SAE JAVA - POO “LIVRE EXPRESS”

Robin FAUCHEUX - Joris VACHEY - Raphaël BROUSSEAU (chef de projet) - Corentin LACOUME

2024-2025

SOMMAIRE

I. Introduction

Au cours de cette SAE, nous devions réaliser une application afin d'assurer la gestion informatique d'une chaîne de librairies permettant aux différents types d'utilisateurs de réaliser des tâches en fonction de leur rôle. Les fonctionnalités de cette application devraient inclure pour un utilisateur de type client : le choix de passer une commande, qu'elle se fasse en ligne ou en magasin, ainsi que la possibilité de consulter le catalogue de livres de l'application. Pour les utilisateurs vendeurs, ils devaient avoir la possibilité de consulter, ainsi que de modifier le stock. De plus, ils doivent pouvoir passer une commande

à la place d'un client. De même que de demander le transfert d'un livre depuis un magasin vers un autre. Dernièrement, les administrateurs doivent avoir la possibilité de créer des comptes vendeurs, ainsi que de pouvoir ajouter de nouvelles librairies au réseau, mais aussi de gérer les stocks communs à toutes les librairies et enfin de consulter les statistiques de ventes des différentes librairies. De plus, l'application doit permettre d'accéder à des parties que nous avions réalisé lors de la SAE Exploitation d'une base de données. Ces parties incluent : L'accès aux informations du tableau de bord, l'accès à la réalisation de factures, même si celles ci doivent se présenter au format PDF, ce qui n'était pas le cas lors de la SAE précédente, ainsi que la capacité à voir les palmarès des meilleurs livres, auteurs, vendeurs et boutiques. Nous avions aussi des contraintes à réaliser telles que la proposition de livres recommandés à l'utilisateur. Enfin, cette application devrait permettre d'accéder à toutes ces fonctionnalités sous forme d'une application en ligne de commande, organisée en sous-menus cohérents.

Dans ce rapport, nous allons premièrement faire une analyse de la base de données, puis dans une seconde partie nous allons analyser notre schéma UML afin d'expliquer le fonctionnement de notre application. Dans une troisième partie, nous allons expliquer l'implémentation de nos schémas UML dans l'application. Ensuite, nous discuterons de l'élaboration de l'IHM (Interface Humain-Machine), avant de faire un bilan de l'organisation de notre travail en équipe. Enfin, nous effectuerons un bilan global de notre groupe.

II. Analyse de la base de donnée

Dans ce chapitre nous allons expliquer et analyser la base de données fournie avec la SAE. Nous allons commencer par expliquer le schéma MCD et son fonctionnement, avant de nous tourner vers une explication des principales requêtes permettant d'obtenir des informations auprès de la base de données. Finalement, nous analyserons le choix des insertions de la base de données.

