Université Assane Seck de Ziguinchor

UFR : Sciences et Technologies

Filière : Mathématiques Physique Chimie Informatique

Département : Informatique

Mémoire de master 2

Développement d'une application de gestion des pharmacies pour la facilitation de la recherche de médicaments au Sénégal

(SamaPharmacie)

Par : **Mr BABACAR KA**

Sous la direction de : Dr MARIE NDIAYE Maitre de stage : Dr JEANNE TAVARES GUEYE

Année universitaire 2016-2017

Dédicaces

Je dédie ce travail à ma mère, Marie Françoise SAGNA (Séna), à mon père Omar KA, à mes tantes Anna CABO et Claire SAGNA.

Je le dédie spécialement à ma grand-mère Philomène BADIANE.

Remerciements

Je remercie Allah qui m'a permis d'atteindre ce niveau d'étude sans aucune force ni puissance de ma part.

Je remercie mon encadreur Dr Marie Ndiaye qui malgré son emploi du temps chargé a accepté de m'encadrer et de me consacrer une partie de son temps précieux toutes les semaines depuis le début de notre travail.

Je remercie mes parents pour nous avoir inscrits et laissés à l'école. Je les remercie pour le nombre incalculable de prières qu'ils font pour nous.

Je remercie ma tante Anna CABO et mon frère Papa Arfan Mantrand André SANE ainsi tous les autres membres de la famille.

Je remercie ma tante Claire SAGNA et tous les membres de la famille SAGNA qui nous accompagnent et nous soutiennent dans tous nos projets.

Je remercie mon maitre de stage Dr Jeanne TAVARES GUEYE ainsi que son mari Mr Cheikh Tidiane GUEYE et toutes les personnes qui travaillent à la pharmacie à savoir Mr Alouise BASSENE, Mr Gracien DIEDHIOU, Mlle Mathilde TAVARES et Mr Antoine TAVARES. Mon stage à la pharmacie s'est bien déroulé grâce à leur accueil chaleureux, leur gentillesse et leur ouverture.

Je remercie Mr le maire Mamadou SOW qui m'a mis en relation avec mon maitre de stage.

Je remercie Mr Pape Amadou Humbe SANE pour tout ce qu'il a fait pour moi.

Je remercie Mr Jean Pierre DIATTA pour sa gentillesse et sa disponibilité.

Je remercie tous mes enseignants de l'école primaire à l'université.

Table des matières

Liste des tableaux

Listes des figures

Glossaire

Ordonnancier : registre sur lequel on inscrit le nom d'un client, son quartier et certains produits achetés.

2TUP : Two Track Unified Process

Stick-man: Dessin représentant un acteur.

Tiers : partie, couche.

SGBDR : Système de gestion des bases de données relationnelles.

Client : dispositif pouvant utiliser notre application.

Serveur de SGBD : machine hébergeant le SGBD

Serveur web : machine hébergeant la partie web de notre application.

Serveur d'application : machine hébergeant la partie métier de notre application

Classe : Objet virtuel simulant un objet de la vie réelle.

Méthode : action, comportement d'une classe.

Java : Langage de programmation objet.

Persistance : mécanisme responsable de la sauvegarde des données.

Transaction : Une transaction est une succession d'opérations, le plus souvent vers une base de données, dont leurs exécutions doivent toutes réussir ou aucune ne doit être exécutée. Elle permet d'assurer l'intégrité des données.

Déploiement : processus selon lequel une application ou un composant fini est distribué en vue de son installation sur d'autres ordinateurs.

Injection de dépendances : Consiste à créer dynamiquement (injecter) les dépendances entre les différentes classes en s'appuyant sur une description (fichier de configuration ou métadonnées) ou de manière programmatique

Open Source : libre

Framework : ensemble d'outils disponibles pour développer mieux et plus vite. .

Managed Bean : Classe Java qui fait la liaison entre les pages et le code métier.

Langage EL : Petits bouts de code utilisés sur les pages jsf pour lier un composant à un attribut du Managed Bean.

JPA : Java Persistence API

Data source : Source de données.

JNDI: Java Naming and Directory Interface. Cette API fournit une interface unique pour utiliser différents services de nommages ou d'annuaires et définit une API standard pour permettre l'accès à ces services.

Introduction

Contexte

Un jour notre grand-mère qui souffrait d'asthme a eu une crise aux environs de minuit, nous l'avons alors portée jusqu'à la voiture pour l'acheminer à l'hôpital. Une fois là-bas le docteur nous a prescrit un médicament que nous devions acheter d'urgence. Sortis de l'hôpital nous avons parcouru toute la ville mais malheureusement la majorité des pharmacies étaient fermées et celles qui étaient ouvertes ne disposaient pas du médicament en rayon.

Nous avons loué les services d'un taxi avec lequel nous avons parcouru les quartiers périphériques sans résultat.

Enfin, de retour à l'hôpital, on nous a annoncé que la grand-mère était sauvée de justesse grâce à la seule expertise du docteur.

Problématique :

Cet exemple montre qu'au Sénégal, que pour acheter un médicament, les clients sont obligés parfois de parcourir de longues distances, de dépenser beaucoup d'énergie et d'argent, de perdre beaucoup de temps pendant que leurs malades attendent pour recevoir des soins. Devant ce manque de communication les clients ne disposent d'aucune information relative à la disponibilité des médicaments dans les différentes pharmacies d'une part et ne savent pas lesquelles sont de garde d'autre part. Ces informations auraient pu leur apporter une économie de temps et d'argent en leur évitant des allers et retours.

Objectif :

Mettre en place une solution de gestion des pharmacies capable en même temps, de faciliter la recherche de médicaments, de calculer le coût total d'une ordonnance et de faciliter la recherche de pharmacies de garde.

Résultat attendu :

Cette solution devrait permettre aux pharmaciens de, gérer leurs stocks, gérer les ventes, gérer les bons, gérer les dépenses, gérer les utilisateurs, passer des commandes. Elle devrait rendre disponible les listes de pharmacies de garde partout au Sénégal. Elle devrait aussi mettre à la

disposition des clients, une solution pour faciliter la recherche de médicaments, une calculatrice pouvant leur permettre de calculer le cout d'une ordonnance.

Démarche méthodologique :

Pour atteindre notre objectif, nous devions obligatoirement comprendre le fonctionnement des pharmacies et pour y parvenir, nous avons fait des recherches sur internet, nous avons fait des enquêtes et finalement nous avons fait un stage qui a duré six mois dans une pharmacie.

Plan du document :

Notre document sera structuré en quatre (04) chapitres. Dans le premier nous allons faire une petite présentation de la pharmacie et notre processus de développement logiciel. Le deuxième chapitre nous permettra de capturer tous les besoins (fonctionnels et techniques). Dans le troisième, nous ferons l'analyse et la conception. Le quatrième et dernier chapitre nous donnera un petit aperçu sur le codage et les résultats obtenus.

Chapitre 1 : Présentation de la pharmacie et de notre processus de développement

Incontournable et omniprésente, l'informatique régit aussi l'organisation de l'officine. Une pharmacie sans écran d'ordinateur est devenue rare et sans doute en voie d'extinction. Car l'informatique n'a de cesse d'étendre son territoire, toujours à la recherche d'une plus grande efficacité à offrir au pharmacien, au patient et à la société [1]. C'est dans ce cadre que s'inscrit notre sujet de mémoire intitulé « ***Développement d'une application de gestion des pharmacies pour la facilitation de la recherche de médicaments au Sénégal***».

Dans cette partie nous allons d'abord présenter le fonctionnement des pharmacies et ensuite présenter le processus qu'on a adopté pour pouvoir mettre en place notre application.

I. Présentation de la pharmacie

Ici, nous allons présenter le fonctionnement des pharmacies en nous basant sur une officine appelée Pharmacie Lyndiane Jatiir.

I .1. Historique, situation, organisation et fonctionnement

La pharmacie Lyndiane Jatiir est créée en 1992 par arrêté ministériel et s'appelait pharmacie Colobane. A l'époque, elle se trouvait au quartier Colobane près de l'ancienne SONADIS sur la même route que le Collège Saint Charles Lwangua.

Aujourd'hui, elle se situe à Lyndiane plus exactement au Boulevard Alpha.

Elle compte quatre (4) employés et la pharmacienne. Les employés sont divisés en groupes qui se relaient tous les jours à 16 h. Elle est ouverte de 8h à 21h tous les jours sauf les jours fériés.

Comme toutes les pharmacies, elle travaille avec des fournisseurs (LABOREX, COPHASE, DUOPHARM etc.) qui lui livrent des produits avec un bon de livraison (BL) sur lequel apparaissent tous les produits commandés, leurs prix publics, leurs prix de cession, leurs prix totaux et le montant de la facture.

Une fois que les produits sont livrés à la pharmacie, nous les réceptionnons en contrôlant, pour chaque produit du BL, la date de péremption, la quantité livrée et l'état du produit reçu.

Après contrôle des produits, nous les étiquetons d'abord, ensuite nous les rangeons (selon leur forme, leur tableau) et enfin nous saisissons le BL à la machine.

I.2. Terminologie

Produit (Médicament) : représente la marchandise vendue dans la pharmacie.

Il peut avoir une dénomination commune internationale (DCI ou CID en anglais). Il peut être sous forme comprimé, sirop, pommade, injection, ovule, suppositoire, usage externe, sachet ou ampoule. Il a un dosage, il peut appartenir à un tableau. Il a aussi une désignation et un prix.

- ✓ il existe des produits dont le prix dépend d'un pourcentage.
  (Prix = prix unitaire * %+TVA)
- ✓ un prix de cession
  Prix d'achat des pharmaciens.
- ✓ un prix public
  Prix de vente pour le public.

Les produits sont livrés par un fournisseur ce qui fait que les prix diffèrent parfois de quelques francs.

Stock : C'est la quantité de produits présents dans la pharmacie. Dans un stock on peut avoir plusieurs échantillons dont les dates de péremptions sont différentes ce qui fait que ces dernières sont gardées à l'intérieur en attendant que celles qui ont les dates de péremptions les plus proches soient vendues.

Vente : C'est la cession de produits contre leurs prix de vente. Tous les employés de la pharmacie peuvent vendre. Une vente, peut contenir plusieurs produits, a une date, un total facture et un total encaissé. Toute vente d'un produit du tableau A (rouge) et C (Vert) doit être notée dans l'ordonnancier.

Il existe aussi des réductions selon les types de clients.

Commande : C'est le fait de demander des produits à un fournisseur. Tous les employés de la pharmacie peuvent passer une commande. Une commande est adressée à un fournisseur, faite à une date par un utilisateur et validée par un utilisateur. Elle est faite quand la quantité de certains produits atteint une valeur minimale fixée.

Livraison : C'est l'action de livrer les produits commandés par une pharmacie. Elle est faite par un fournisseur. Elle a un numéro de bordereau, contient la liste des produits commandés et le total à payer.

Approvisionnement : C'est le fait d'augmenter la quantité de certains stocks. Il est fait après une livraison. Il a le numéro de bordereau de la livraison, l'identifiant du fournisseur et la date de l'approvisionnement.

Client : personne venant acheter souvent des produits à la pharmacie.

Bon : C'est le fait de sortir un produit de la pharmacie sans encaisser l'argent immédiatement. La pharmacie peut faire des bons à des clients connus ou garantis par un employé de la pharmacie. Un client peut avoir un ou plusieurs bons. Un bon peut contenir plusieurs produits.

Gestion des avoirs : Un avoir c'est un produit qui est déjà acheté par un client mais pas encore livré au client.

Sorties : Dépenses faites par la pharmacie pour son bon fonctionnement.

I.3. Présentation du système en place

Dans tous les domaines, il est important de voir ce qui existe déjà, y réfléchir pour pouvoir garder ce qui en est bon, essayer de corriger ce qui ne l'est pas et ajouter de nouvelles choses si possible. Ainsi, dans cette partie nous allons étudier l'existant en nous basant particulièrement sur un logiciel utilisé dans la pharmacie. Dans cette étude nous allons d'abord voir quelles sont les technologies utilisées ensuite nous verrons les fonctionnalités offertes et enfin nous allons voir les problèmes rencontrés par ses utilisateurs.

I.3.1. Technologies utilisées

Ce logiciel est créé sous Excel 2007 avec une base de données Access 2007.

I.3.2. Les fonctionnalités offertes par ce logiciel

Le logiciel offre les fonctionnalités suivantes :

- ✓ *Authentification des utilisateurs :*
- ✓ *Enregistrement des produits*
- ✓ *Gestion des ventes*
- ✓ *Gestion des clients*
- ✓ *Gestion des bons*

I.3.3. Les problèmes rencontrés par les utilisateurs

- ✓ Comme c'est un logiciel installé et configuré avec une base de données locale alors :
  - Si la machine tombe en panne le pharmacien ne pourra pas enregistrer les produits tant qu'elle n'est pas réparée.
  - S'il voulait utiliser une autre machine, il serait obligé d'appeler le technicien, trouver une connexion internet afin qu'il puisse faire l'installation à distance en contrôlant la machine avec team viewver.
  - Si malheureusement la panne lui fait perdre toutes ses données alors il perd le suivi de sa pharmacie via le logiciel.
  - Pour que le technicien se déplace il faut lui payer le déplacement, le logement et la nourriture.
- ✓ Impossible que plusieurs utilisateurs utilisent le logiciel au même moment.
- ✓ Manque de suivi du logiciel :

  Depuis que le logiciel a été acheté, il n'a pas subi de mises à jour. Et le technicien (celui qui lui a vendu à la pharmacie) n'a jamais mis les pieds à la pharmacie pour des questions liées au logiciel.

II. Présentation du processus de développement

Un processus définit une séquence d'étapes, en partie ordonnées, qui concourent à l'obtention d'un système logiciel ou à l'évolution d'un système existant **[2].**

Dans notre cas on va essayer d'adopter un processus capable de répondre aux contraintes de changements continuels imposées au système d'information appelé 2TUP en passant d'abord par la capture des besoins (fonctionnels et techniques), ensuite l'analyse et la conception et enfin le codage et la démonstration.

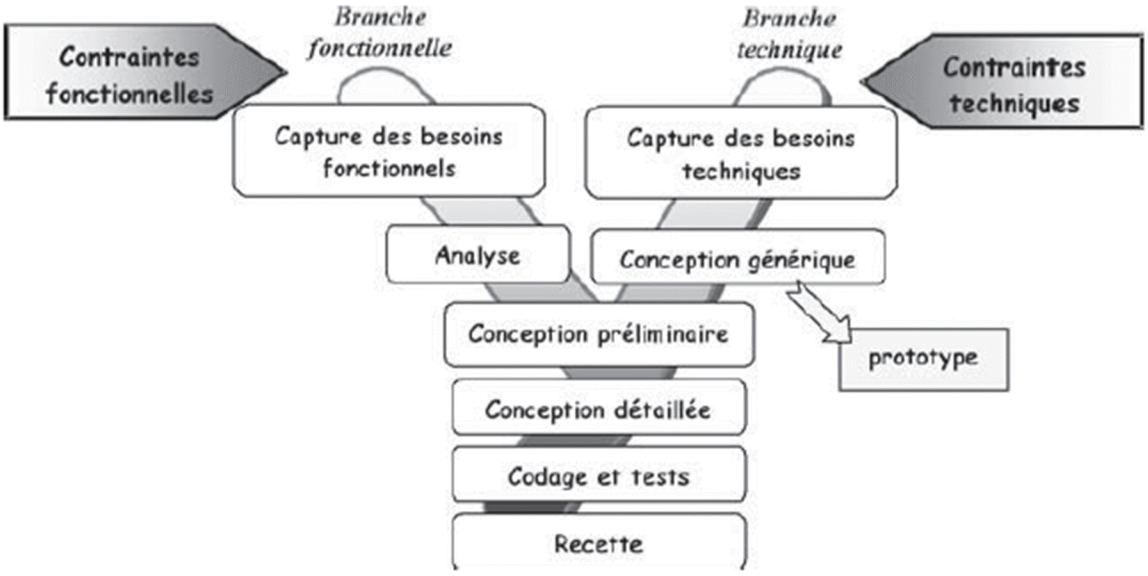


Figure 1 : Le processus de développement en Y [3]

Chapitre 2 : Capture des besoins

Dans ce chapitre nous allons recenser tous les besoins. Ainsi nous allons voir d'abord les besoins fonctionnels ensuite les besoins techniques.

