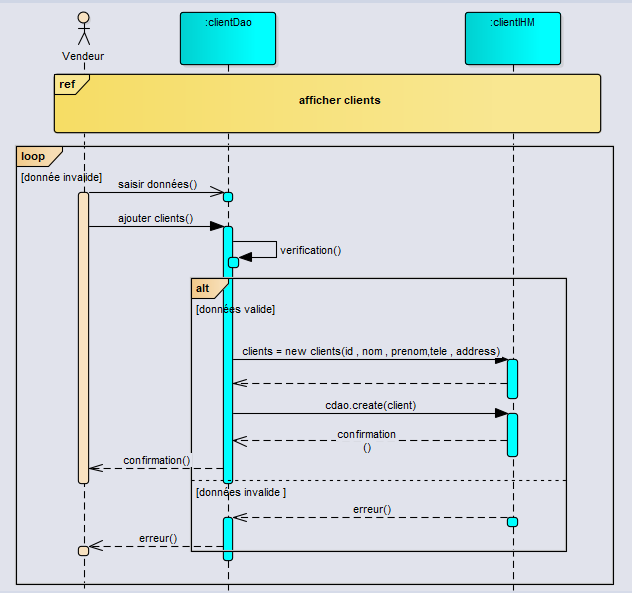


Figure 35:diagramme de séquence ajouter client

## Module de gestion des ventes :

### Spécification de besoins :

Durant cette activité nous allons définir les principaux cas d’utilisation à prendre en compte par notre application.

En effet, notre application doit permettre aux utilisateurs de ce module de :

* Créer une vente
* Afficher la liste des ventes
* Afficher le détail d’une vente
* Modifier une vente
* Supprimer une vente

1. **Diagramme de cas d’utilisation pour la gestion des ventes**

Le diagramme de cas d’utilisation ci-dessous « gestion des ventes » met en évidence les principaux cas d’utilisation de ce module. En occurrence, le système doit donner la possibilité au magasinier d’afficher, rechercher, ajouter, modifier et supprimer une vente. Dans la suite, nous allons essayer de donner plus des détails pour montrer quelles sont les données à utiliser pour réaliser ces cas. Ensuite, nous allons présenter des diagrammes de séquences pour montrer comment créer et supprimer une vente et mettre en évidence les parties du système qui vont interagir entre eux pour réaliser ce cas.

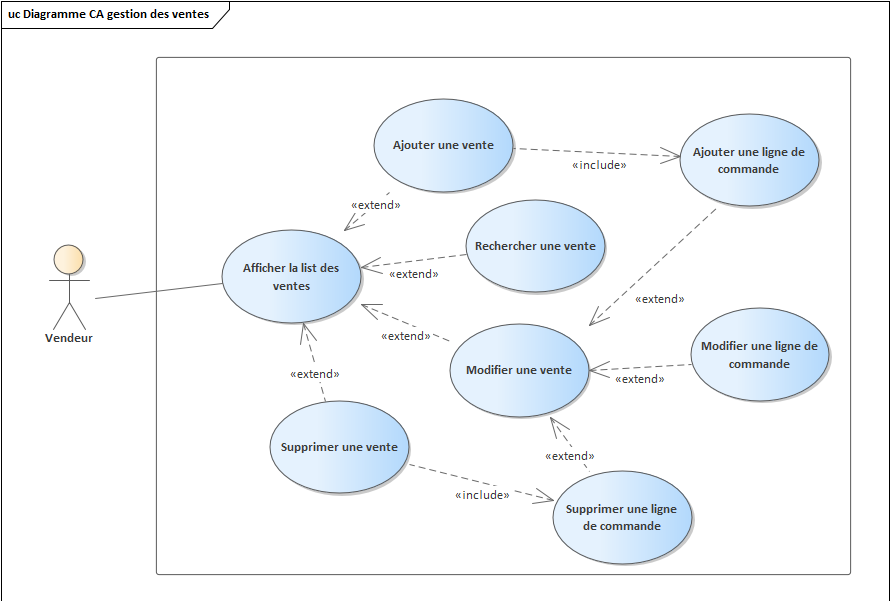


Figure 36: diagramme de cas d'utilisation pour la gestion des ventes

### Analyse de besoins :

Pour répondre aux fonctionnalités précédentes, il faut définir les données à gérer par notre système. En effet une vente est définie par un code à générer automatiquement par le système, une date (YY/MM/YYYY), un ensemble de lignes de commandes et un total calculé à partir des lignes de commandes. Chaque ligne de commande fait l’objet d’un produit accompagné d’une quantité vendu (qte) et un sous total calculé automatiquement (Prix Vente \* qte). En se basant sur ces données nous allons définir notre diagramme de classe métier pour la gestion des catégories.

1. **Diagramme de classes métier pour la gestion des ventes**

Le diagramme de classes « gestion des ventes » présenter les principaux attributs à prendre en compte.

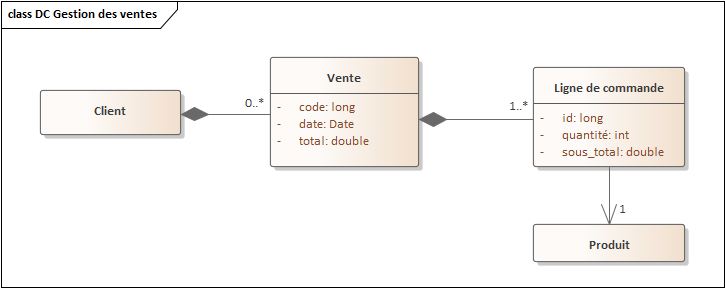


Figure 37:Diagramme de classes métier pour la gestion des ventes

Une vente appartient à un seul client et un client a zéro ou plusieurs ventes, une vente est composée d’un seule ou plusieurs lignes de commande et une ligne de commande est liée à un seul produit.