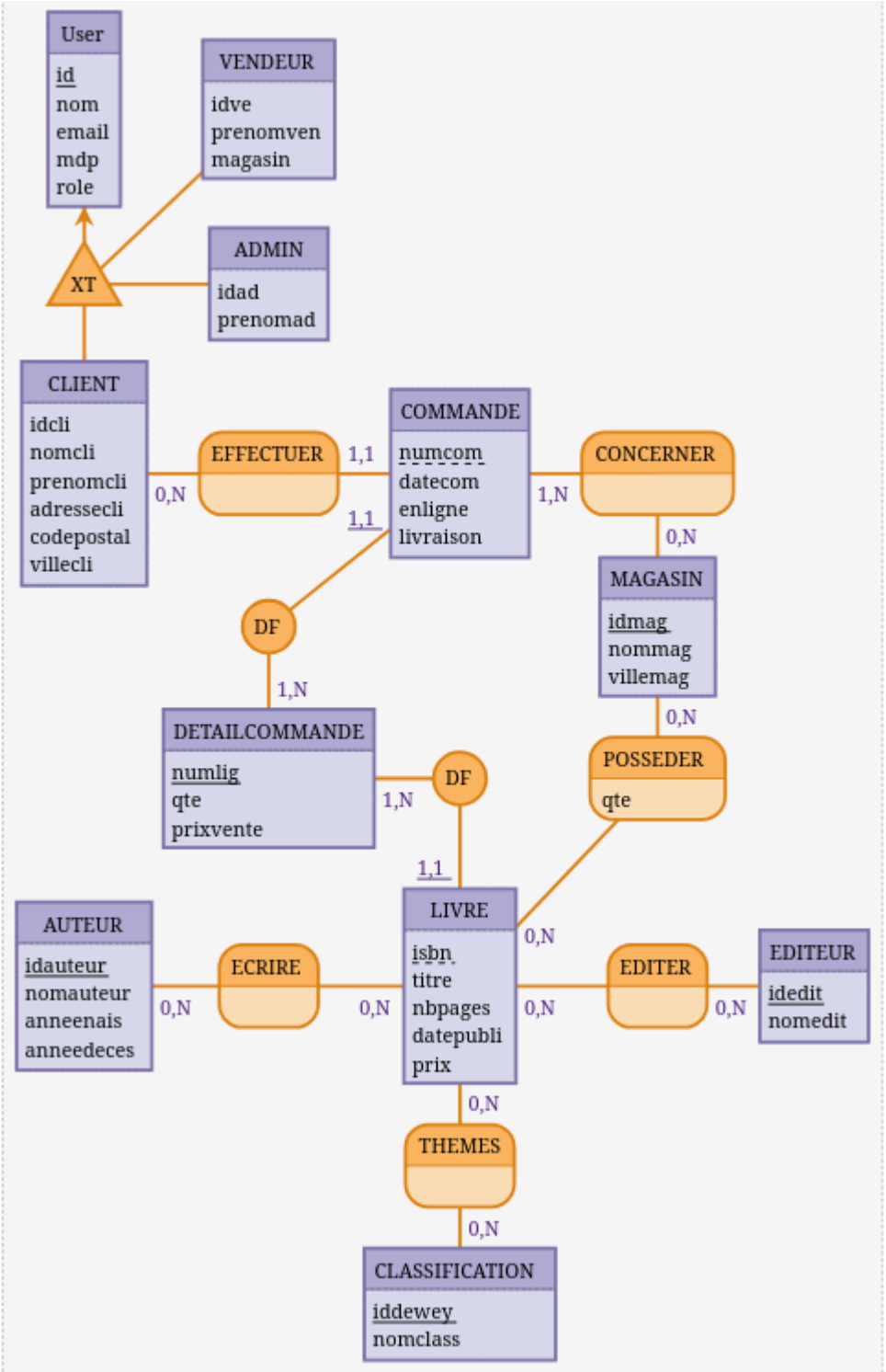

Figure 1, MCD de la base Livre Express.

A. Explication du MCD / MLD

Un MCD (Modèle Conceptuel de Données) est une représentation graphique et logique d'un système de données, qui permet de modéliser les entités (futures tables), leurs attributs et les relations entre elles. Il aide à comprendre la structure de la base de données et à identifier les liens entre les données pour faciliter la navigation et les requêtes.

User : Cette table contient toutes les informations nécessaires à l'identification des utilisateurs.

- Clé primaire : idu -> un numéro qui permet l'identification d'un utilisateur de manière unique.

Client : Cette table contient toutes les informations nécessaires à l'identification des clients. C'est une spécialisation de user.

- Clé primaire : idcli -> un numéro qui permet l'identification d'un client de manière unique.

Vendeur : Cette table contient toutes les informations nécessaires à l'identification des clients. C'est une spécialisation de user.

- Clé primaire : idcli -> un numéro qui permet l'identification d'un client de manière unique.

Admin: Cette table contient toutes les informations nécessaires à l'identification des clients. C'est une spécialisation de user.

- Clé primaire : idcli -> un numéro qui permet l'identification d'un client de manière unique. C'est une spécialisation de user.

Commande : Cette table répertorie toutes les commandes effectuées par les clients. Elle contient des informations comme la date où a été faite la commande, si elle a été faite en magasin ou en ligne et ou elle doit être livrée.

- Clé primaire : numcom -> numéro qui permet d'identifier un client de manière unique.
- Clé étrangère : idclient -> permet d'associer un client a une commande.
  idmag -> permet d'associer un magasin a une commande.