I. Les besoins fonctionnels

La capture des besoins fonctionnels est la première étape de la branche gauche du cycle en Y. Elle formalise et détaille l'étude qui a été faite au cours du chapitre précédant. Dans cette partie nous allons essayer de nous conformer aux étapes de la **Figure2** ci-dessous. Cependant, les étapes **1** (Identification des cas d'utilisation) et **3** (Organisation des cas d'utilisation) sont associées pour donner une étape, l'étape **5** (Validation et consolidation) n'est pas traitée. Ainsi notre capture des besoins fonctionnels sera structurée en trois étapes. Dans la première nous allons identifier les acteurs, identifier les cas d'utilisation, les organiser et représenter nos diagrammes de cas d'utilisation. Dans la deuxième étape nous ferons les descriptions des différents cas d'utilisation. Dans la troisième et dernière étape, nous essayerons d'identifier les classes participantes..

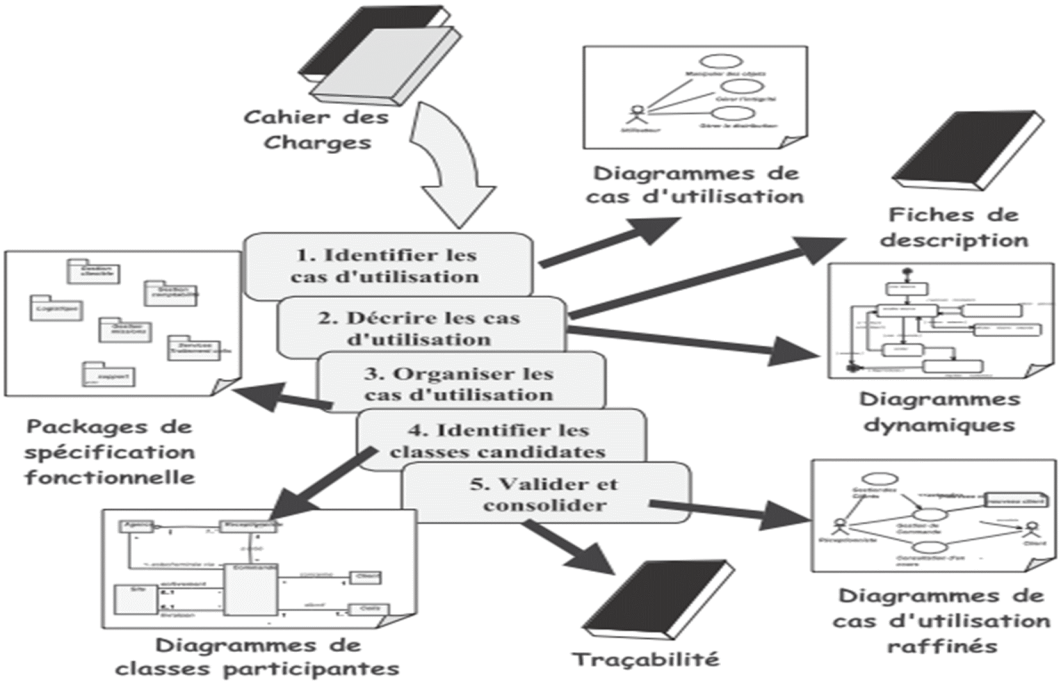


Figure 2 : Démarche de capture des besoins fonctionnels **[4]**

I.1. Identification (acteurs, domaines fonctionnels, cas d'utilisation) et représentation des diagrammes de cas d'utilisation

Dans cette section nous allons identifier d'abord les acteurs qui interagissent avec notre système ensuite nous allons identifier les domaines fonctionnels et leurs cas d'utilisation et enfin nous représenterons les diagrammes de cas d'utilisation.

I.1.1. d'utilisation Identification des acteurs

Un *acteur* représente l'abstraction d'un rôle joué par des entités externes (Utilisateur, dispositif matériel ou autre système) qui interagissent directement avec le système étudié **[5].** Après lecture du cahier des charges nous avons identifié cinq (5) acteurs qui sont : le pharmacien qui a tous les droits sur la pharmacie, les employés, les clients, le fournisseur à qui les commandes sont adressées et l'administrateur de l'application.

I.1.2. Identification des domaines fonctionnels et des cas d'utilisation

Un cas d'utilisation (use case) représente un ensemble de séquences d'actions réalisées par le système et produisant un résultat observable intéressant pour un acteur particulier **[6].**

Un cas d'utilisation modélise un service rendu par le système. Il exprime les interactions acteurs/système et apporte une valeur ajoutée « notable » à l'acteur concerné **[7].**

Quant au domaine fonctionnel, c'est le regroupement des fonctionnalités de notre système par domaines d'activité.

Le tableau ci-dessous liste les différents domaines fonctionnels avec leurs cas d'utilisation.

Cas d'utilisation	Domaines fonctionnels
S'authentifier (CU01)	**Gestion des utilisateurs (DF01)**
Ajouter utilisateur (CU02)	
Editer utilisateur (CU03)	
Supprimer utilisateur (CU04)	
Afficher utilisateur (CU05)	
Ajouter commande (CU01)	**Gestion du stock (DF02)**
Editer commande (CU02)	
Supprimer commande (CU03)	
Afficher commandes (CU04)	
Ajouter stock (CU05)	
Editer stock (CU06)	
Supprimer stock (CU07)	
Afficher stock (CU08)	

Approvisionner (CU09)	
Editer Approvisionnement (CU10)	
Afficher Approvisionnement (CU11)	
	Gestion de la clientèle (DF03)
Ajouter un bon (CU01)	
Editer un bon(CU02)	
Supprimer un bon (CU03)	
Afficher bons (CU04)	
Ajouter un client (CU05)	
Editer un client (CU06)	
Supprimer un client (CU07)	
Afficher clients (CU08)	
Calculer coût ordonnance(CU09)	
Voir pharmacies de garde(CU10)	
Voir astuces sur la santé(CU11)	
Rechercher produit(CU12)	
Ajouter une vente (CU01)	**Gestion ventes/dépenses (DF04)**
Editer une vente (CU02)	
Supprimer une vente (CU03)	
Afficher ventes (CU04)	
Ajouter une dépense (CU05)	
Editer une dépense (CU06)	
Supprimer une dépense (CU07)	
Afficher dépenses (CU08)	
Ajouter fournisseur (CU01)	**Administration (DF05)**
Editer fournisseur (CU02)	
Supprimer fournisseur (CU03)	
Afficher tous les fournisseurs (CU04)	
Ajouter pharmacie (CU05)	
Editer pharmacie (CU06)	
Supprimer pharmacie (CU07)	
Afficher pharmacies (CU08)	

Tableau 1 : Liste des domaines fonctionnels et leurs cas d'utilisation.

I.1.3. Diagrammes de cas d'utilisation

Après avoir identifié les cas d'utilisation de notre système on peut maintenant passer à leur modélisation. Cette modélisation consiste à regrouper tous les cas d'utilisation, représentés par des ellipses, dans un rectangle représentant notre système « pharmacie » et des acteurs, représentés par des stick-man, reliés aux ellipses par des traits. Dans notre cas chaque domaine fonctionnel aura son diagramme, tous les cas d'utilisation requièrent une authentification mais certains liens « include » ne seront pas matérialisés pour ne pas trop surcharger le diagramme.

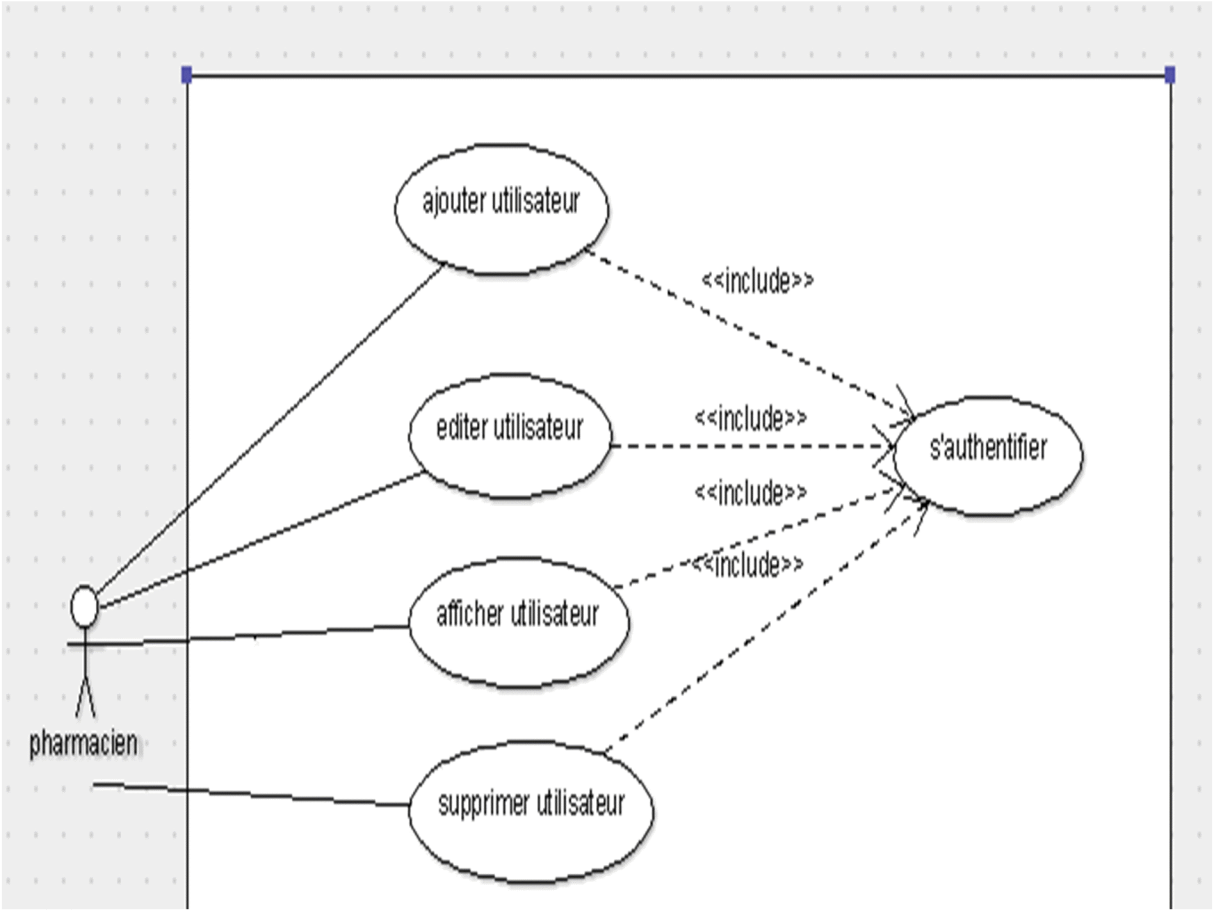


Figure 3 : Diagramme de cas d'utilisation du domaine fonctionnel **Gestion des utilisateurs (DF01)**

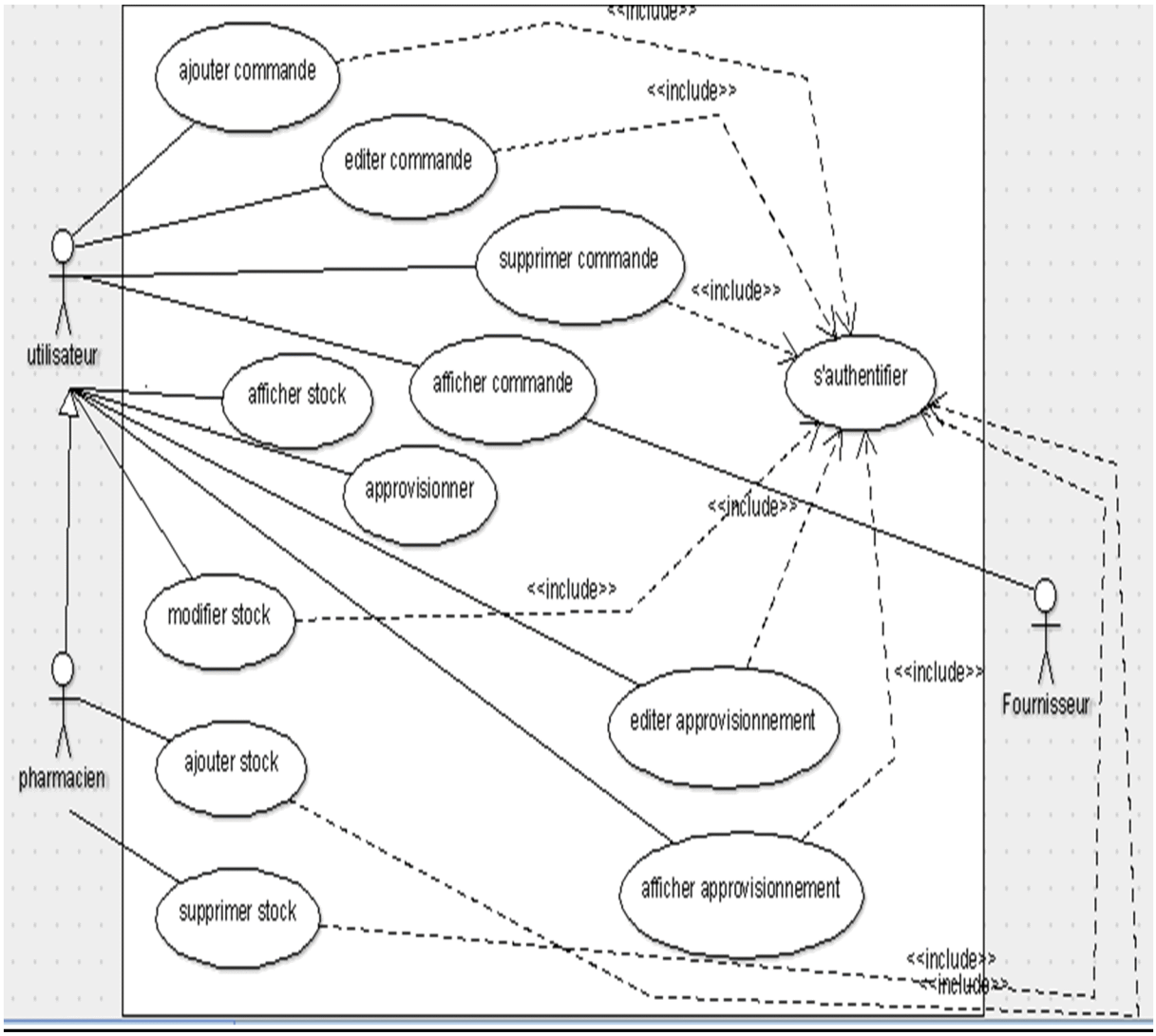


Figure 4 : Diagramme de cas d'utilisation du domaine fonctionnel **Gestion du stock (DF02)**

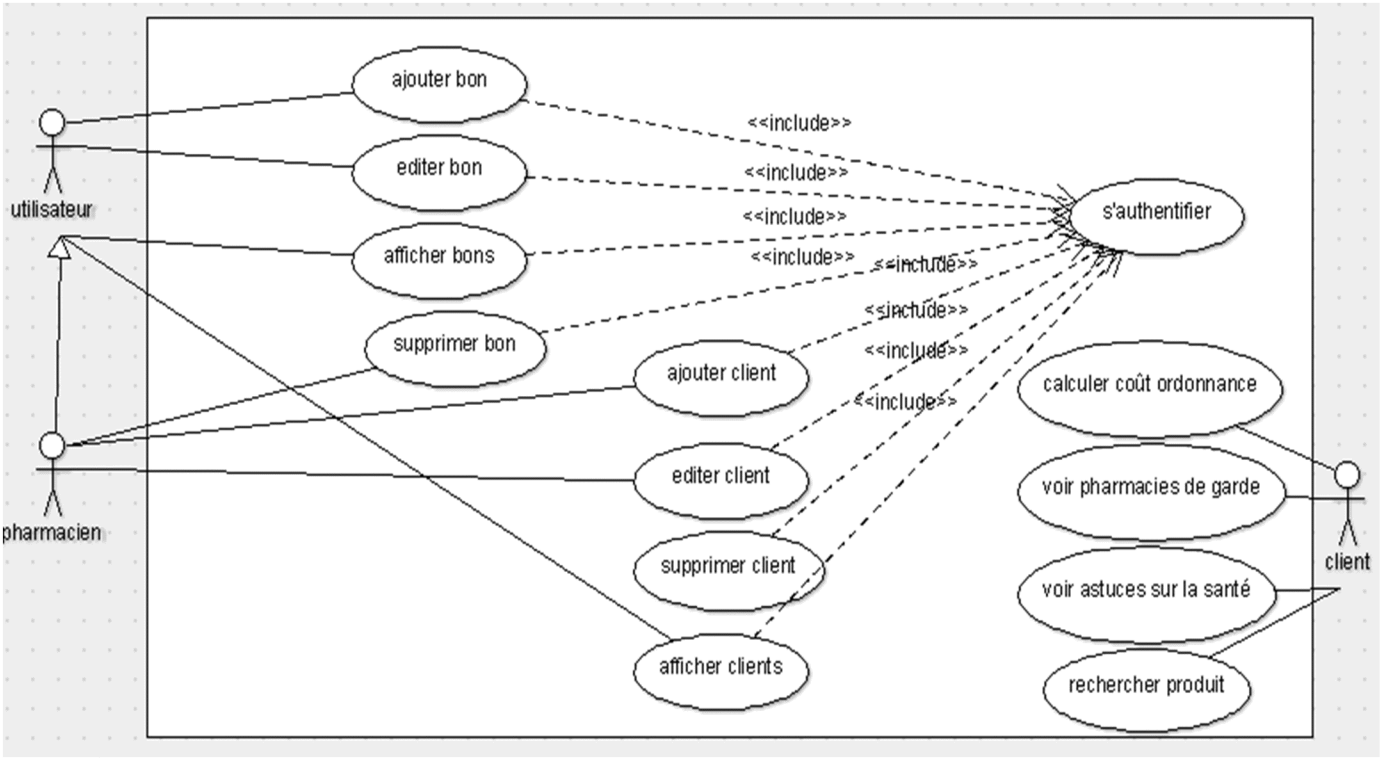


Figure 5 : Diagramme de cas d'utilisation du domaine fonctionnel **Gestion de la clientèle (DF03)**