1. **Diagramme de composants :**

Les diagrammes de composants décrivent le système modélisé sous forme de composants réutilisables et mettent en évidence leurs relations de dépendance.

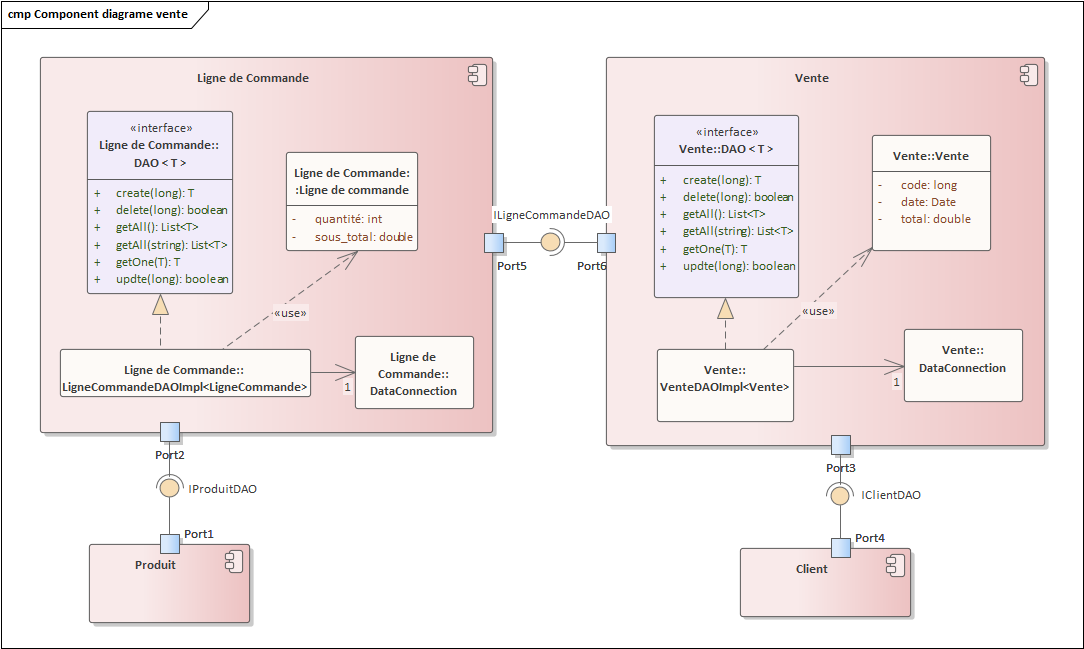


Figure 38: diagramme de composant pour la gestion des ventes

Le composant « Ligne de commande » requiert l’interface « IPrdouitDAO » fournir par Le composant « Produit ». De plus, Le composant « Vente » requiert l’interface « IClientDAO » fournir par Le composant « Client » et le composant « Ligne de commande » requiert l’interface « IVentDAO » fournir par le composant « Vente »

### Conception du système de gestion des ventes :

1. **Diagramme de classes de conception :**

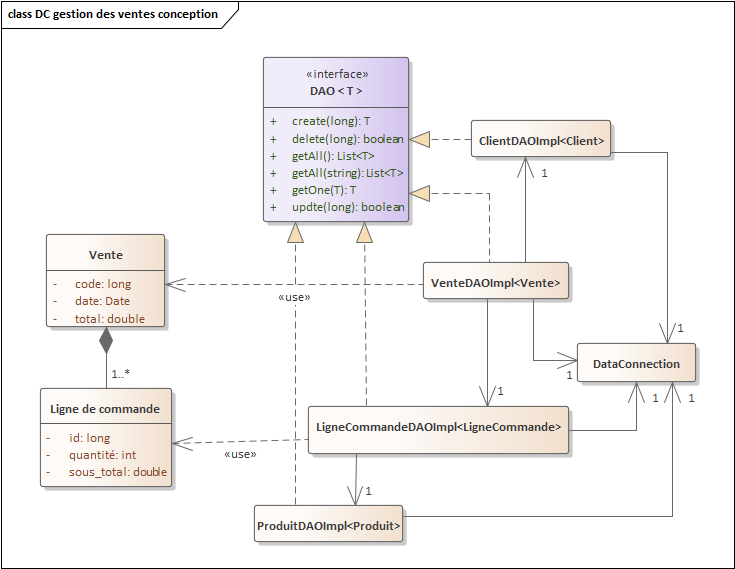


Figure 39:Diagramme de classes de conception

1. **Diagramme de séquences boite blanche**

Le diagramme de séquence est parmi les vues dynamiques importantes de UML. Il décrit, en fonction du temps, une interaction pour l’obtention d’un résultat observable.

Nous allons présenter les 2 diagrammes de séquences pour bien pour bien illustrer les cas d’utilisation :