Auteur : Cette table contient les informations utiles à une librairie, elle n'inclut donc que son nom ainsi que ses dates de naissance et de décès, ainsi qu'un identifiant unique. Chaque auteur est relié aux livres qu'il a rédigés par une table Écrire.

- Clé primaire : idauteur -> permet d'identifier chaque auteur de manière unique.

Livre : Cette table inclut les informations utiles à l'achat d'un livre par un client. Elle contient l'isbn d'un livre, mais aussi son titre, son nombre de pages ainsi que sa date de publication et son prix.

- Clé primaire : isbn -> permet d'identifier chaque livre de façons uniques

DétailCommande : Cette table est une entité faible dépendant de commande et permet de détailler des informations pour chaque livre de chaque commande. Elle est donc identifiée par l'identifiant de la commande ainsi que le numéro de la ligne parmi la commande. Elle contient des informations incluant le livre commandé sur cette ligne, ainsi que la quantité de ce même livre acheté dans la commande. Enfin, elle contient le prix de vente résultant du prix du livre et de la quantité du livre acheté.

- Clé primaire : numcom,numlig -> permet d'identifier chaque commande de manière unique
- Clé étrangère : isbn -> permet d'associer un livre dans le détail de commande
  numcom -> permet d'associer une commande dans le détail de commande

Posséder : Cette table est une relation entre un livre et un magasin permettant d'obtenir la quantité en stock de ce même livre dans le magasin. Cette quantité est identifiée en utilisant l'identifiant du livre, ainsi que l'identifiant du magasin,

- Clé primaire : idmag, isbn -> permet d'identifier chaque quantité de manière unique
- Clé étrangère : isbn -> permet d'associer un livre dans un magasin
  idmag -> permet d'associer un magasin ou l'on vérifiera si un livre y est présent

Éditeur: Cette table contient les informations sur les différents auteurs, qu'ils aient écrit des livres en stock dans la bibliothèque ou non. Ils sont identifié avec un id Dewey (système de classification bibliothécaire basé sur une structure décimale) qui est la clé primaire et son relié à des livré par la table "Editer"

- Clé primaire : idedit -> permet d'identifier de façon unique les Éditeur

classification: Cette table contient les informations sur les différents sujets et thèmes abordés par les livres, qu'ils y en aient en stock dans la bibliothèque ou non. Ils sont identifié avec un identifiant (clé primaire) et son relié à des livré par la table "Themes"

- Clé primaire : iddewey -> ce qui permet encore d'identifier une classification de façon unique

Magasin : Cette table répertorie tous les magasins appartenant à *Valle Lire.* Elle contient son nom, ainsi que sa ville d'activité.

- clé primaire : nummag -> Un numéro qui permet l'identification unique de chaque magasin.

Écrire : Cette table permet de savoir qui a écrit un livre. Elle contient l'isbn d'un livre ainsi que idauteur qui est l'id de l'auteur qui a écrit ce livre

- Clé primaire : isbn et idauteur
- Clé étrangère : isbn et idauteur

Éditer: Cette table permet de savoir qui a édité un livre précis. Les attributs sont isbn et nomedit.

- Clé primaire : isbn et nomedit
- Clé étrangère : isbn et nomedit

Thème: Cette table permet d'identifier un thème précis pour chaque livre. Cette table contient deux attributs qui sont iddewey et isbn.

- Clé primaire : isbn et iddewey
- Clé étrangère isbn et iddewey

B. Explication des principales requêtes

Les requêtes sont des injonctions que l'on effectue auprès d'une base de données afin d'en obtenir des informations. Ces requêtes peuvent être plus ou moins complexes en fonction des informations que nous désirons obtenir et de leur répartition sur les différentes tables de la base de données Les requêtes de cette SAE permettent d'extraire différents types d'informations à partir de la base de données de la librairie. Certaines identifient des livres commandés à une date précise ou par des clients spécifiques , comme ceux commandés le 1er décembre 2024 ou par des lecteurs de René Goscinny en 2021

```
--requête livres commandés 1er décembre 2024
select isbn, titre, nbpages, datepubli, prix
from LIVRE natural join DETAILCOMMANDE NATURAL JOIN COMMANDE
WHERE datecom=str_to_date('01/12/2024','%d/%m/%Y');

--requête client ayant commandé des livres de René Goscinny en 2021
select distinct idcli, nomcli, prenomcli, adressecli, codepostal,
villecli
from CLIENT
natural join COMMANDE natural join DETAILCOMMANDE natural join LIVRE
natural join ECRIRE natural join AUTEUR
where year(datecom) = 2021 and nomauteur = "René Goscinny" ;
```