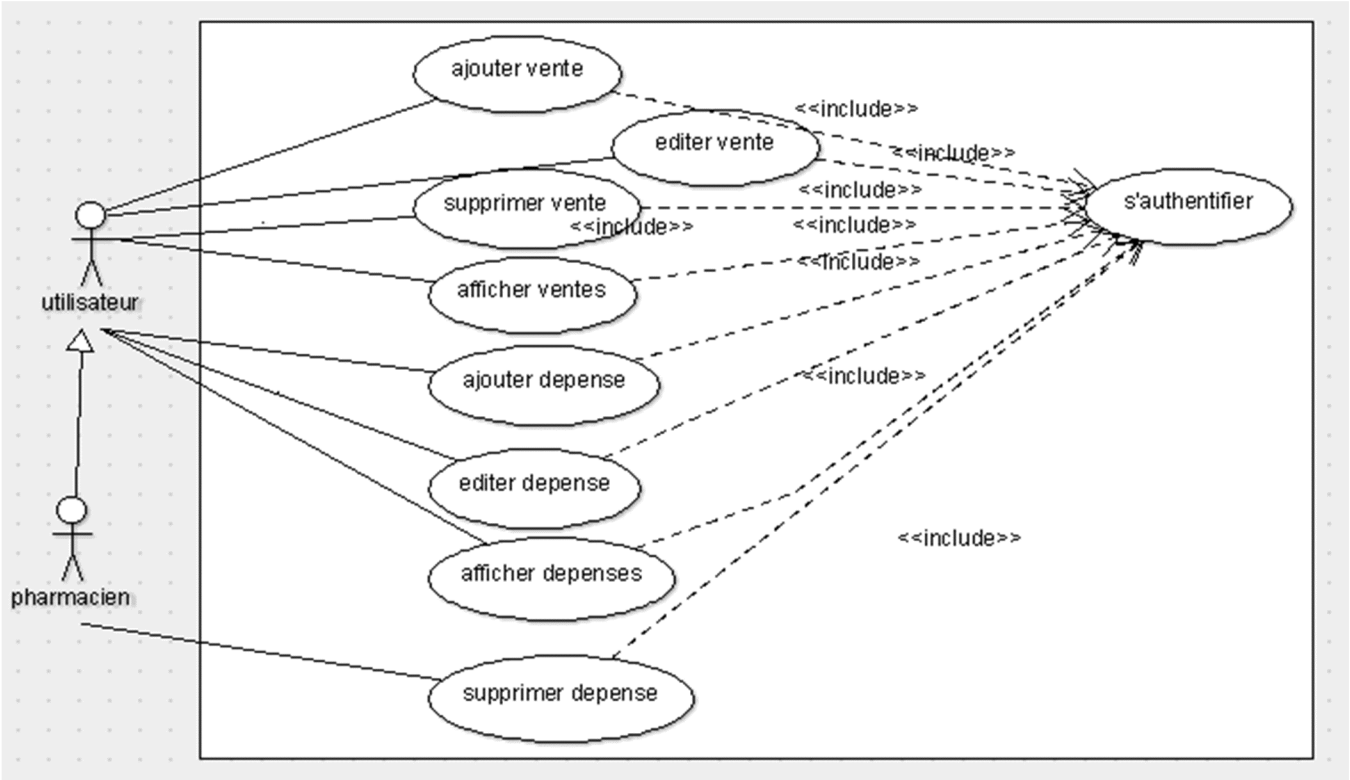


Figure 6 : Diagramme de cas d'utilisation du domaine fonctionnel **Gestion ventes/dépenses (DF04)**

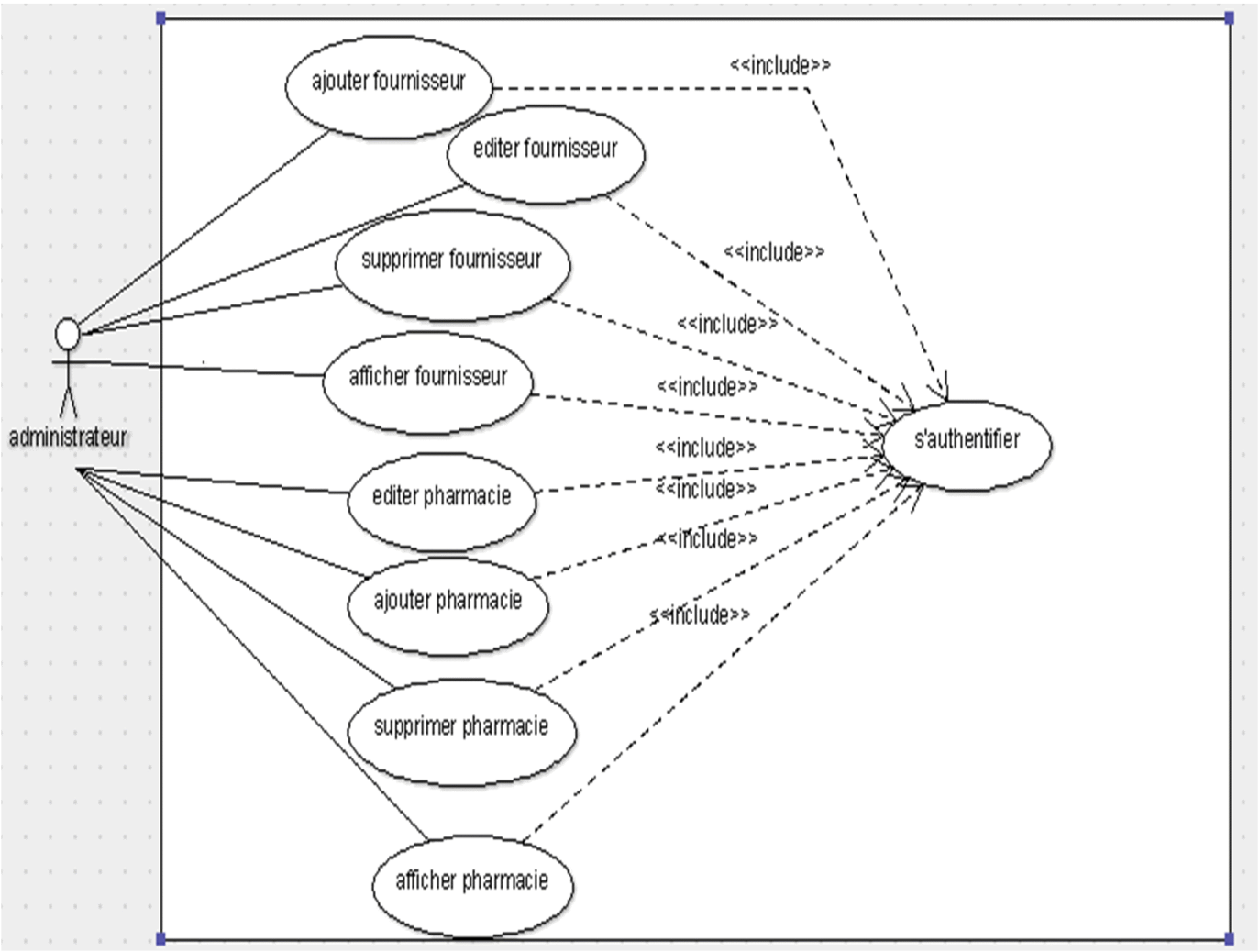


Figure 7 : Diagramme de cas d'utilisation du domaine fonctionnel **Administration (DF05)**

I.2. Description des cas d'utilisation

Pour documenter les cas d'utilisation, la description textuelle est indispensable, car elle permet de communiquer facilement et précisément avec les utilisateurs. Mais, le texte présente des désavantages puisqu'il est difficile de montrer comment les enchaînements se succèdent. Il est donc recommandé de compléter la description textuelle par un ou plusieurs diagrammes dynamiques, qui apporteront un niveau supérieur de formalisation. Ainsi notre description des cas d'utilisation sera divisée en deux parties. Dans la première nous allons faire une description textuelle des différents cas d'utilisation et dans la deuxième partie nous ferons leurs descriptions dynamiques.

I.2.1. Description textuelle des cas d'utilisation

La description textuelle consiste à donner, pour chaque cas d'utilisation, un titre, un résumé, l'acteur(s) concerné(s), la précondition, le scenario nominal, la post condition, etc.

DF01-CU01	
Titre	S'authentifier
Résumé	Ce cas permet de s'identifier avant d'accéder au système.
Acteur (s)	Pharmacien, Employé, Fournisseur, Administrateur
Pré condition	Ajouter compte utilisateur.
Scénario nominal	✓ L'utilisateur saisit son login et son mot de passe. ✓ L'utilisateur valider les informations saisies. ✓ Le système vérifie le login et le mot de passe. ✓ Le système récupère toutes les informations de l'utilisateur. ✓ Le système vérifie le profil de l'utilisateur.
Post condition	Redirection vers la page d'accueil du profil correspondant.
Exception	Login ou un mot de passe incorrecte.

Tableau 2 : Description du cas d'utilisation «s'authentifier »

DF01-CU02	
Titre	Ajouter utilisateur
Résumé	Permet au pharmacien d'ajouter des utilisateurs.
Acteur (s)	Pharmacien
Pré condition	S'authentifier
Scénario nominal	✓ Afficher formulaire d'ajout d'un utilisateur. ✓ Saisir les informations de l'utilisateur. ✓ Valider l'ajout.
Post condition	Utilisateur ajouté avec succès.

Tableau 3 : Description du cas d'utilisation «Ajouter utilisateur »

DF01-CU03	
Titre	Editer utilisateur
Résumé	Permet au pharmacien d'éditer les informations d'un utilisateur.
Acteur (s)	Pharmacien.
Pré condition	S'authentifier, ajouter utilisateur.

Scénario nominal	✓ Choisir l'utilisateur à éditer. ✓ Afficher le formulaire de modification. ✓ Modifier les champs concernés. ✓ Valider la modification.
Post condition	utilisateur édité avec succès.
Exception	Les informations sur l'utilisateur sont mal saisies.

Tableau 4 : Description du cas d'utilisation «Editer utilisateur»

DF01-CU04	
Titre	Supprimer utilisateur
Résumé	Permet au pharmacien de supprimer un utilisateur.
Acteur (s)	Pharmacien
Pré condition	S'authentifier, ajouter utilisateur.
Scénario nominal	✓ Sélectionner l'utilisateur à supprimer. ✓ Valider la suppression.
Post condition	utilisateur supprimée avec succès.

Tableau 5 : Description du cas d'utilisation «supprimer utilisateur»

DF01-CU05	
Titre	Afficher utilisateurs
Résumé	Permet au pharmacien d'afficher les utilisateurs.
Acteur (s)	Pharmacien
Pré condition	S'authentifier, ajouter utilisateur.

Tableau 6 : Description du cas d'utilisation «Afficher utilisateurs »

DF02-CU01	
Titre	Ajouter commande
Résumé	Permet à un utilisateur de passer une commande.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier.
Scénario nominal	✓ Sélectionner les produits à commander. ✓ Spécifier la quantité à commander pour chaque produit. ✓ Valider la commande.
Post condition	Commande lancée avec succès.

Tableau 7 : Description du cas d'utilisation «Ajouter commande »

DF02-CU02	
Titre	Editer commande
Résumé	Permet à un utilisateur d'éditer une commande.
Acteur (s)	Pharmacien, employé.
Pré condition	S'authentifier, ajouter commande.
Scénario nominal	✓ Choisir la commande à éditer. ✓ Afficher le formulaire de saisie. ✓ Modifier les champs concernés. ✓ Valider la modification.
Post condition	commande éditée avec succès.
Exception	Les informations sur la commande sont mal saisies.

Tableau 8 : Description du cas d'utilisation «Editer commande »

DF02-CU03	
Titre	Supprimer commande
Résumé	Permet à un utilisateur de supprimer une commande.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier, ajouter commande.
Scénario nominal	✓ Sélectionner la commande à supprimer. ✓ Valider la suppression.
Post condition	Commande supprimée avec succès.

Tableau 9 : Description du cas d'utilisation «supprimer commande»

DF02-CU04	
Titre	Afficher commande
Résumé	Permet à l'utilisateur d'afficher les commandes.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier, ajouter commande.

Tableau 10 : Description du cas d'utilisation «Afficher commande »

DF02-CU05	
Titre	Ajouter stock
Résumé	Permet à un utilisateur d'ajouter un nouveau stock dans la pharmacie.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier, ajouter produit.

Scénario nominal	✓ Sélectionner le produit que l'on veut ajouter le stock. ✓ Remplir le formulaire d'ajout de stock. ✓ Valider l'ajout de stock.
Post condition	Stock ajouté avec succès.

Tableau 11 : Description du cas d'utilisation «Ajouter stock »

DF02-CU06	
Titre	Editer stock
Résumé	Permet à un utilisateur d'éditer un stock.
Acteur (s)	Pharmacien, employé.
Pré condition	S'authentifier, ajouter stock.
Scénario nominal	✓ Choisir le stock à éditer. ✓ Afficher le formulaire de saisie. ✓ Modifier les champs concernés. ✓ Valider la modification.
Post condition	stock édité avec succès.
Exception	Les informations sur le stock sont mal saisies.

Tableau 12 : Description du cas d'utilisation «Editer stock »

DF02-CU07	
Titre	Supprimer stock
Résumé	Permet à un utilisateur de supprimer un stock.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier, ajouter stock.
Scénario nominal	✓ Sélectionner le stock à supprimer. ✓ Valider la suppression.
Post condition	Stock supprimé avec succès.

Tableau 13 : Description du cas d'utilisation «supprimer stock»

DF02-CU08	
Titre	Afficher stock
Résumé	Permet à l'utilisateur d'afficher les stocks.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier, ajouter stock.

Tableau 14 : Description du cas d'utilisation «Afficher stock »

DF02-CU09	
Titre	Approvisionner les stocks
Résumé	Permet aux utilisateurs d'augmenter la quantité totale d'un stock
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier, ajouter stock.
Scénario nominal	✓ Sélectionner les stocks à approvisionner. ✓ Saisir la quantité pour chaque stock. ✓ Saisir les dates de péremption. ✓ Remplir le formulaire d'approvisionnement. ✓ Valider l'approvisionnement.
Post condition	Stocks approvisionnés avec succès

Tableau 15 : Description du cas d'utilisation «Approvisionner les stocks»

DF02-CU10	
Titre	Editer approvisionnement
Résumé	Permet aux utilisateurs de modifier un approvisionnement
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier
Scénario nominal	✓ Sélectionner l'approvisionnement. ✓ Remplir formulaire de modification. ✓ Valider la modification.
Post condition	Modification faite avec succès

Tableau 16 : Description du cas d'utilisation «Editer approvisionnement»

DF02-CU11	
Titre	Afficher Approvisionnement
Résumé	Permet à l'utilisateur d'afficher les stocks.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier

Tableau 17 : Description du cas d'utilisation «Afficher stock »

DF03-CU01	
Titre	Ajouter bon
Résumé	Permet à un utilisateur d'ajouter un bon pour un client.
Acteur (s)	Pharmacien, Employé

Pré condition	S'authentifier
Scénario nominal	✓ Afficher formulaire d'ajout. ✓ Saisir les informations du client. ✓ Saisir le montant et la date. ✓ Valider la commande.
Post condition	Bon ajouté avec succès.

Tableau 18 : Description du cas d'utilisation «Ajouter bon »

DF03-CU02	
Titre	Editer bon
Résumé	Permet à un utilisateur d'éditer les informations d'un bon.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier, ajouter bon.
Scénario nominal	✓ Choisir le bon à éditer. ✓ Afficher le formulaire de saisie. ✓ Modifier les champs concernés. ✓ Valider la modification.
Post condition	bon édité avec succès.
Exception	Les informations sur le bon sont mal saisies.