* Diagramme de séquence ajouter une ligne de commande

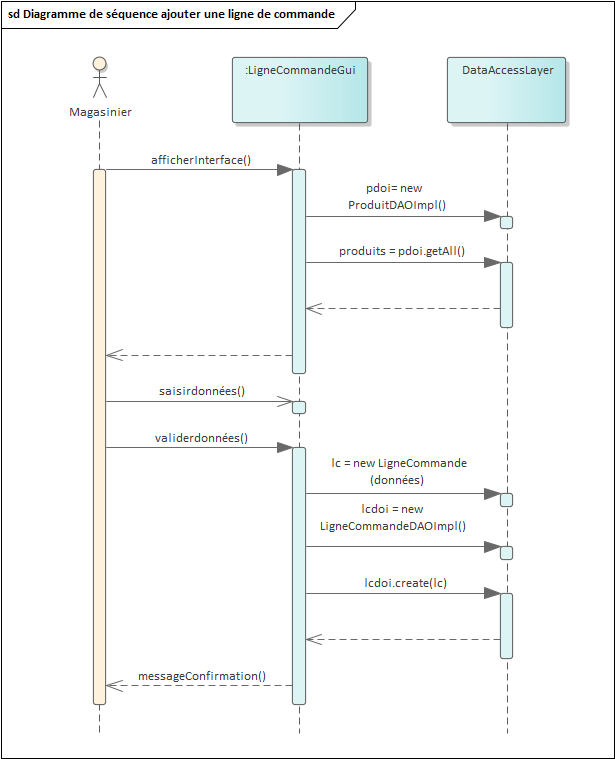


Figure 40: Diagramme de séquence ajouter une ligne de commande

* Diagramme de séquence ajouter une vente

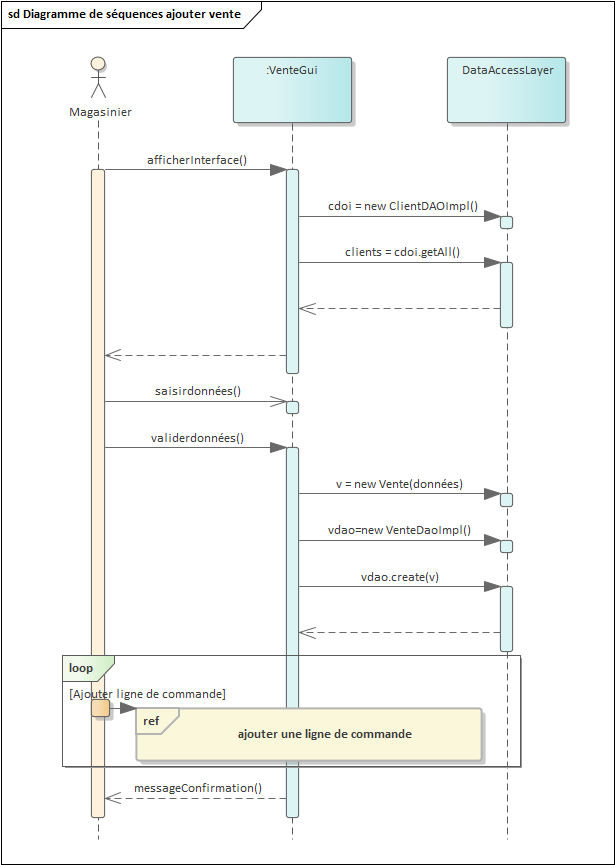


Figure 41:diagramme de séquence ajouter une vente

## Module de gestion des paiements :

### Spécification de besoins :

Le module de gestion des paiements doit permettre à ces utilisateurs d’effectuer les règlements d’une vente en les ajoutant à la base de données, d’afficher le détail de règlements d’une vente, de modifier et de supprimer des règlements. Le règlement d’une vente peut se faire de plusieurs manières :

* Règlement comptant en espèce
* Règlement comptant par chèque
* Règlement comptant en ligne
* Règlements par traites.

Pour le règlement comptant en espèce le vendeur ou le caissier valide la réception du montant total de la vente

Pour le règlement par chèque le vendeur ou le caissier valide la réception du chèque et en même temps précise les informations relatives au chèque (propriétaire du chèque, date d’effet).

Pour le règlement comptant en ligne, le client donne la carte bancaire au vendeur ou au caissier. Ensuite ce dernier insère la carte au lecteur de paiement et saisit le montant de la vente. Ensuite, le client saisit le code de sa carte pour se connecter au serveur de paiement et valider le montant à payer.

Pour le règlement par traite, le client peut répartir le montant total de la vente en plusieurs montants :

* Il peut effectuer une avance (espèce ou chèque) à payer immédiatement
* Il peut ensuite répartir le reste en plusieurs traites, chaque traite a un montant, une date de paiement prévue, et une date de paiement effective. La traite est liée à un chèque contenant son montant. Le chèque peut avoir des informations (propriétaire, banque, numéro).

1. Diagramme de cas d’utilisation :

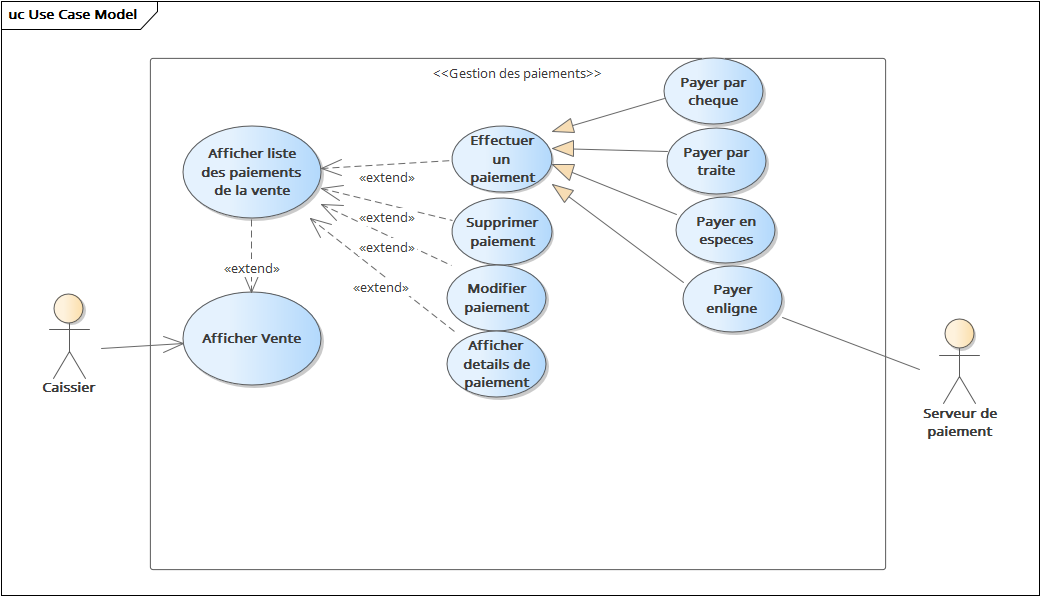


Figure 42: Diagramme de cas d’utilisation<<Gestion des paiements>>

Le diagramme de cas d’utilisation ci-dessus « gestions des paiements » met en évidence les principaux cas d’utilisation de ce module. En occurrence, le système doit donner la possibilité au vendeur d’afficher la liste des paiements d’une vente, ajouter un paiement, modifier paiement. Dans la suite, nous lui donnons la possibilité d’afficher les détails d’un paiement ou le supprimer, sans oublier la possibilité de choisir le mode de paiement avec l’aide du serveur de paiement si l’option choisie est paiement en ligne.