Tableau 19: Description du cas d'utilisation «Editer bon »

DF03-CU03	
Titre	Supprimer bon
Résumé	Permet de supprimer un bon.
Acteur (s)	Pharmacien
Pré condition	S'authentifier, ajouter produit, ajouter bon.
Scénario nominal	✓ L'utilisateur choisit le bon à supprimer. ✓ Valider suppression.
Post condition	Bon supprimé avec succès.

Tableau 20 : Description du cas d'utilisation «Supprimer bon »

DF03-CU04	
Titre	Afficher bons
Résumé	Permet à l'utilisateur d'afficher tous les bons.
Acteur (s)	Pharmacien, Employé

Pré condition	S'authentifier, ajouter bon.

Tableau 21 : Description du cas d'utilisation «Afficher bons »

DF03-CU05	
Titre	Ajouter client
Résumé	Permet à un utilisateur d'ajouter un client.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier
Scénario nominal	✓ Afficher formulaire d'ajout. ✓ Saisir les informations du client. ✓ Valider l'ajout.
Post condition	Client ajouté avec succès.

Tableau 22 : Description du cas d'utilisation «Ajouter client »

DF03-CU06	
Titre	Editer client
Résumé	Permet à un utilisateur d'éditer les informations d'un client.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier, ajouter client.
Scénario nominal	✓ Choisir le client à éditer. ✓ Afficher le formulaire de saisie. ✓ Modifier les champs concernés. ✓ Valider la modification.
Post condition	client édité avec succès.
Exception	Les informations sur le client sont mal saisies.

Tableau 23 : Description du cas d'utilisation «Editer client »

DF03-CU07	
Titre	Supprimer client
Résumé	Permet de supprimer un client.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier, ajouter client.
Scénario nominal	✓ L'utilisateur choisit le client à supprimer. ✓ Valider suppression.

Post condition	Client supprimé avec succès.

Tableau 24 : Description du cas d'utilisation «Supprimer produit »

DF03-CU08	
Titre	Afficher clients
Résumé	Permet à l'utilisateur d'afficher tous les clients.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier, ajouter client.

Tableau 25 : Description du cas d'utilisation «Afficher clients »

DF03-CU09	
Titre	Calculer coût ordonnance
Résumé	Permet aux clients de calculer le coût d'une ordonnance.
Acteur (s)	Client
Scénario nominal	✓ Choisir les produits. ✓ Ajouter à l'ordonnance.

Tableau 26 : Description du cas d'utilisation «Calculer coût ordonnance»

DF03-CU10	
Titre	Voir pharmacies de garde.
Résumé	Permet aux clients de voir les pharmacies qui sont de garde.
Acteur (s)	Client
Scénario nominal	✓ Saisir la région.

Tableau 27 : Description du cas d'utilisation «Voir pharmacies de garde»

DF03-CU11	
Titre	Voir astuces sur la santé.
Résumé	Permet aux clients de voir des astuces sur la santé.
Acteur (s)	Client

Tableau 28 : Description du cas d'utilisation «Voir astuces sur la santé»

DF03-CU12	
Titre	Rechercher un produit.
Résumé	Permet aux clients de rechercher un produit dans les différentes pharmacies.
Acteur (s)	Client

Scénario nominal	✓ Saisir le nom du produit. ✓ Valider. ✓ Voir liste des pharmacies.

Tableau 29 : Description du cas d'utilisation «Rechercher un produit»

DF04-CU01	
Titre	Ajouter vente
Résumé	Permet à un utilisateur de vendre des produits.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier
Scénario nominal	✓ Ajouter les produits sur le panier. ✓ Valider la vente.

Tableau 30 : Description du cas d'utilisation «Ajouter vente»

DF04-CU02	
Titre	Editer vente
Résumé	Permet à un utilisateur d'éditer une vente.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier
Scénario nominal	✓ Choisir la vente à éditer. ✓ Afficher le formulaire de saisie. ✓ Modifier les champs concernés. ✓ Valider la modification.
Post condition	vente éditée avec succès.
Exception	Les informations sur la vente sont mal saisies.

Tableau 31 : Description du cas d'utilisation «Editer vente»

DF04-CU03	
Titre	Supprimer une vente
Résumé	Permet de supprimer une vente.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier
	✓ L'utilisateur choisit la vente à supprimer.

Scénario nominal	✓ Valider suppression.
Post condition	Vente supprimée avec succès.

Tableau 32 : Description du cas d'utilisation «Supprimer vente »

DF04-CU04	
Titre	Afficher ventes
Résumé	Permet à l'utilisateur d'afficher toutes les ventes.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier

Tableau 33 : Description du cas d'utilisation «Afficher ventes »

DF04-CU05	
Titre	Ajouter dépense
Résumé	Permet à un utilisateur d'ajouter une dépense.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier
Scénario nominal	✓ Afficher formulaire d'ajout. ✓ Saisir le montant dépensé. ✓ Valider la dépense.
Post condition	dépense ajoutée avec succès.

Tableau 34 : Description du cas d'utilisation «Ajouter vente»

DF04-CU06	
Titre	Editer dépense
Résumé	Permet à un utilisateur d'éditer les informations d'une dépense.
Acteur (s)	Pharmacien, Employé
Pré condition	S'authentifier
Scénario nominal	✓ Choisir la dépense à éditer. ✓ Afficher le formulaire de saisie. ✓ Modifier les champs concernés. ✓ Valider la modification.
Post condition	dépense éditée avec succès.
Exception	Les informations sur la dépense sont mal saisies.

Tableau 35 : Description du cas d'utilisation «Editer dépense »

DF04-CU07	
Titre	Supprimer dépense
Résumé	Permet de supprimer une dépense.
Acteur (s)	Pharmacien
Pré condition	S'authentifier
Scénario nominal	✓ L'utilisateur choisit la dépense à supprimer. ✓ Valider suppression.
Post condition	dépense supprimée avec succès.

Tableau 36 : Description du cas d'utilisation «Supprimer dépense »

DF04-CU08	
Titre	Afficher dépenses
Résumé	Permet à l'utilisateur d'afficher toutes les dépenses.
Acteur (s)	Pharmacien
Pré condition	S'authentifier, ajouter dépense.

Tableau 37 : Description du cas d'utilisation «Afficher dépenses »

DF05-CU01	
Titre	Ajouter fournisseur
Résumé	Permet à un utilisateur d'ajouter un fournisseur.
Acteur (s)	Administrateur
Pré condition	S'authentifier
Scénario nominal	✓ Afficher formulaire d'ajout. ✓ Saisir les informations du fournisseur. ✓ Valider l'ajout.
Post condition	fournisseur ajouté avec succès.

Tableau 38: Description du cas d'utilisation «Ajouter fournisseur»

DF05-CU02	
Titre	Editer fournisseur
Résumé	Permet à l'administrateur d'éditer les informations d'un fournisseur.
Acteur (s)	Administrateur
Pré condition	S'authentifier
Scénario nominal	✓ Choisir le fournisseur à éditer. ✓ Afficher le formulaire de saisie.

	✓ Modifier les champs concernés. ✓ Valider la modification.
Post condition	fournisseur édité avec succès.
Exception	Les informations sur le fournisseur sont mal saisies.

Tableau 39 : Description du cas d'utilisation «Editer fournisseur »

DF05-CU03	
Titre	Supprimer fournisseur
Résumé	Permet de supprimer un fournisseur.
Acteur (s)	Administrateur
Pré condition	S'authentifier, ajouter fournisseur.
Scénario nominal	✓ L'utilisateur choisit le fournisseur à supprimer. ✓ Valider suppression.
Post condition	fournisseur supprimé avec succès.

Tableau 40 : Description du cas d'utilisation «Supprimer fournisseur

DF05-CU04	
Titre	Afficher fournisseurs
Résumé	Permet à l'administrateur d'afficher tous les fournisseurs.
Acteur (s)	Administrateur.
Pré condition	S'authentifier

Tableau 41 : Description du cas d'utilisation «Afficher fournisseurs »

DF05-CU05	
Titre	Ajouter pharmacie
Résumé	Permet à un utilisateur d'ajouter une pharmacie.
Acteur (s)	Administrateur
Pré condition	S'authentifier
Scénario nominal	✓ Afficher formulaire d'ajout. ✓ Saisir les informations de la pharmacie. ✓ Valider l'ajout.
Post condition	pharmacie ajoutée avec succès.

Tableau 42: Description du cas d'utilisation «Ajouter pharmacie»

DF05-CU06	
Titre	Editer pharmacie
Résumé	Permet à l'administrateur d'éditer les informations d'une pharmacie.
Acteur (s)	Administrateur.
Pré condition	S'authentifier, ajouter pharmacie.
Scénario nominal	✓ Choisir la pharmacie à éditer. ✓ Afficher le formulaire de saisie. ✓ Modifier les champs concernés. ✓ Valider la modification.
Post condition	pharmacie éditée avec succès.
Exception	Les informations sur la pharmacie sont mal saisies.

Tableau 43 : Description du cas d'utilisation «Editer pharmacie »

DF05-CU07	
Titre	Supprimer pharmacie
Résumé	Permet de supprimer une pharmacie.
Acteur (s)	Administrateur
Pré condition	S'authentifier
Scénario nominal	✓ L'utilisateur choisit la pharmacie à supprimer. ✓ Valider suppression.
Post condition	Pharmacie supprimée avec succès.

Tableau 44 : Description du cas d'utilisation «Supprimer pharmacie »

DF05-CU08	
Titre	Afficher pharmacies
Résumé	Permet à l'administrateur d'afficher toutes pharmacies.
Acteur (s)	Administrateur.
Pré condition	S'authentifier

Tableau 45 : Description du cas d'utilisation «Afficher fournisseurs »

I.2.2. Description dynamique par des diagrammes d'activité de quelques cas d'utilisation

Comme nous l'avons dit en I.2, la description textuelle est indispensable, car elle permet de communiquer facilement et précisément avec les utilisateurs. Mais, le texte présente des désavantages puisqu'il est difficile de montrer comment les enchaînements se succèdent. Alors

nous allons compléter notre description textuelle par des diagrammes d'activités de quelques cas d'utilisation de notre système.

Diagramme d'activités pour l'authentification

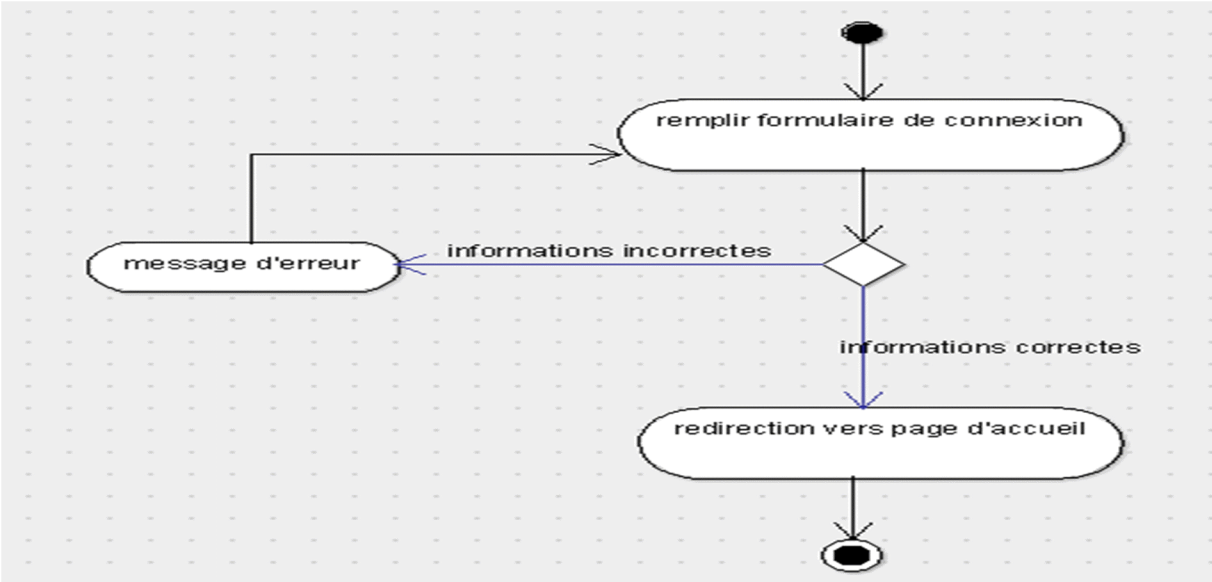


Figure 8 : Diagramme d'activités pour l'authentification.

Diagramme d'activités pour l'ajout d'un produit

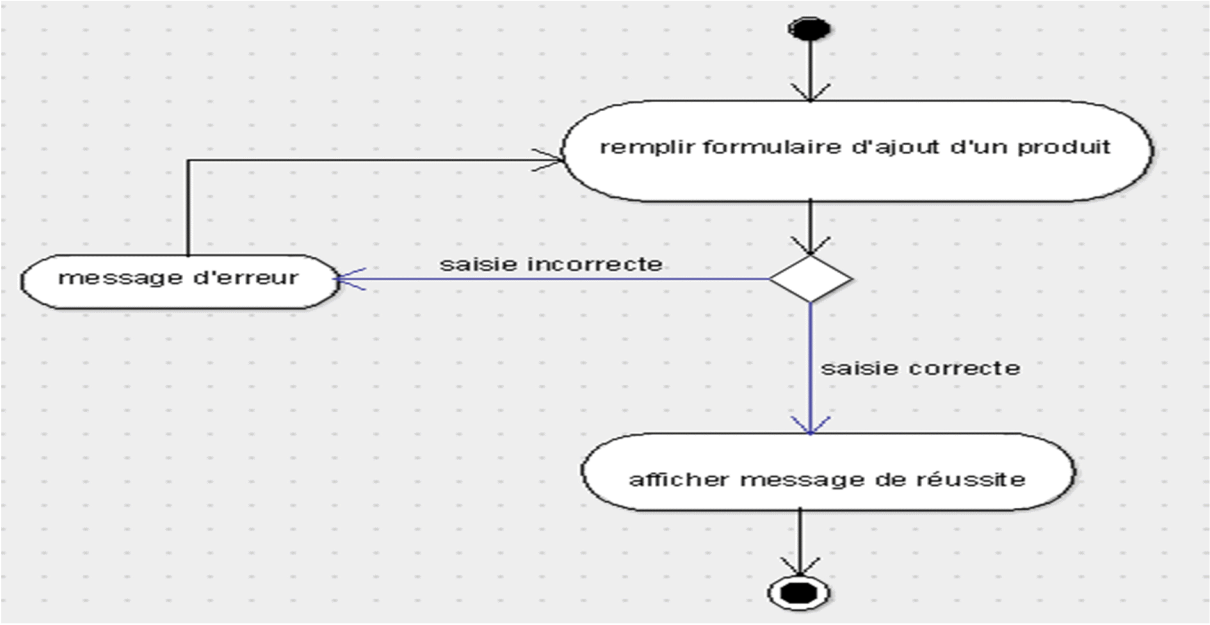


Figure 9 : Diagramme d'activités pour l'ajout d'un produit

NB : Presque toutes les activités d'ajout se font de la même façon.

Diagramme d'activités pour la modification d'un produit

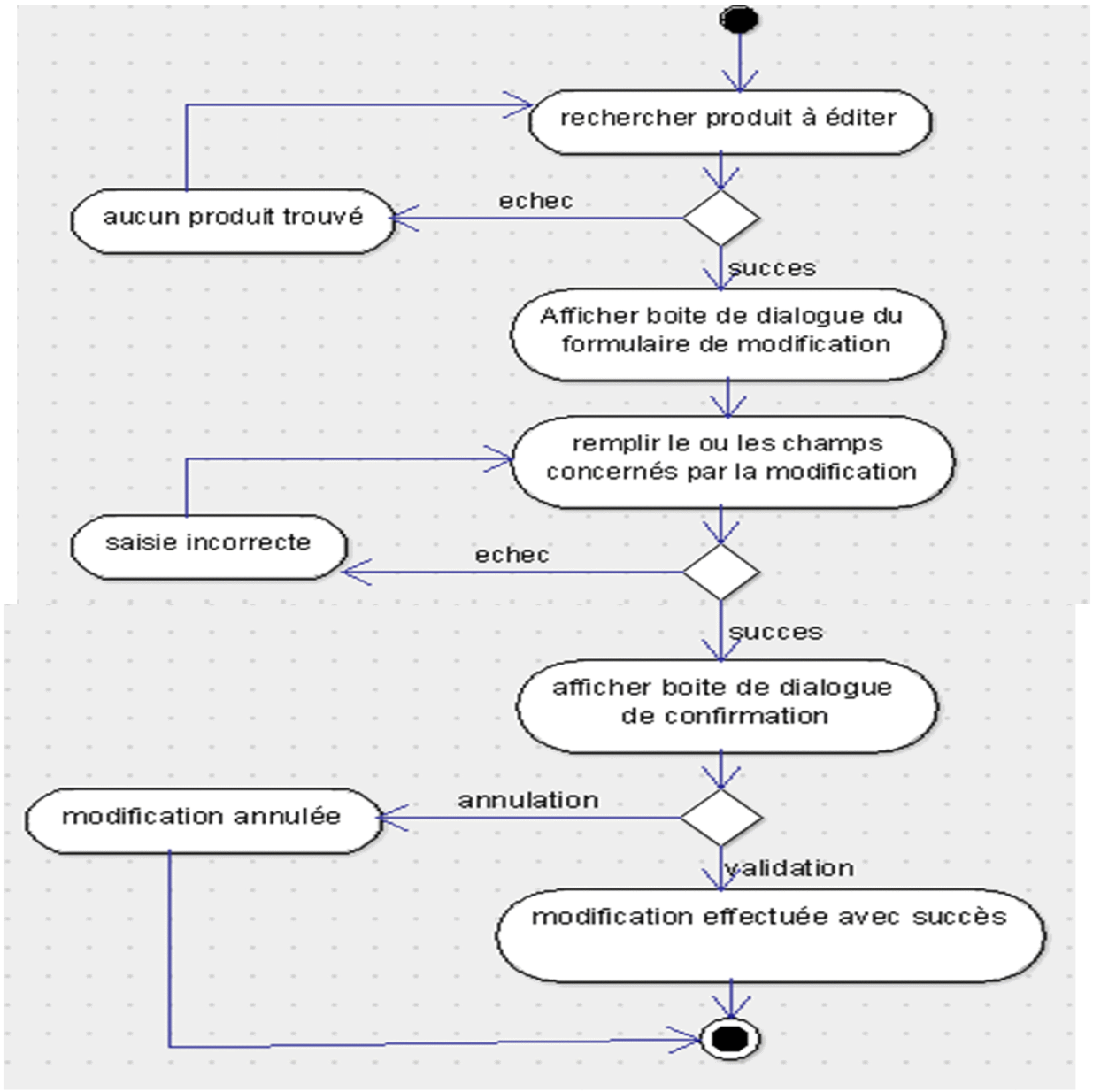


Figure 10 : Diagramme d'activités pour la modification d'un produit

NB : Presque toutes les activités de modification se font de la même façon.

I.3. Classes participantes

Cette partie nous permettra de cibler toutes les classes qui vont participer à l'analyse objet et pouvant éventuellement figurer dans le diagramme de classes complet.

- **Classes participantes à la fonctionnalité « approvisionner »**

  Une pharmacie s'approvisionne plusieurs fois. Un approvisionnement est fait après une livraison et peut concerner plusieurs Stocks et éventuellement leurs SousStock.

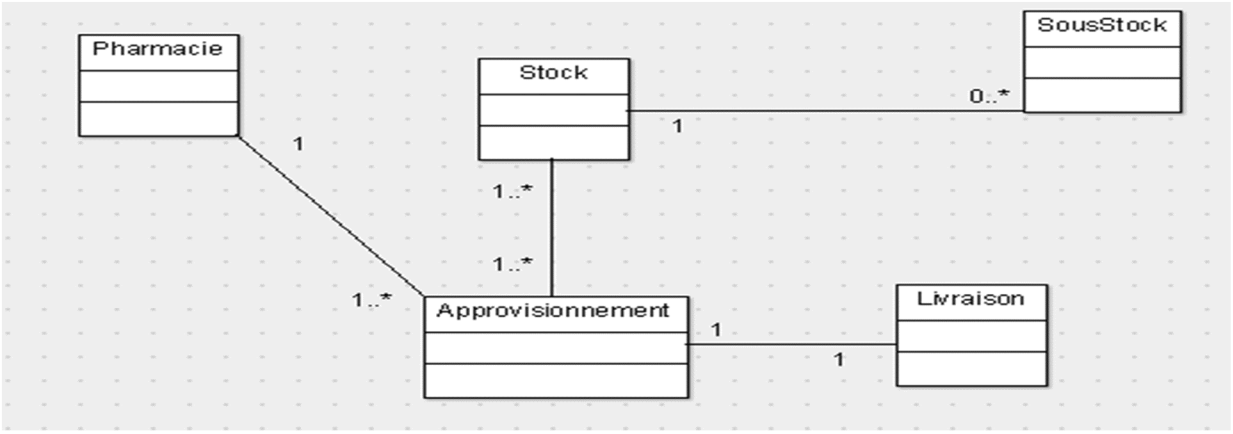


Figure 11 : Diagramme de classes participantes à l'approvisionnement.

- **Classes participantes à la fonctionnalité « vendre »**

  Un utilisateur de la pharmacie peut faire plusieurs ventes. Une vente peut concerner plusieurs Stocks et éventuellement leurs SousStock.

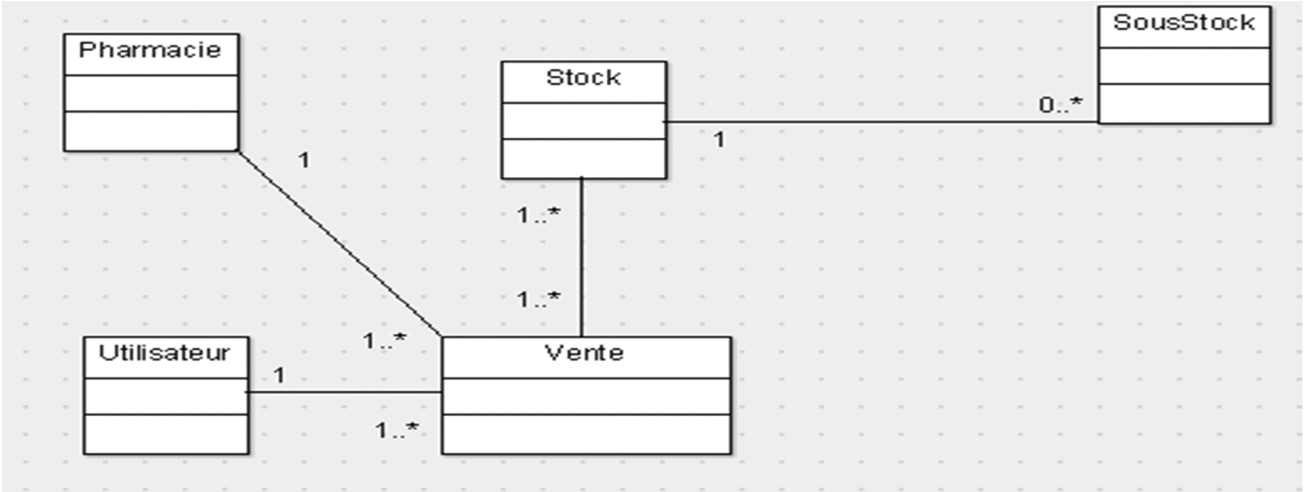


Figure 12: Diagramme de classes participantes à la vente.

- **Classes participantes à la fonctionnalité « Rechercher produit »**

Un produit est recherché dans les Stocks de la Pharmacie.

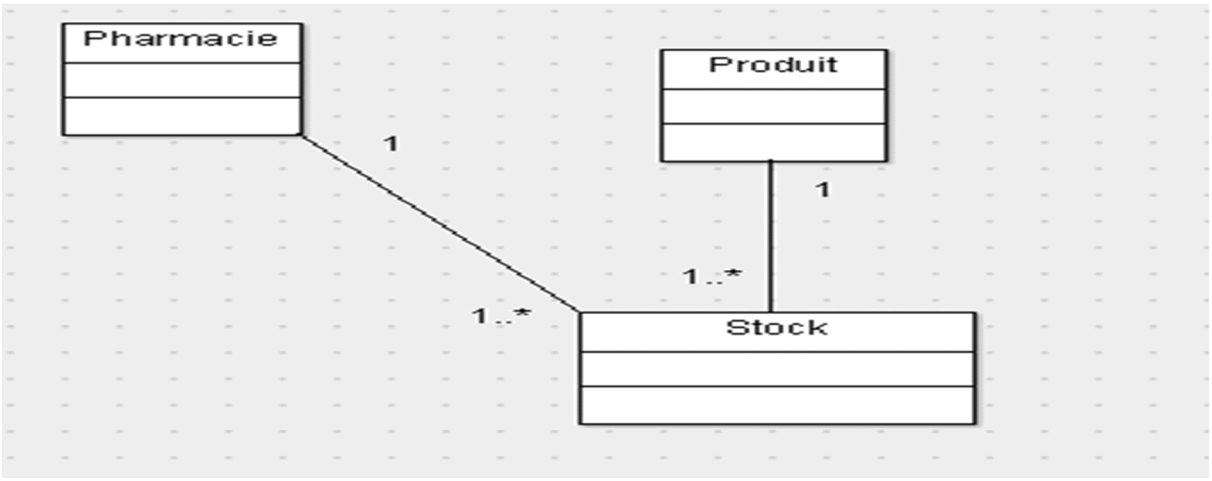


Figure 13 : Diagramme de classes participantes à la recherche de produit.

II. Les besoins techniques

La capture des besoins techniques est une activité de la branche gauche du processus de développement (2TUP). Elle occupe une étape très importante dans le développement d'un système. En effet, cette partie traite toutes les contraintes techniques pour la mise en place de notre système.

II.1. Architecture de l'application

Compte tenu du nombre important de pharmacies au Sénégal, des problèmes liés aux mises à jour, nous avons adopté l'architecture trois tiers (**n-tiers**). Cette architecture physique est assez semblable à l'architecture client/serveur, mais en plus des « clients» et du serveur de données, un serveur d'application intervient comme un troisième tiers. Les machines clientes, également appelées «clients légers» ne contiennent que l'interface de l'application. Elles sont déchargées de tout traitement. Ainsi tout le traitement est assuré par le serveur d'application, qui sert de liaison entre l'interface applicative et les données.

Elle offre :

- ✓ De bonnes performances grâce à la répartition des charges de travail ;
- ✓ Une disponibilité de l'application et de l'information en temps réel.

A cette architecture physique, nous ajoutons une méthode appelée MVC (**Model-View-Controller**). C'est une méthode de conception qui organise l'interface homme-machine d'un logiciel. Cette méthode simplifie les opérations de maintenance et de mise à jour, facilite la tâche du développeur et permet de faire un bon découpage du travail.
Elle repose sur la séparation du traitement, des données et de la présentation d'où MVC.

Notre système doit aussi répondre aux critères suivants :

- *La convivialité :* Les interfaces utilisateurs doivent être ergonomiques, simples à maitriser et conforme au langage des pharmaciens.
- *La rapidité de traitement* : compte tenu du nombre important de transactions journalières dans chaque pharmacie, notre application doit être rapide dans le traitement des transactions.
- *La disponibilité :* L'application doit être disponible tous les jours et toutes les heures afin que les pharmacies de garde puissent travailler.

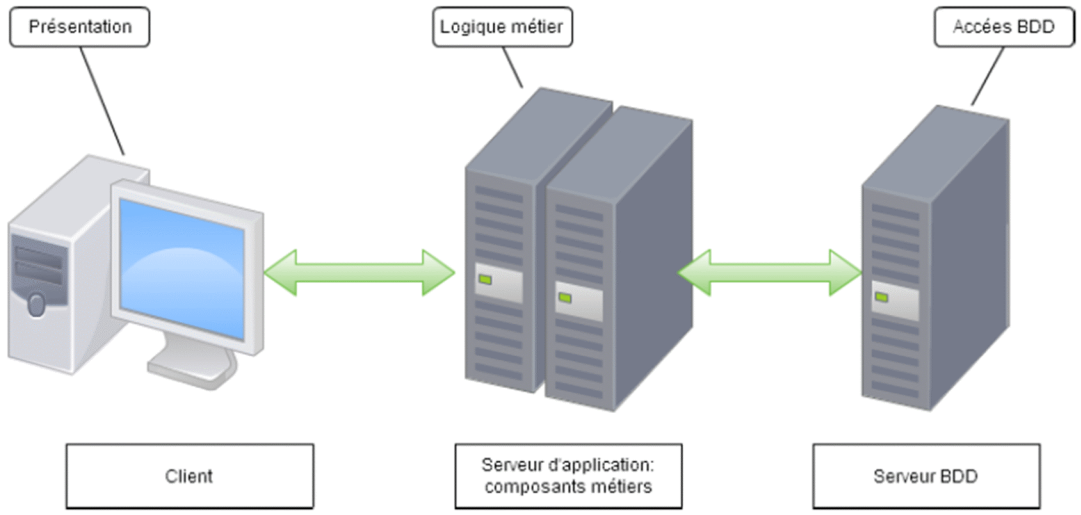


Figure 14 : Architecture trois tiers.

II.2. Environnement d'exécution

Le choix de l'architecture influe sur le choix de l'environnement d'exécution de notre application. En effet, l'architecture nécessite :

- Un ordinateur, une tablette ou un mobile équipé d'un navigateur web pour **les clients**.
- Un ordinateur avec une importante RAM et une importante capacité de stockage pour le **serveur de SGBDR**.
- Un ordinateur disponible à tout moment pour le **serveur Web**.
- Un ordinateur disponible à tout moment et rapide dans l'exécution des méthodes métiers pour le **serveur d'application.**

Mais présentement, un seul ordinateur fait office de serveur web, de serveur de SGBDR et de serveur d'application. Ses caractéristiques sont les suivants :

Processeur **: 1.30 GHz**.

Mémoire RAM : **4 Go**.

Système d'exploitation : **64 bits**.

Après ce long chapitre dans lequel nous avons capturé les besoins fonctionnels et techniques de notre système, nous allons pouvoir attaquer notre chapitre suivant qui est essentiellement basé sur l'analyse orientée objet.

Chapitre 3 : Analyse et Conception

I. Analyse

La phase d'analyse représente la deuxième étape de la branche gauche du cycle en Y.

Après capture et analyse des besoins, nous avons obtenu un découpage fonctionnel exprimé à travers les cas d'utilisation (le modèle de spécification fonctionnelle). Pour passer à la phase d'analyse, nous allons changer complétement l'organisation du modèle et nous fonder sur les principes de l'approche orientée objet. A cet effet, nous allons passer d'une structuration fonctionnelle via les cas d'utilisation et les packages de cas d'utilisation, domaines fonctionnels dans notre cas, à une structuration objet via les classes. Notre analyse se divisera en deux parties. Dans la première partie nous allons découper nos classes en catégories et dans la seconde partie nous allons développer notre modèle statique

I.1. Découpage en catégories

Le découpage en catégories vient juste après la capture des besoins fonctionnels. Ainsi, il représente la première partie et le début de l'analyse objet. Il permet de structurer les classes candidates en les regroupant dans des ensembles. Nos classes sont divisées en deux catégories, voir image ci-dessous.

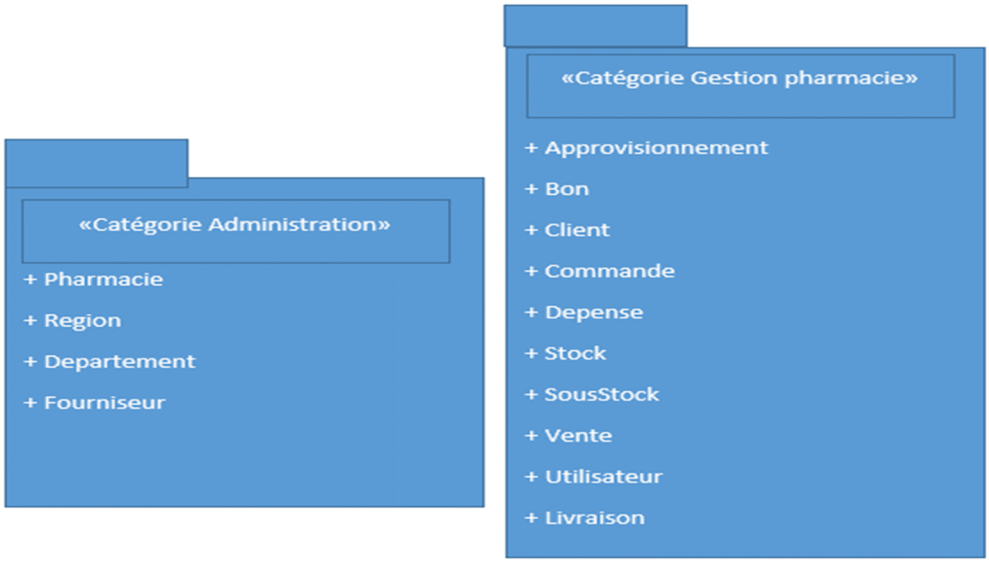


Figure 15: Découpage en catégories des classes.