1. Diagramme de sequence  Boite noire:

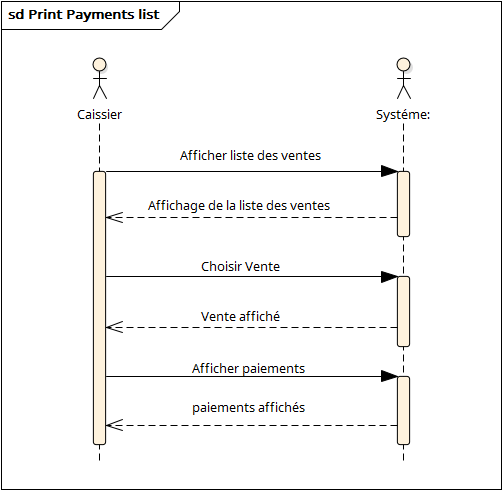


Figure 43 : Diagramme de séquence boite noire de <<Afficher liste des paiements>>

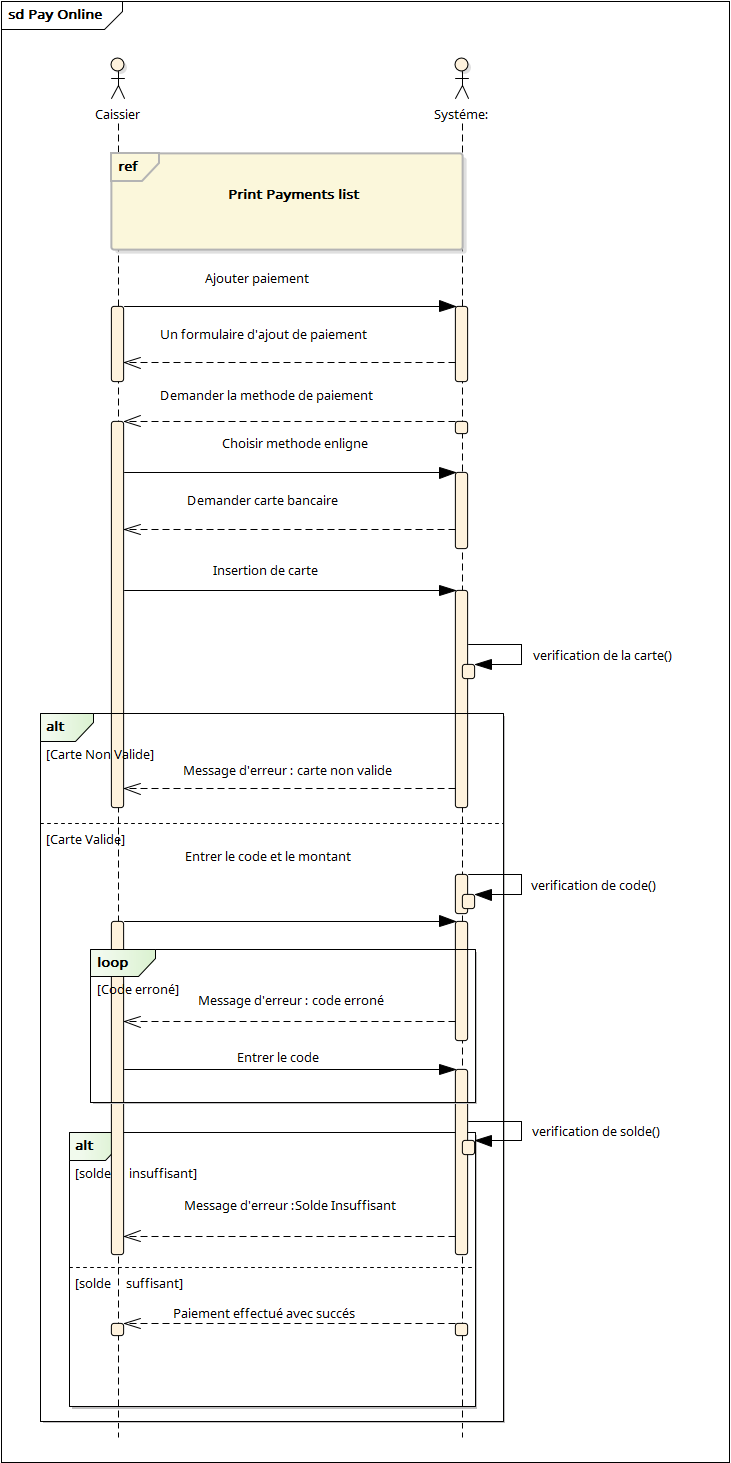


Figure 44: diagramme de séquence boite noire de <<Payer en ligne>>

### B. Analyse de besoins :

1. Diagramme de classe métier:

Comme montré dans le diagramme, une vente peut avoir un ou plusieurs paiements, un paiement est défini par une date de paiement, Montant et un identificateur et il a un mode de paiement où on spécifie le type de paiement.

Le mode de paiement peut avoir plusieurs relations , si le paiement et par chèque le mode de paiement est lié avec un chèque qui est caractérisé par la date d’effet, identificateur et le propriétaire , ce chèque représente une banque qui est définie par un ID et la nom, si le paiement est en ligne alors le montant doit être pris par un compte bancaire qui est défini par un solde, si le paiement est par traite , on crée une traite qui est identifié par la date effectif , date prévue et un ID.

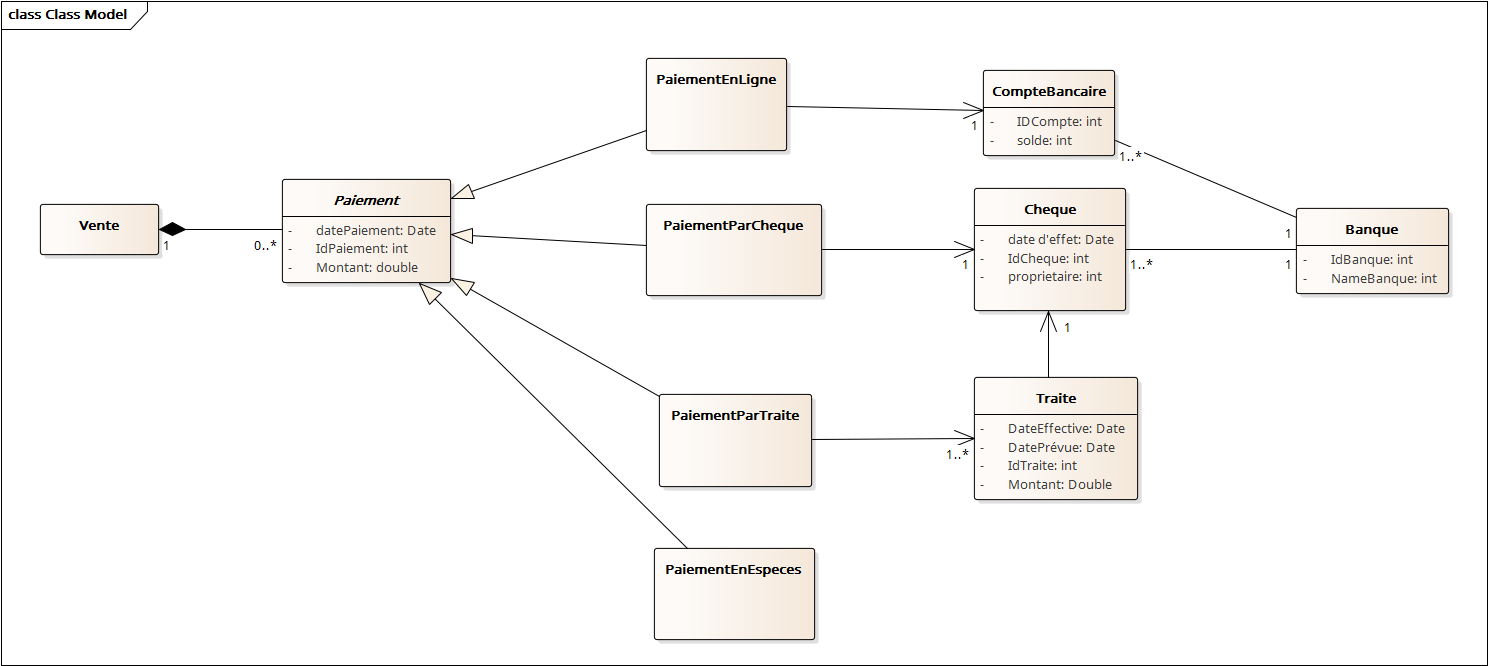


Figure 45: diagramme de classe métier

2. Diagramme de composants :

Dans notre module gestion de paiement , nous devons implementer la classe DAO qui vas être utilisée par la composant de PaymentIHM , IPayDao necessite le composant vente pour le fonctionnement des principales fonctionnalités.