I.2. développement du modèle statique

Le développement du modèle statique constitue la deuxième activité de l'étape d'analyse. Les classes candidates, identifiées lors de la capture des besoins fonctionnels, réorganisées ensuite lors du découpage en catégories vont être étudiées en détail. Cette étude pourrait aboutir à la suppression ou à l'ajout de certaines classes.

- La classe « Produit » : Selon la description faite au niveau de la capture des besoins, nous obtenons les attributs du tableau ci-dessous.

Attributs	Descriptions
DCI	La Dénomination Commune Internationale
Forme	Comprimé, sirop, gélule, ovule, suppositoire, injection, crème, pommade, ampoule, sachet, usage externe
Dosage	La quantité des éléments actifs présents dans le produit (en gramme le plus souvent)
Tableau	Classement du produit
Libellé	Désignation du produit
Prix cession	Prix de vente pour les pharmaciens
Prix public	Prix de vente pour le public

Tableau 46 : Attributs de « Produit » selon la capture des besoins.

- Après de la classe « Produit » affinement on obtient les attributs du tableau ci-dessous.

Attribut	Type	Description
id_medicament	Long	Clé primaire de la classe « Produit »
dci_medicament	String	DCI du produit.
libelle_medicament	String	Désignation du produit de: la forme cetamyl 400mg cp b/x.
prix_calculatrice	String	Prix fixé pour la calculatrice.
tableau_medicament	String	Le classement du produit.
perisable	String	Renseigne sur la péremption du produit.

Tableau 47 : La classe « Produit » après affinement.

- **Stock** : la classe « Stock » est une autre appellation de la classe « Produit » après une mise en place du produit dans la pharmacie. Elle hérite certains attributs de « Produit » et les combine avec ceux du tableau ci-dessous.

Attribut	Type	Description
id_stock	Long	La clé primaire de la classe Stock
quantite_totale	int	représente le nombre total d'échantillons du produit dans la pharmacie.
prix_unitaire	double	
quantite_minimale	int	si cette quantité est atteinte alors ajouter le produit à la commande.
prix_cession	String	Prix de vente pour les pharmacies

Tableau 48 : « Stock » après affinement.

- **SousStock** : Comme il existe des stocks qui peuvent avoir des échantillons de date de péremptions différentes alors nous ajoutons la classe SousStock pour contrôler la péremption.

Attribut	Type	Description
idSousStock	Long	
datePeremption	String	
quantite	int	
valide	boolean	

Tableau 49: « SousStock » après affinement.

- **Commande** :

Attribut	Type	
idCommande	Long	
date	String	
Etat	String (traitée, non traitée)	

Tableau 50: « Commande » après affinement.

- **Vente** :

Attribut	Type	Description
id_vente	Long	
date_vente	String	
total_ventes	String	
loginVendeur	String	
etatVente	String (fermée, ouverte)	

Tableau 51: « Vente » après affinement.

- **Livraison** : Cette classe est supprimée car la livraison n'est pas prise en charge.
- Les classes **DetailsApprovisionnement**, **DetailsBon**, **DetailsCommande**, **DetailsVente** sont des classes de jointures entre deux entités liées par la cardinalité **n : m** (voir annexes).

NB : voir annexes sur les entités pour les relations et les autres classes.

II. Conception

La conception est la phase où s'effectue la fusion de toutes les études faites précédemment. Elle se divise en trois parties qui sont : la conception générique, la conception préliminaire et la conception détaillée. Cependant, dans notre cas, nous allons faire cette conception sans nous situer dans ses différentes parties. Dans un premier temps nous allons concevoir notre modèle logique (classes, associations, attributs) et dans un second temps concevoir notre diagramme de composants.

II.1. Le modèle logique

La conception du modèle logique consiste à concevoir les classes, concevoir les associations, concevoir les attributs et les opérations (optionnel) comme le montre la **Figure16** ci-dessous. Cependant, dans notre cas, nous allons représenter le diagramme de classes complet de notre système sans illustrer les différentes étapes, représentées sur la **Figure16,** de sa conception.

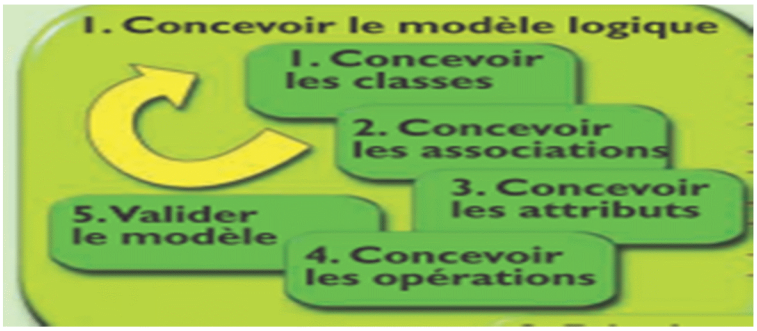
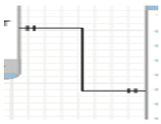
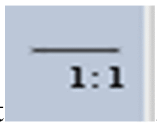
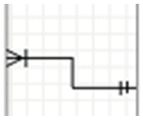
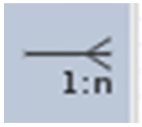


Figure 16 : Etapes du modèle logique **[8].**

 : Relation entre classes représentant 

.

 : Relation entre classes représentant 

: Relation entre classes représentant **n:m**

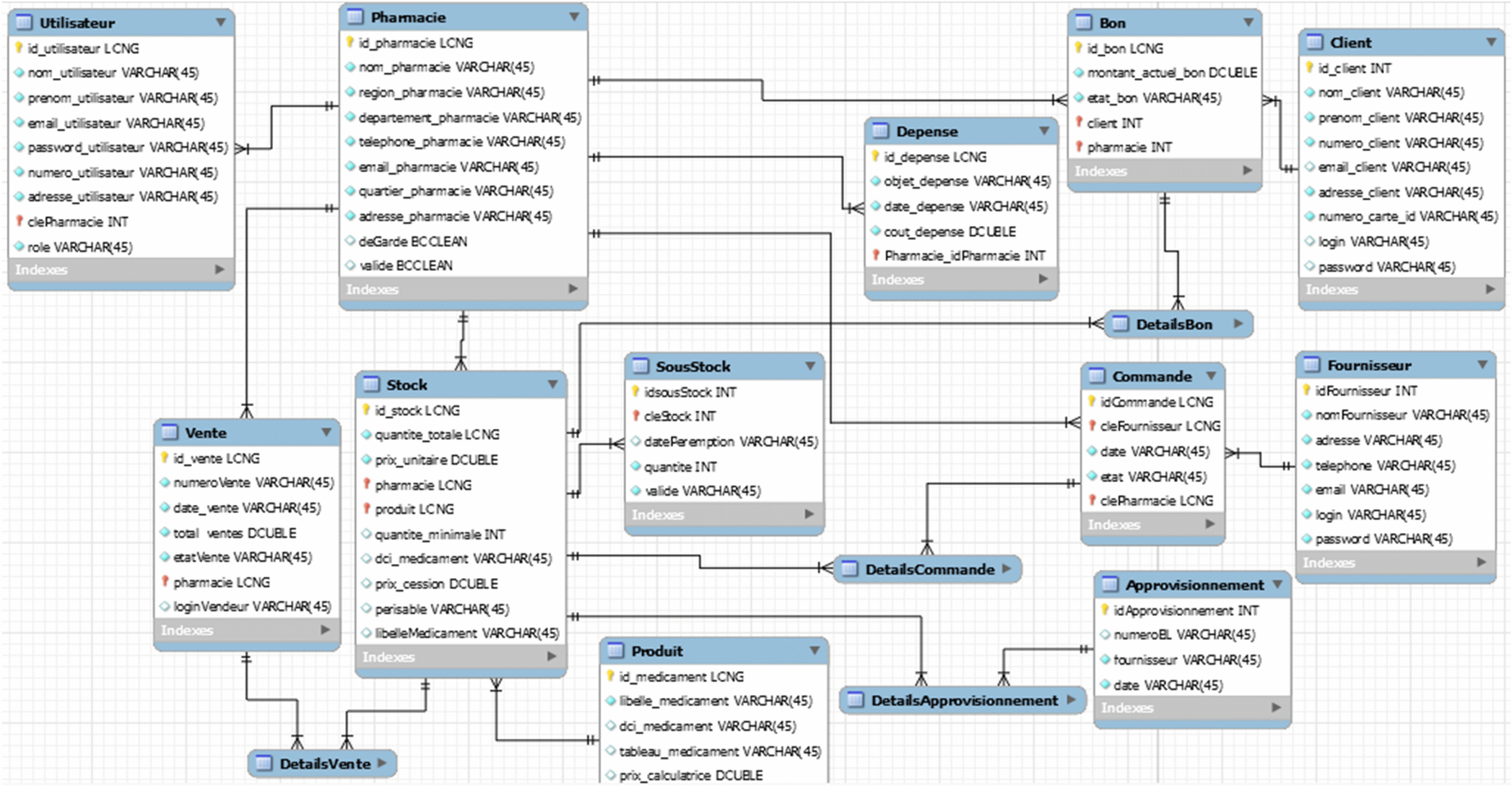
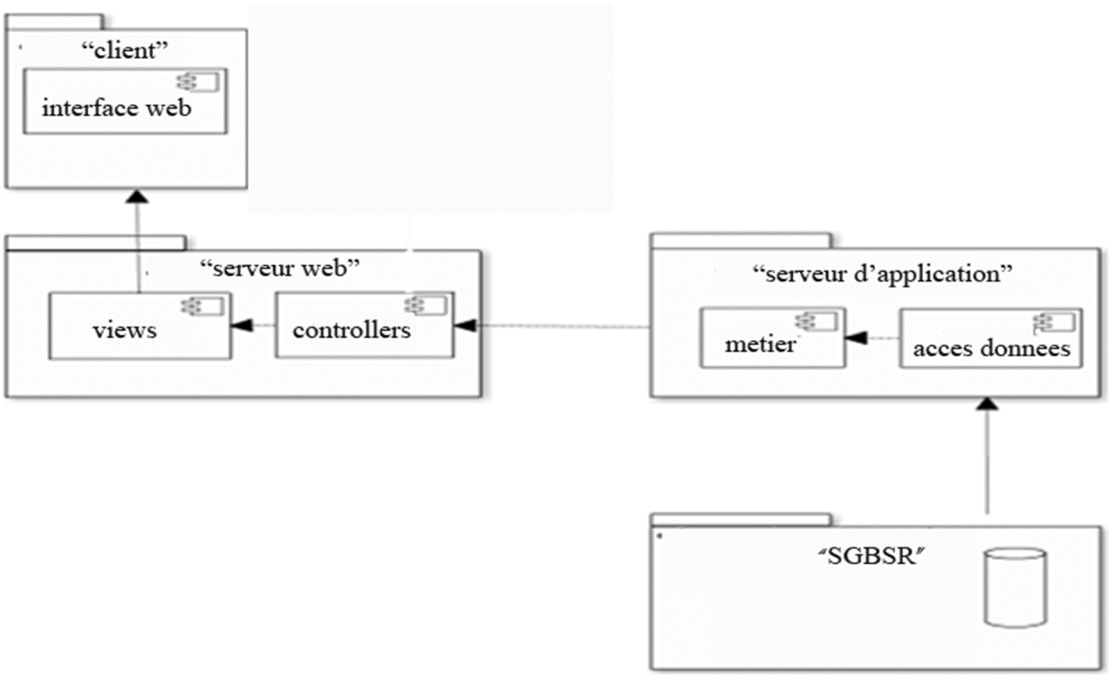


Figure 17 : Diagramme de classes.

II.2. Diagramme de composants

Le diagramme de composants définit l'architecture logicielle du système. Il permet de représenter le système et les sous-systèmes du modèle physique de l'architecture logicielle à réaliser. Ce diagramme permet de mettre en évidence les dépendances entre les composants.



.

Figure 18 : Diagramme de composants.

Chapitre 4 : Codage et présentation de l'application

I. Codage

Situé sur la branche du milieu de notre processus de développement, le codage représente une partie importante dans la mise en place d'un système. Nous allons, dans cette étape, faire d'abord une description des outils utilisés ensuite illustrer quelques parties de la couche métier et enfin présenter notre application par quelques captures d'écran.

I.1. Description des outils utilisés

Pour développer notre application nous avons utilisé plusieurs outils et nous allons faire une petite présentation de ces derniers.

✓ **Eclipse**

Eclipse est une plate-forme universelle pour des environnements de développement intégrés. Elle est fondée sur une architecture ouverte et extensible. C'est un logiciel gratuit portable, car écrit en Java, indépendant de tout langage et convivial.

✓ **Postgresql**

PostgreSQL est un système de gestion de bases de données relationnel robuste et puissant, aux fonctionnalités riches et avancées, capable de manipuler en toute fiabilité de gros volumes de données, mêmes dans des situations critiques [8].

.

Figure 19 : postgresql

✓ **Plate-forme JEE**

- ***Les composants et le serveur Java EE utilisés***

Les développements Java EE reposent sur un découpage en couches ou tiers, nous parlons alors d'applications multi-tiers. Trois grands tiers sont représentés : La couche présentation (tiers Web), la couche métier (tiers Métier ou tiers Business), la couche stockage des informations (tiers Enterprise Information System).

Composant web ou tiers web : Java Server Faces (JSF)

La technologie JavaServer Faces est construite à partir de Servlets et fournit un framework de développement pour accélérer la création d'applications Web.

Cette framework est basée sur la notion de composants. Une page JSF est une page xhtml (ou jsp) liée aux Managed Bean via le language EL.

A cette technologie nous avons ajouté d'autres outils comme :

Primefaces : bibliothèque de composants graphiques qu'on peut utiliser avec JSF pour gagner en productivité et efficacité.

Bootstrap : Framework CSS pour nous permettre d'avoir une application responsive.

BootsFaces : Du Bootstrap pour JSF.

Spring security : framework d'authentification et de contrôle d'accès. Il gère **L'authentification** qui consiste à garantir que la personne connectée est bien celle qu'elle prétend être et les **autorisations** qui consistent à vérifier que la personne connectée a bien les permissions d'effectuer une action donnée ou d'accéder à une ressource.

Composant métier ou tiers métier : Entreprises JavaBeans (EJB)

Un EJB est un composant Java EE exécuté par un conteneur spécifique dans une machine virtuelle java. Ce composant est une archive JAR (Java Archive), contenant un ou plusieurs objets java particuliers appelés Beans, possédant et implémentant un certain nombre d'interfaces. Ces Beans permettent au programmeur de travailler sur une base de données sans se soucier d'un quelconque SGBDR car un EJB Possède un langage de requêtes qui lui est propre appelé EJB-QL. Il possède un ou plusieurs fichiers de configuration XML (persistence.xml) et s'intègre dans une architecture trois tiers.

Un Bean est un objet Java particulier qui hérite d'une interface définissant un ensemble de méthodes gérant son cycle de vie au sein du conteneur. La spécification EJB propose différents types de Beans ayant chacun une tâche qui lui est propre.

Les Beans utilisés :

Les Entity Bean qui sont la représentation objet des données présentes en base de données. Pour être simple un Entity correspond à une table en base de données. La modification de cet objet

engendre des modifications sur les données correspondantes en base de données. ***Les Sessions Bean*** qui accomplissent des tâches spécifiques pour un client distant ou local. Et en fonction des besoins l'état d'un Bean peut être conservé ou non d'où l'existence de deux types de Bean Session : Statefull (avec état) et Stateless (sans état).

Serveur: Jboss as 7.1.1.Final

Les serveurs Java EE proposent plusieurs types de conteneurs (containers en anglais). Chaque conteneur a un rôle bien défini et offre un ensemble de services. Une application Java EE de type Web nécessite un conteneur Web pour son exécution alors qu'une application utilisant les EJB nécessite un conteneur EJB.

Nous utilisons dans notre cas un serveur dans lequel les deux types d'application (WEB et EJB) peuvent s'exécuter en même temps : **Jboss as.**

JBoss Application Server ou JBoss AS, est un serveur d'applications Java EE Libre, écrit en Java, publié sous licence GNU LGPL. Étant écrit en Java, il peut être utilisé sur tout système d'exploitation fournissant une machine virtuelle Java (JVM).

I.2. Codage de la couches métier

Pour la partie métier nous allons créer un projet EJB. Comme tout projet EJB, nous aurons à créer les interfaces (Locale et Remote), implémenter ces deux interfaces et créer les entités (classes persistantes).