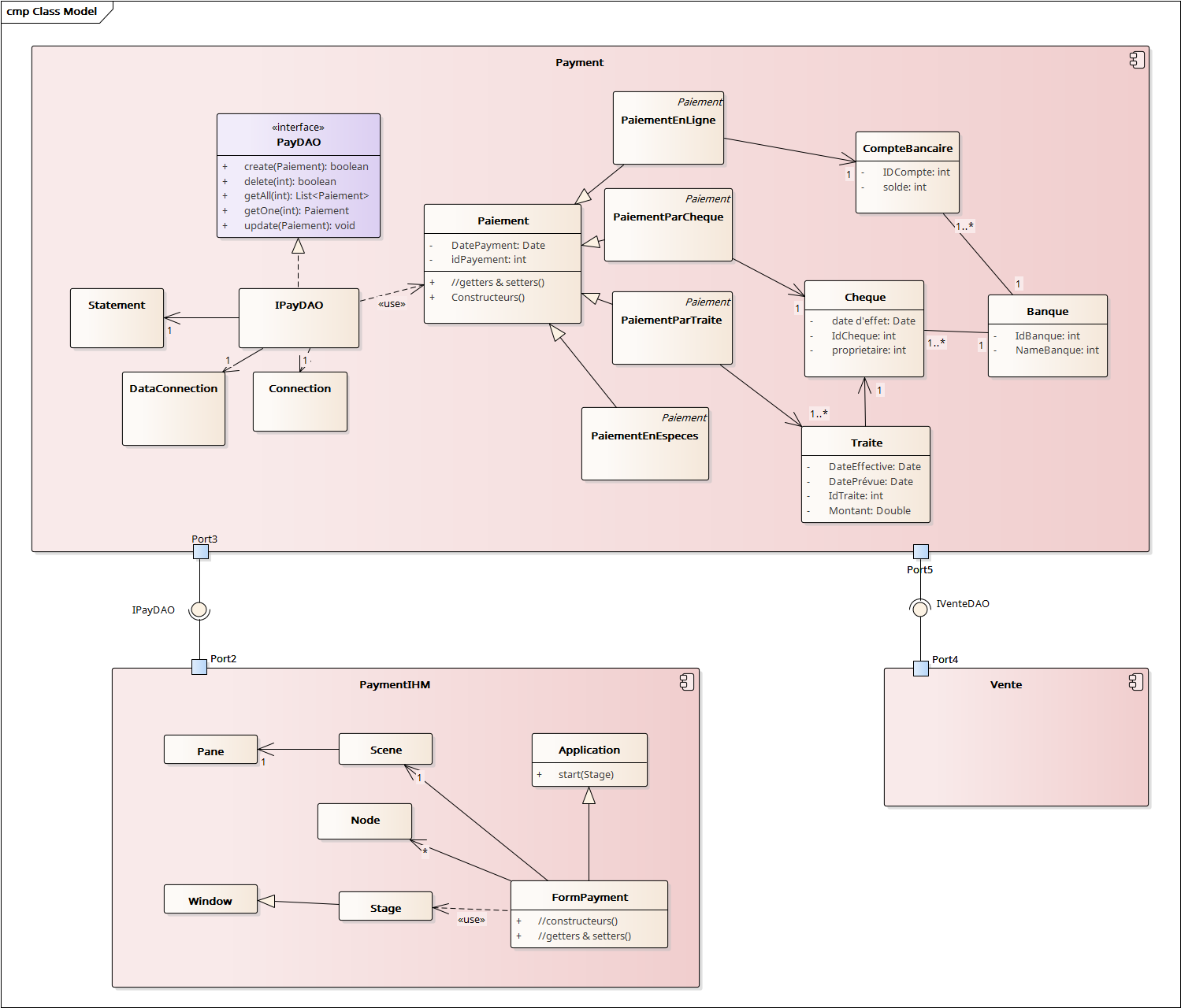


Figure 46: diagramme de classe composants

### C. Conception du système de gestion des paiements :

1. Diagramme de Classe de Conception

Le diagramme resume l’ensemble des classe à implementer dans notre application , Les mode de paiement(especes,traite,cheque,enligne) sont des classe filles de la classe Paiement , l’interface DAO utilisent un ensemble des classes(dataconnection, connection , statement ..) pour la connexion avec la base de données, IPayDAO doit definir les fonctions de PayDAO (Ajouter, modifire , afficher …).

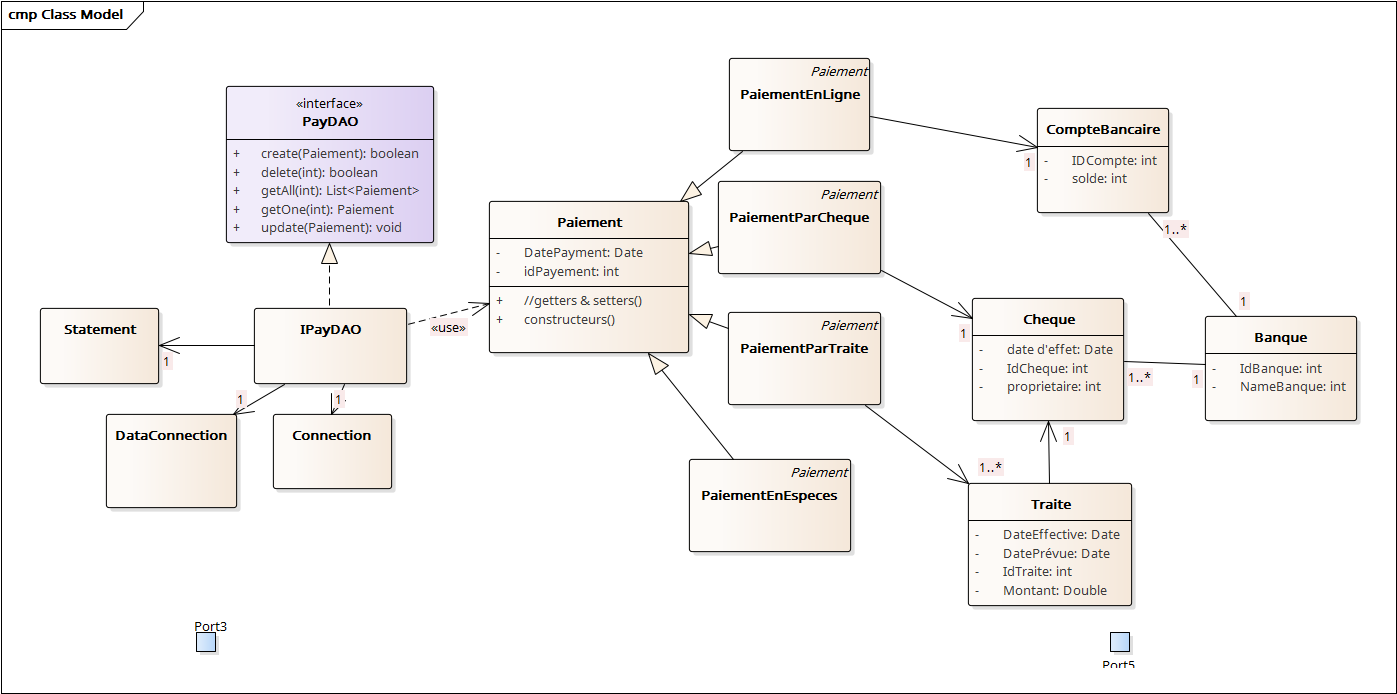


Figure 47:Diagramme de classe de conception

1. Diagramme de sequence  Boite Blanche:

Le système est une boite blanche, on s’intéresse à chaque composant du système intervenant dans la réalisation du CU, le système est modélisé comme étant un ensemble d’entités

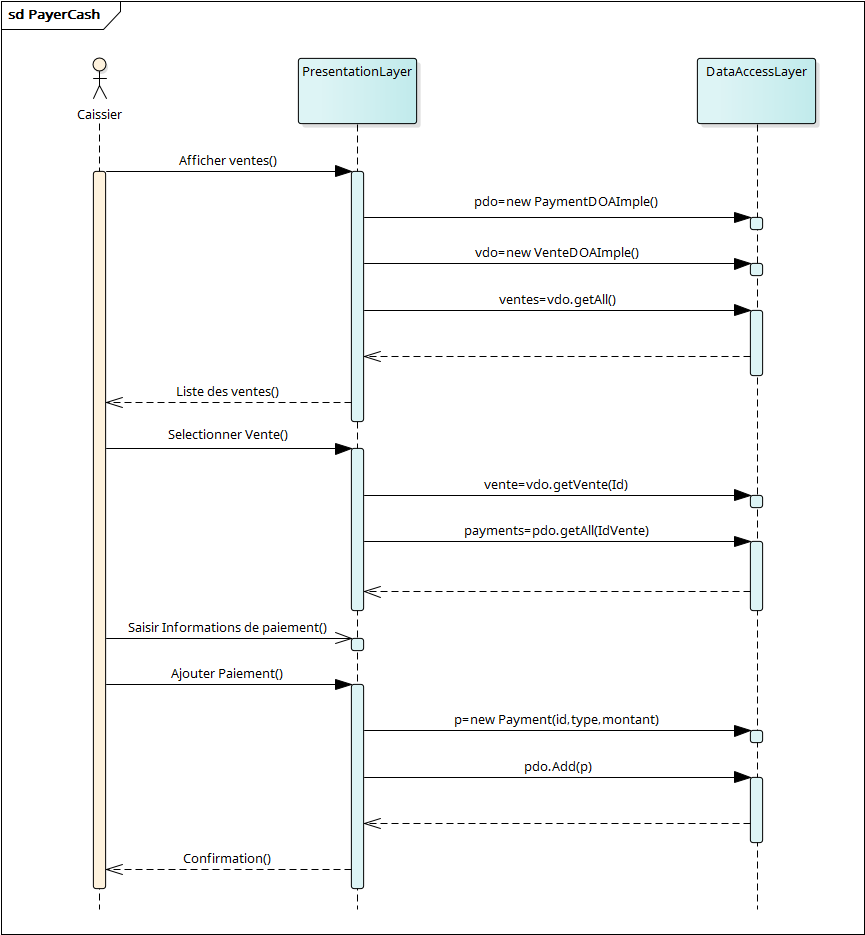


Figure 48: diagramme de séquence boite blanche de <<Payer en espèces>>

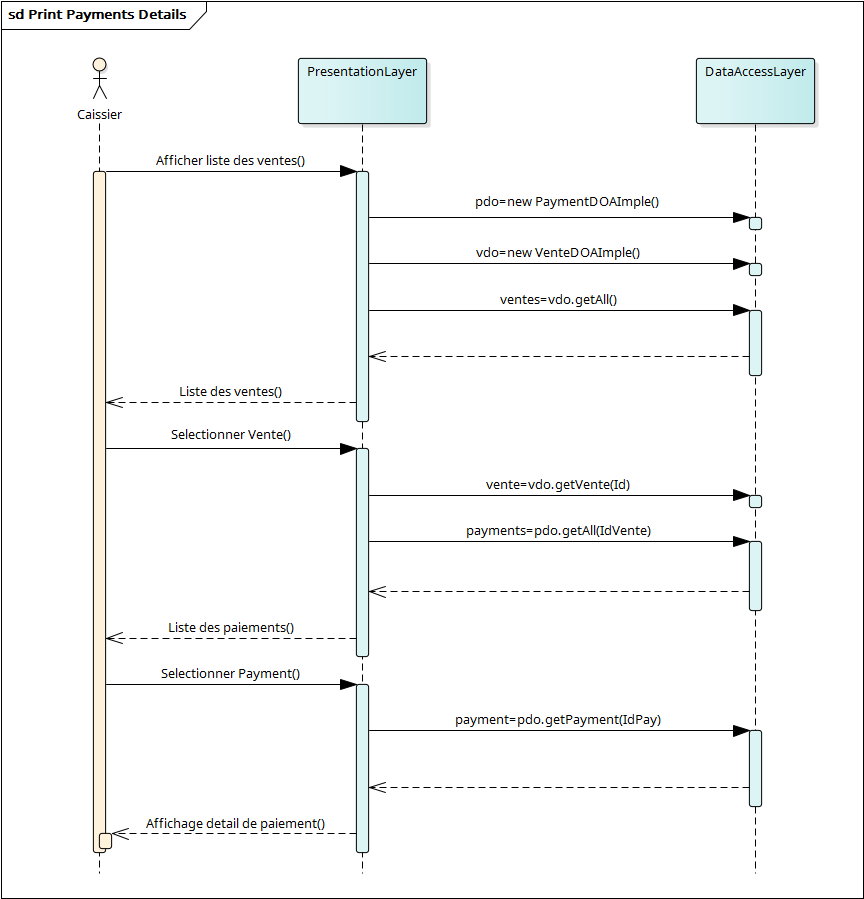


Figure 49: diagramme de séquence boite blanche de <<afficher liste de paiements>>

# Conclusion

Afin de garantir que notre logiciel de gestion de magasin répond aux exigences de délais, de budget et donne satisfaction au client, nous avant opté à utiliser le concept de génie logiciel. Ce concept en effet, désigne l'ensemble des méthodes, des techniques et outils concourant à la production d'un logiciel, au-delà de la seule activité de programmation.

Ainsi, nous avons décomposée notre projet en 4 modules : gestion de catégorie, gestion de produit, gestion de client, gestion de vente et enfin gestion de paiement.