- ***Codage d'une entité***

Avec JPA (Java Persistence API) les programmeurs n'ont plus besoin d'être des spécialistes en base de données ou d'un SGBD quelconque. JPA leur permet de programmer les classes (entités) qui seront transformées en tables dans une base de données. JPA permet aux programmeurs d'utiliser des notions avancées en base de données tout en restant programmeurs. Seule une entité sera codée dans cette partie (voir annexes pour les autres).

```java
@Entity
public class Pharmacie implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id_pharmacie;
    private String nom_pharmacie;
    private String telephone_pharmacie;
    private String email_pharmacie;
    private String region_pharmacie;
    private String departement_pharmacie;
    private String quartier_pharmacie;
    private String adresse_pharmacie;

    /**
     * Gestion de la relation entre la Pharmacie et ses Utilisateurs.
     */
    @OneToMany(mappedBy="pharmacie")
    private List<Utilisateur> liste_des_utilisateurs = new ArrayList<Utilisateur>();
    /**
     * Gestion de la relation entre la Pharmacie et ses Stocks.
     */
    @OneToMany(mappedBy="pharmacie")
    private  List<Stock>liste_des_stocks = new ArrayList<Stock>();
    /**
     * Gestion de la relation entre la Pharmacie et ses Commandes.
     */
    @OneToMany(mappedBy="pharmacie")
    private List<Commande>liste_des_commandes = new ArrayList<Commande>();

```

Figure 20 : Codage d'une entité

@Entity signifie que cette classe est une table. JPA créera une table, de même nom que la classe, dans notre base de données.

@Id signifie que l'attribut qui est en dessous (id_pharmacie) est la clé primaire de la table.

@GeneratedValue signifie que la clé primaire est automatiquement générée par le SGBD.

@OneToMany signifie que cette classe (table) est liée à plusieurs Utilisateurs (1 à plusieurs).

- **Codage des interfaces et implémentation**

```java
@Local
public interface Interface_Locale {

   //Pharmacie
      public void ajouter_pharmacie(Pharmacie pharmacie);
      public List<Pharmacie>listeDesPharmacies();
      public void supprimer_pharmacie(Long idPharmacie);
      public Pharmacie avoir_pharmacie(Long idPharmacie);
      //medicament
      public void ajouter_medicament(Medicament medicament);
      public List<Medicament>listeDesMedicament();
      public Medicament avoir_medicament(String medicament);
      public Medicament avoir_medicament_by_id(Long idMedicament);
      //localite
      public void ajouter_region(Region region);
      public List<Region>listeDesRegions();
      public void ajouter_departement(Departement departement);
      public List<Departement>listeDesDepartements();
      public List<Departement>listeDesDepartementsRegion(Long region);
      public Region avoir_region(String region);
```

Figure 21 : Codage de l'interface locale

```
@Remote
public interface Interface_Distante {

   //Pharmacie
         public void ajouter_pharmacie(Pharmacie pharmacie);
         public List<Pharmacie>listeDesPharmacies();
         public void supprimer_pharmacie(Long idPharmacie);
         public Pharmacie avoir_pharmacie(Long idPharmacie);
         //medicament
         public void ajouter_medicament(Medicament medicament);
         public List<Medicament>listeDesMedicament();
         public Medicament avoir_medicament(String medicament);
         public Medicament avoir_medicament_by_id(Long idMedicament);
         //localite
         public void ajouter_region(Region region);
         public List<Region>listeDesRegions();
         public void ajouter_departement(Departement departement);
         public List<Departement>listeDesDepartements();
         public List<Departement>listeDesDepartementsRegion(Long region);
         public Region avoir_region(String region);
```

Figure 22 : Codage de l'interface distante

```
@Stateless(name="SAMA_PHA")
public class Implementation_Interfaces implements Interface_Locale, Interface_Distante{

   @PersistenceContext(unitName="UP_MEMOIRE")
   private EntityManager samaEM;


   @Override  public void ajouter_pharmacie(Pharmacie pharmacie) {
      samaEM.persist(pharmacie);
   }

   @Override
   public List<Pharmacie> listeDesPharmacies() {
      Query requete = samaEM.createQuery("select c from Pharmacie c");
      return requete.getResultList();
   }

   @Override
   public void supprimer_pharmacie(Long idPharmacie) {
      // TODO Auto-generated method stub
```

Figure 23 : Implémentation des deux interfaces

@PersistenceContext permet de mettre en relation notre contrôleur à notre base de données.

- ***Présentation du fichier « persistence.xml »***

persistence.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <persistence xmlns="http://java.sun.com/xml/ns/persistence"
4            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5            xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
6            http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
7            version="1.0">
8
9        <persistence-unit name="UP_MEMOIRE">
10          <jta-data-source>java:/memoireDS</jta-data-source>
11          <properties>
12             <property name="hibernate.hbm2ddl.auto" value="create"/>
13          </properties>
14       </persistence-unit>
15
16 </persistence>
```

Figure 24 : fichier persistence.xml

Le fichier de persistance ci-dessus permet de mettre en relation notre projet EJB et notre data source.
La valeur « UP_MEMOIRE » est utilisée par l'Entity manager de notre contrôleur pour lancer des requêtes vers la base de données. La valeur java:/memoireDS est le nom JNDI fixé lors de la configuration de notre data source.

II. Présentation de l'application

Enfin !

Comme dans tout travail, il existe une étape de présentation du résultat obtenu et nous voici à cette étape. Notre présentation se fera par des captures d'écran de quelques pages importantes de notre application. Comme tous les acteurs de notre application ont leur propre interface, nous allons présenter d'abord l'interface des clients (visiteurs) ensuite celle des pharmaciens et des utilisateurs et enfin celle des fournisseurs.

- **Présentation de l'interface des clients**

L'image ci-dessous représente la page d'accueil de notre application. Elle permet aux clients de rechercher des produits dans les différentes pharmacies qui utilisent cette application.

Figure 25 : page d'accueil de l'application.

Comme on peut le voir sur le menu de la **Figure 25**, le client peut :

✓ **Visualiser la liste des pharmacies de garde**

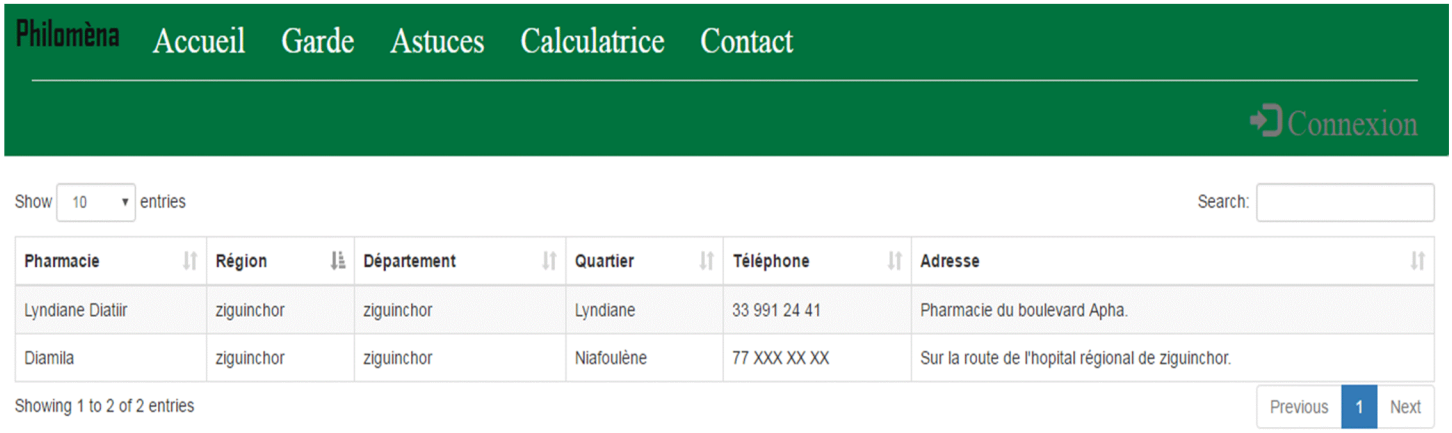


Figure 26 : Liste des pharmacies de garde.

✓ **Calculer le coût d'une ordonnance**

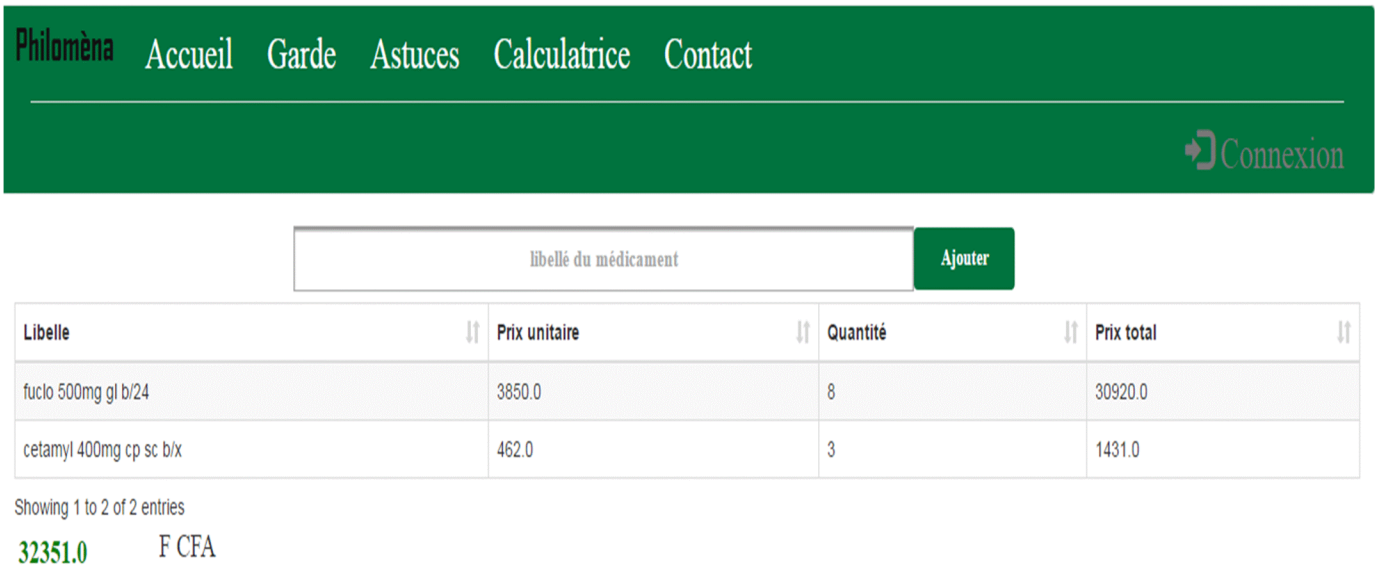


Figure 27 : exemple de calcul.

- **Présentation de l'interface des pharmaciens et des utilisateurs**

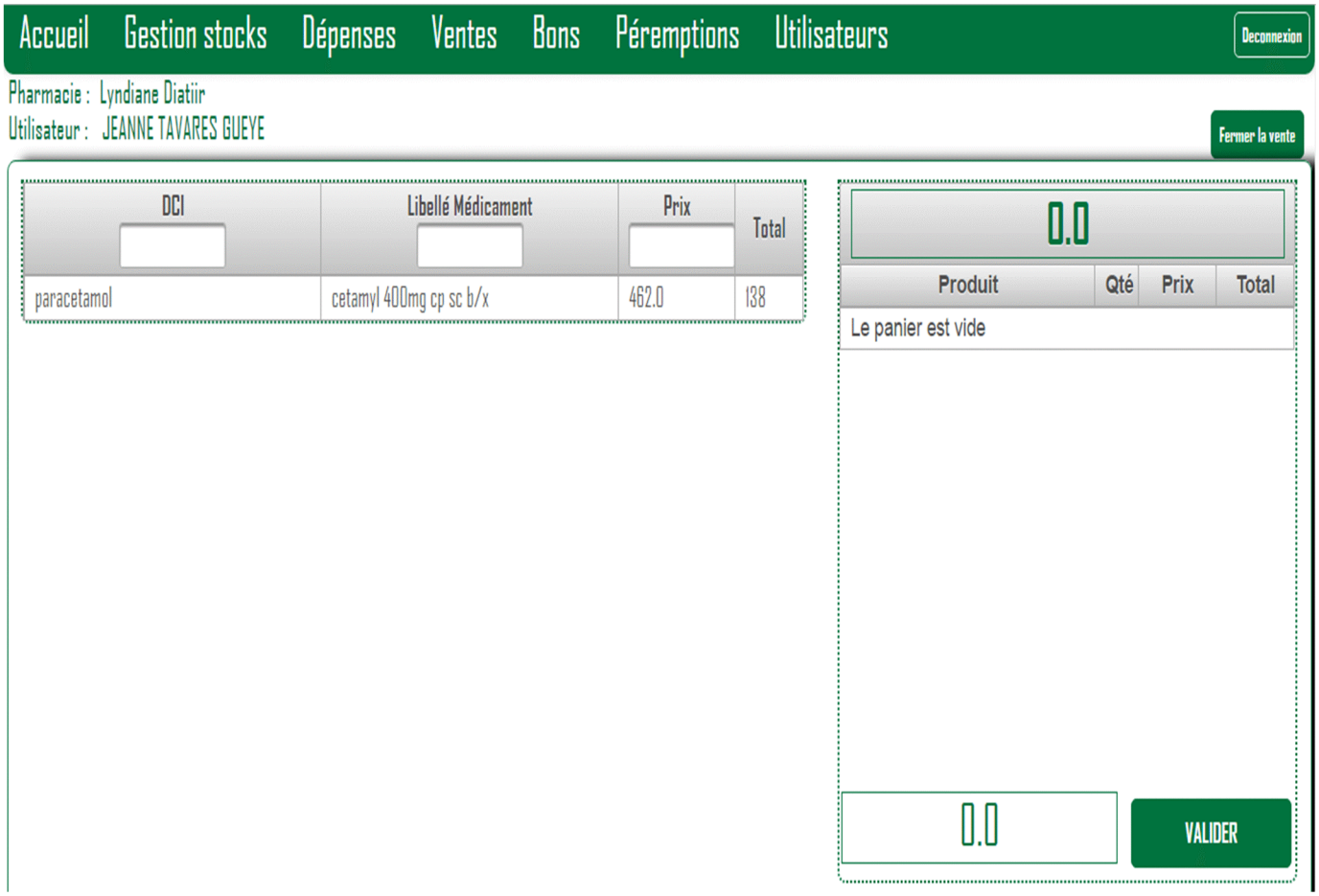


Figure 28 : accueil pharmaciens et utilisateurs.

La **Figure 28** représente la page d'accueil des pharmaciens et des utilisateurs (Vendeurs).

Comme on peut le voir sur le menu, les pharmaciens peuvent gérer leurs stocks, les dépenses, les bons, les ventes, les utilisateurs et contrôler les dates de péremption.

Les utilisateurs (Vendeurs) ont la même interface, la seule différence est que leur interface ne permet pas de gérer les utilisateurs.

- **Présentation de l'interface des fournisseurs**

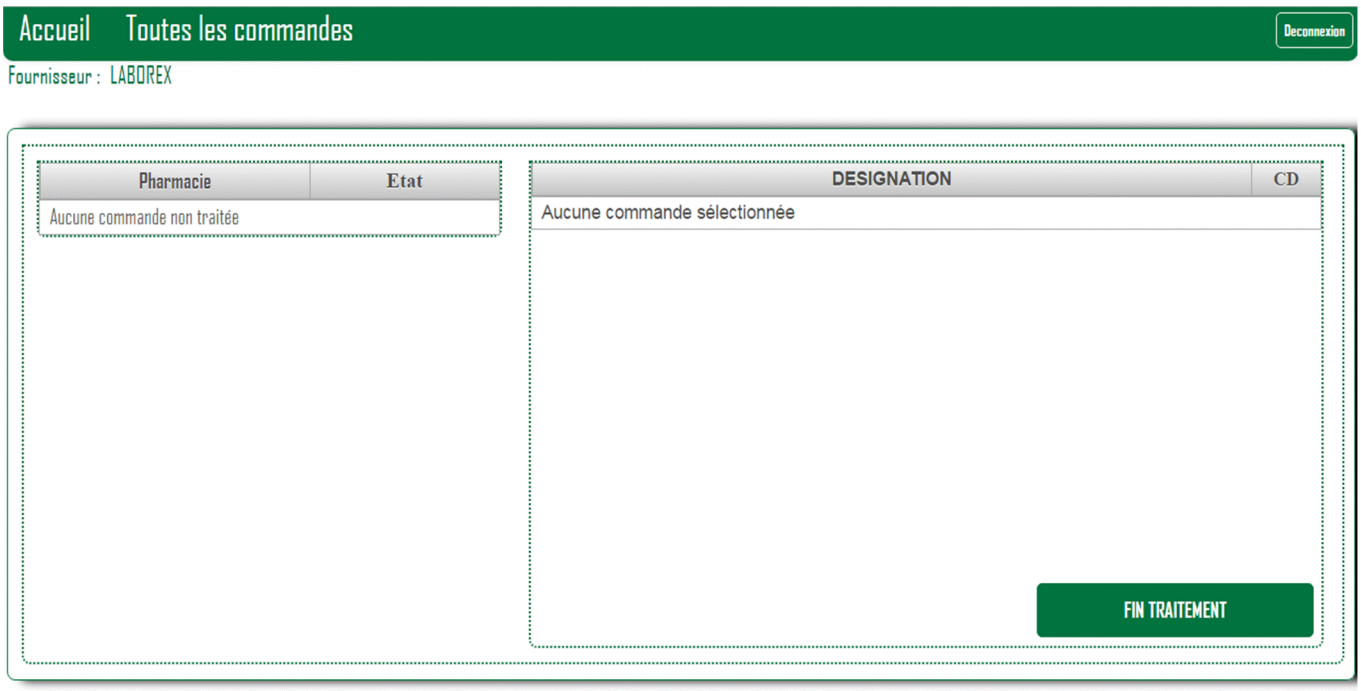


Figure 29 : accueil fournisseurs.

L'interface du fournisseur lui permet de réceptionner les commandes passées par les différentes pharmacies qui utilisent cette application.

A l'exception des clients, tous les autres utilisateurs passent par notre formulaire de connexion ci-dessous.

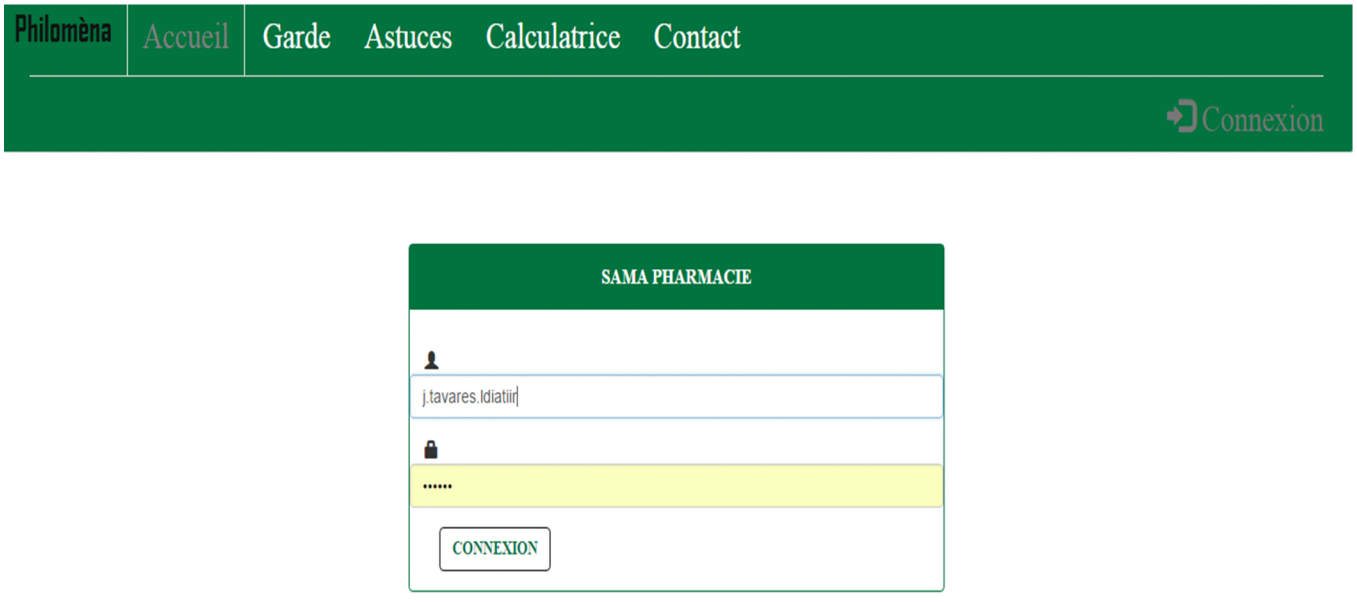


Figure 30 : page de connexion.

Conclusion et Perspectives

Notre stage à la pharmacie Lyndiane Jatiir, pour les besoins de notre mémoire intitulé « Développement d'une application de gestion des pharmacies pour la facilitation de la recherche de médicaments au Sénégal », nous a facilité notre recherche d'informations sur la gestion des pharmacies. Ce stage nous a permis de faire une enquête globale pour bien comprendre leurs fonctionnements. Cette enquête a abouti à la mise en place de notre cahier des charges au Chapitre1. Avec ce cahier des charges, nous avons pu capturer tous les besoins fonctionnels et techniques de notre système au Chapitre2. Cette capture des besoins nous a permis de faire l'analyse et la conception au chapitre3. Après toutes ces études faites dans ces différents chapitres, nous avons commencé notre quatrième et dernier chapitre pour la réalisation (codage) et la présentation de l'application.

Après des mois de stage et de développement, nous décidons enfin de faire une pause afin de mettre notre application à l'appréciation du public avec ses fonctionnalités les plus importantes. Aujourd'hui nous avons une application capable de gérer une pharmacie c'est-à-dire, une application en mesure de ***gérer les ventes***, ***gérer les bons***, ***gérer les dépenses***, ***gérer les utilisateurs***, ***passer des commandes***, ***réceptionner les produits livrés*** et ***prévenir*** le pharmacien ***3 mois avant la péremption*** d'un produit. Ensuite nous avons une application pouvant permettre à la population, de ***rechercher les pharmacies*** qui ont un produit donné, de voir la liste des ***pharmacies de garde***, de ***calculer le coût de leur ordonnance***, de voir les ***astuces sur la santé*** que ce soit avec leurs smart phones, leurs tablettes ou bien leurs ordinateurs personnels car la partie clients est totalement ***responsive***.

Beaucoup de modules restent encore à etre crées car cette application est une petite partie d'un ensemble beaucoup plus vaste. En effet, Sama pharmacie fait partie d'une grande application appelée **Philoména** qui a pour objectif d'informatiser tout le système sanitaire du Sénégal. Cette informatisation se fait par des étapes :

- ✓ Etape des pharmacies
- ✓ Etape des hôpitaux
- ✓ Etape des mutuels de santé.

Bibliographie

[1] Le pharmacien N 084, l'avenir sera informatique ou ne sera pas, 21 Février 2011.

[2] P a s c a l **R o q u e s** • F r a n c k **V a l l é e,** UML 2 en action : De l'analyse des besoins à la conception **4e édition,** EYROLLES.

[3] P a s c a l **R o q u e s** • F r a n c k **V a l l é e,** UML 2 en action : De l'analyse des besoins à la conception **4e édition,** EYROLLES.

[4] P a s c a l **R o q u e s** • F r a n c k **V a l l é e,** UML 2 en action : De l'analyse des besoins à la conception **4e édition,** EYROLLES.

[5] P a s c a l **R o q u e s** • F r a n c k **V a l l é e,** UML 2 en action : De l'analyse des besoins à la conception **4e édition,** EYROLLES.

[6] P a s c a l **R o q u e s** • F r a n c k **V a l l é e,** UML 2 en action : De l'analyse des besoins à la conception **4e édition,** EYROLLES.

[7] P a s c a l **R o q u e s** • F r a n c k **V a l l é e,** UML 2 en action : De l'analyse des besoins à la conception **4e édition,** EYROLLES.

[8] P a s c a l **R o q u e s** • F r a n c k **V a l l é e,** UML 2 en action : De l'analyse des besoins à la conception **4e édition,** EYROLLES.

[9] Wahid Gazzah, Rapport de Projet de Fin d'Etudes : Développement d'un lecteur de code à barre universel pour Android.

Webographie

http://www.d-booker.fr/content/72-postgresql

http://www.jmdoudoux.fr/java/dej/chap-hibernate.htm

Annexes

- **Développement de l'entité Utilisateur (table utilisateur)**