Evidemment, il est nécessaire de mettre en place un logiciel de qualité. Pour cela, nous avons choisi d’adopter un cycle de vie en Y non seulement à cause de la petite taille de notre logiciel et son pilotage de risque mais aussi sa nature itératif et incrémentale ainsi que la simplicité de son intégration avec la méthode agile « Scrum ». En effet, le processus Scrum étant Itératif et incrémental s’adapte parfaitement à la décomposition de notre projet.

Pour réaliser chaque module, nous avons adopter l’architecture à 2 couches :

* PresentationLayer (IHM) en utilisant JAVAFX
* DataAccessLayer (accès aux données) en adoptant le design pattern DAO.

Enfin, à travers ce projet nous avons pu appliquer la majorité des notions apprises dans les deux module java et UML.

En effet, ce projet demandait des notions au niveau programmation java, nous avons donc dû faire preuve d'une certaine agilité en termes de compétences et de connaissances.

Ce projet nous a également permis de consolider des bases dans un nouveau monde de programmation JavaFX.

Finalement, nous tenons à remercier monsieur Abdelwahab NAJI pour ses efforts considérables pendant toute l’année, et pour le temps qui a consacré pour la réussite de cette formation en java.

# Table des figures

[Figure 1 : activité d'un processus 8](#_Toc29129589)

[Figure 2: activités d’un processus de développement logiciel 8](#_Toc29129590)

[Figure 3: cycle de vie en cascade 9](file:///C:\Users\khadi\Desktop\magasin.docx#_Toc29129591)

[Figure 4: cycle de vie en V 9](file:///C:\Users\khadi\Desktop\magasin.docx#_Toc29129592)

[Figure 5 : Cycle de vie incrémentale 10](file:///C:\Users\khadi\Desktop\magasin.docx#_Toc29129593)

[Figure 6: Cycle de vie en spirale 10](file:///C:\Users\khadi\Desktop\magasin.docx#_Toc29129594)

[Figure 7 : cycle de vie en Y 11](file:///C:\Users\khadi\Desktop\magasin.docx#_Toc29129595)

[Figure 8 : méthode agile SCRUM 13](file:///C:\Users\khadi\Desktop\magasin.docx#_Toc29129596)

[Figure 9: Architecture de l'application de gestion de magasin 15](file:///C:\Users\khadi\Desktop\magasin.docx#_Toc29129597)

[Figure 10: diagramme de contexte de gestion de magasin 18](#_Toc29129598)

[Figure 11: diagramme de package 19](#_Toc29129599)

[Figure 12: diagramme de cas d'utilisation pour la gestion des catégories 20](#_Toc29129600)

[Figure 13: diagramme de classe pour la gestion des catégories 21](#_Toc29129601)

[Figure 14: Diagramme de séquence de cas d’utilisation afficher catégorie 22](#_Toc29129602)

[Figure 15 : Diagramme de séquence de cas d’utilisation Ajouter catégorie 23](#_Toc29129603)

[Figure 16: diagramme de classe DAO 24](#_Toc29129604)

[Figure 17: categorieIHM 24](#_Toc29129605)

[Figure 18: Diagramme de classe CategorieDAO 25](#_Toc29129606)

[Figure 19 : diagramme de composant 26](#_Toc29129607)

[Figure 20: diagramme de séquence afficher catégorie boite blanche 27](#_Toc29129608)

[Figure 21:diagramme de séquence Ajouter catégorie boite blanche 28](#_Toc29129609)

[Figure 22: diagramme de cas d'utilisation de système de gestion des produits 30](#_Toc29129610)

[Figure 23: diagramme de séquence boite noire pour la création d'un produit 31](#_Toc29129611)

[Figure 24: diagramme de classe métier de gestion des produits 32](#_Toc29129612)

[Figure 25: diagramme de composants de gestion des produits 33](#_Toc29129613)

[Figure 26: diagramme de séquence pour le cas créer un produit 34](#_Toc29129614)

[Figure 27: diagramme de cas d'utilisation pour la gestion des clients 35](#_Toc29129615)

[Figure 28 : diagramme de classe métier 35](#_Toc29129616)

[Figure 29: Diagramme de séquence afficher client 36](#_Toc29129617)

[Figure 30: Diagramme de séquence Ajouter client 37](#_Toc29129618)

[Figure 31: clientIHM 38](#_Toc29129619)

[Figure 32: ClientDao 39](#_Toc29129620)

[Figure 33: diagramme de composant pour la gestion des clients 40](#_Toc29129621)

[Figure 34:Diagramme de séquence afficher clients boite blanche 41](#_Toc29129622)

[Figure 35:diagramme de séquence ajouter client 42](#_Toc29129623)

[Figure 36: diagramme de cas d'utilisation pour la gestion des ventes 43](#_Toc29129624)

[Figure 37:Diagramme de classes métier pour la gestion des ventes 44](#_Toc29129625)

[Figure 38: diagramme de composant pour la gestion des ventes 45](#_Toc29129626)

[Figure 39:Diagramme de classes de conception 46](#_Toc29129627)

[Figure 40: Diagramme de séquence ajouter une ligne de commande 47](#_Toc29129628)

[Figure 41:diagramme de séquence ajouter une vente 48](#_Toc29129629)

[Figure 42: Diagramme de cas d’utilisation<<Gestion des paiements>> 49](#_Toc29129630)

[Figure 43 : Diagramme de séquence boite noire de <<Afficher liste des paiements>> 50](#_Toc29129631)

[Figure 44: diagramme de séquence boite noire de <<Payer en ligne>> 51](#_Toc29129632)

[Figure 45: diagramme de classe métier 52](#_Toc29129633)

[Figure 46: diagramme de classe composants 53](#_Toc29129634)

[Figure 47:Diagramme de classe de conception 54](#_Toc29129635)

[Figure 48: diagramme de séquence boite blanche de <<Payer en espèces>> 55](#_Toc29129636)

[Figure 49: diagramme de séquence boite blanche de <<afficher liste de paiements>> 56](#_Toc29129637)