```
17 @Entity
18 public class Utilisateur implements Serializable {
19     private static final long serialVersionUID = 1L;
20     @Id
21     @GeneratedValue(strategy=GenerationType.IDENTITY)
22     private Long id_utilisateur;
23     private String nom_utilisateur;
24     private String prenom_utilisateur;
25     @Column(unique=true, nullable=false)
26     private String login_utilisateur;
27     private String password_utilisateur;
28     private String role_utilisateur;
29     /**
30      * Relation entre l'Utilisateur et la Pharmacie.
31      */
32     @ManyToOne
33     private Pharmacie pharmacie;
```

- **Développement de l'entité Depense (table depense)**

```
11 @Entity
12 public class Depense implements Serializable {
13     private static final long serialVersionUID = 1L;
14     @Id
15     @GeneratedValue(strategy=GenerationType.IDENTITY)
16     private Long id_depense;
17     private String objet_depense;
18     private String date_depense;
19     private double cout_depense;
20     /**
21      * Relation entre Depense et Pharmacie.
22      */
23     @ManyToOne
24     private Pharmacie pharmacie;
```

- **Développement de l'entité Stock (table stock)**

```
21 public class Stock  implements Serializable{
22     private static final long serialVersionUID = 1L;
23     @Id
24     @GeneratedValue(strategy=GenerationType.IDENTITY)
25     private Long id_stock;
26     private int quantite_totale;
27     private double prix_unitaire;
28     private int quantite_minimale;
29     private String dci_medicament;
30     private String libelleMedicament;  //pour faciliter l'affichage des stocks
31     private double prix_cession;
32     private String perisable;
33      // Relation entre le Stock et la Pharmacie.
34     @ManyToOne
35     private Pharmacie pharmacie;
36      //Relation entre Stock et Produit
37     @ManyToOne
38     private Produit produit;
39      //Relation entre Stock et SousStock
40     @OneToMany(mappedBy="stock", cascade=CascadeType.ALL)
41     private List<SousStock> liste_des_sousstocks= new ArrayList<SousStock>();
42      // Relation entre Stock et DetailsVente
43     @OneToMany(mappedBy="stock", cascade=CascadeType.ALL)
44     private List<DetailsVente> listeDetailsVente= new ArrayList<DetailsVente>();
```

- **Développement de l'entité SousStock (table sousstock)**

```
11 @Entity
12 public class SousStock implements Serializable{
13    private static final long serialVersionUID = 1L;
14
15    @Id
16    @GeneratedValue(strategy=GenerationType.IDENTITY)
17    private Long idSousStock;
18    private String datePeremption;
19    private int quantite;
20    private boolean valide;
21    /**
22     * Relation entre SousStock et Stock
23     */
24    @ManyToOne
25    private Stock stock;
```

- **Développement de l'entité Vente (table vente)**

```
16 @Entity
17 public class Vente implements Serializable {
18    private static final long serialVersionUID = 1L;
19    @Id
20    @GeneratedValue(strategy=GenerationType.IDENTITY)
21    private Long id_vente;
22    private String date_vente;
23    private double total_ventes;
24    private String etatVente;
25    private String loginVendeur;
26    /**
27     * Relation entre la Vente et Pharmacie
28     */
29    @ManyToOne
30    private Pharmacie pharmacie;
31    /*
32     * Relation entre Vente et DetailsVente
33     */
34    @OneToMany(mappedBy="vente", cascade=CascadeType.ALL)
35    private List<DetailsVente> listeDesDetails = new ArrayList<DetailsVente>();
```

- **Développement de l'entité DetailsVente (table detailsvente)**

```
11 @Entity
12 public class DetailsVente implements Serializable{
13    private static final long serialVersionUID = 1L;
14    @Id
15    @GeneratedValue(strategy=GenerationType.IDENTITY)
16    private Long id_details_vente;
17    private String produit;
18    private int quantite_vendue;
19    private double prix_unitaire;
20    private double prix_total;
21    /**
22     * Relation entre DetailsVente et Vente
23     */
24    @ManyToOne
25    private Vente vente;
26    /**
27     * Relation entre DetailsVente et Stock
28     */
29    @ManyToOne
30    private Stock stock;
```

- **Développement de l'entité Commande (table commande)**

```
16 @Entity
17 public class Commande implements Serializable {
18    private static final long serialVersionUID = 1L;
19    @Id
20    @GeneratedValue(strategy=GenerationType.IDENTITY)
21    private Long idCommande;
22    private String date;
23    private String etat;
24    /*
25     * Relation entre Commande et Pharmacie
26     */
27    @ManyToOne
28    private Pharmacie pharmacie;
29    /*
30     *  commande et DetailsCommande
31     */
32    @OneToMany(mappedBy="commande",cascade=CascadeType.ALL)
33    private List<DetailsCommande> detailsCommande= new ArrayList<DetailsCommande>();
34    /*
35     * Relation entre Commande et Fournisseur
36     */
37    @ManyToOne
38    private Fournisseur fournisseur;
```

- **Développement de l'entité DetailsCommande (table detailscommande)**

```
11 @Entity
12 public class DetailsCommande implements Serializable {
13    private static final long serialVersionUID = 1L;
14    @Id
15    @GeneratedValue(strategy=GenerationType.IDENTITY)
16    private Long idDetailsCommande;
17    private int quantiteCommandee;
18    private String libelleProduit;
19    /**
20     * Relation entre DetailsCommande et Commande
21     */
22    @ManyToOne
23    private Commande commande;
24    /**
25     * Relation entre  DetailsCommande et Stock
26     */
27    @ManyToOne
28    private Stock stock;
```

- **Développement de l'entité Fournisseur (table fournisseur)**

```java
@Entity
public class Fournisseur implements Serializable{
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long idFournisseur;
    @Column(unique=true, nullable=false)
    private String nomFournisseur;
    private String adresse;
    private String email;
    private String telephone;
    @Column(unique=true, nullable=false)
    private String login;
    private String password;
    /**
     * Relation entre Fournisseur et Commande
     */
    @OneToMany(mappedBy="fournisseur",cascade=CascadeType.ALL)
    private List<Commande>liste_des_commandes = new ArrayList<Commande>();
```

Développement de l'entité Bon (table bon)

```java
@Entity
public class Bon implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id_bon;
    private double montant_actuel_bon;
    private String etat_bon="NON PAYE";
    /**
     * Relation entre Bon et DetailsBon.
     */
    @OneToMany(mappedBy="bon", cascade=CascadeType.ALL)
    private List<DetailsBon>liste_details_bon = new ArrayList<DetailsBon>();
    /**
     * Relation entre Bon et Client.
     */
    @ManyToOne
    private Client client;
    /**
     * Relation entre Bon et Pharmacie.
     */
    @ManyToOne
    private Pharmacie pharmacie;
```

- **Développement de l'entité Client (table client)**

```java
@Entity
public class Client implements Serializable{
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id_client;
    private String nom_client;
    private String prenom_client;
    private String email_client;
    private String adresse_client;
    private String numero_client;
    private String numero_carte_id;
    private String login;
    private String password;
    /**
     * Relation entre Client et Bon
     */
    @OneToMany(mappedBy="client", cascade=CascadeType.ALL)
    private List<Bon>liste_des_bons_client = new ArrayList<Bon>();
```

- **Développement de l'entité Medicament (Produit) (table medicament)**

```
17 @Entity
18 public class Produit  implements Serializable{
19     private static final long serialVersionUID = 1L;
20     @Id
21     @GeneratedValue(strategy=GenerationType.IDENTITY)
22     private Long id_medicament;
23     private String dci_medicament;
24     @Column(nullable=false)
25     private String libelle_medicament;
26     private String tableau_medicament;
27     private double prix_calculatrice;
28     private String perisable;
29     /*
30      * Relation entre Médicament et Stock
31      */
32        @OneToMany(mappedBy="produit", cascade=CascadeType.ALL)
33        private List<Stock> listeDesStocks = new ArrayList<Stock>();
34     /*
35     * Relation entre Produit et DetailsBon
36     */
37     @OneToMany(mappedBy="produit", cascade=CascadeType.ALL)
38     private List<DetailsBon> listeDesDetailsBon = new ArrayList<DetailsBon>();
```

- **Développement de l'entité DetailsBon (table detailsbon)**

```
13 public class DetailsBon implements Serializable {
14     private static final long serialVersionUID = 1L;
15     @Id
16     @GeneratedValue(strategy=GenerationType.IDENTITY)
17     private Long id_detail_bon;
18     private int quantite;
19     private double prix_unitaire;
20     private double prix_total;
21     private String date;
22     private String utilisateur;
23     private String nom_produit;
24     /**
25      * Relation entre DetailsBon et Stock .
26      */
27     @ManyToOne
28     private Stock produit;
29     /**
30      * Relation entre DetailsBon et Bon
31      */
32     @ManyToOne
33     private Bon bon;
```