La première requête ci-dessus (celle concernant les livres commandés le 1/12/25) est découpée en 3, la première partie avec le select permet de sélectionner les informations que l'on veut afficher (ici ce sera l'identifiant unique du livre, son titre et ces autres caractéristiques). La deuxième partie concerne les tables dans lesquelles nous allons chercher l'information dans les tables de la base de données qui contiennent les informations du select. Cette action(celle d'aller chercher l'information) est réalisée avec le mot clé FROM. Dans cette seconde partie nous faisons aussi ce que appel des jointure entre les tables avec le mot clé JOIN qui possède plusieur préfixe comme NATURAL (qui permet de joindre une table à une autre sans préciser les attributs en communs). Ces jointure respectent la structure de la base données et elles sont le moyen de se déplacer dans la base. En clair, joindre les tables permet de fusionner les informations. Enfin notre requête est composée d'une troisième partie qui va concerner les conditions que doivent respectées les informations trouvées par les deux précédentes parties pour être affichées. Le mot clé WHERE permet

dans cette exemple de sélectionner les informations dont l'année de la date de la commande datecom est égal à 2024.

D'autres requêtes recherchent les livres sans auteur ou encore des livres disponibles en quantité suffisante

```
select isbn, titre, nommag, qte
from LIVRE
natural left join ECRIRE
natural join POSSEDER
natural join MAGASIN
where idauteur is null and isbn is not null and qte > 8;
```

Cette requête effectue une jointure naturelle avec un `LEFT JOIN` entre deux tables. Le mot-clé `LEFT` signifie que toutes les lignes de la table de gauche seront conservées dans le résultat, même si aucune correspondance n'est trouvée dans la table de droite.Sans le `LEFT`, c'est-à-dire avec une jointure naturelle simple, seules les lignes ayant des correspondances dans les deux tables sont affichées.Avec un `LEFT JOIN`, si une ligne de la table de gauche ne correspond à aucune ligne dans la table de droite, elle apparaîtra quand même dans le résultat, mais les colonnes provenant de la table de droite contiendront la valeur `NULL`.

Des requêtes permettent aussi de sélectionner toutes les informations utiles à un livre comme son isbn , titre, prix, sa disponibilité dans un magasin donné. Il y a aussi une partie des requêtes visant à fournir des statistiques, par exemple le nombre de clients vivant dans la même ville que chaque magasin

```
select idmag, nommag, ifnull(count(idcli), 0) as nbcli

from CLIENT
natural join COMMANDE
natural join MAGASIN
where villemag = villecli
group by idmag;
```

Cette requête montre l'utilisation d'un group by qui permet de trier le résultat en fonction d'un attribut. De plus, il y a l'utilisation du `ifnull` ainsi que du `count`. Le `count` permet dans ce cas de compter le nombre de clients ayant commandé dans chaque magasin la colonne `idcli`, et si ce résultat est `NULL` donc qu'aucun client n'a commandé dans ce magasin le `ifnull` permet de remplacé le `NULL` par un 0. Il y aussi la quantité de livres achetés à une date donnée par magasin.

```
select m.nommag, ifnull(sum(qte),0) as nbex
```

```
from COMMANDE co
natural join DETAILCOMMANDE
right join MAGASIN m on m.idmag = co.idmag
and co.datecom = str_to_date("15/09/2022", '%d/%m/%Y')
group by m.nommag;
```

On voit sur cette requête l'utilisation d'un right join qui permet de combiner deux tables en sélectionnant les lignes de la table de droite. Il y a aussi l'utilisation du str_do_date qui une fonction SQL utilisée pour convertir une chaîne de caractères (string) en date Plusieurs requêtes servent à produire des données pour des graphiques : chiffre d'affaires par thème, par mois.

```
with CA as (
select sum(prixvente*qte) CAT from COMMANDE natural join DETAILCOMMANDE
   where year(datecom) = 2024),
   CAParTheme as (
       select iddewey, nomclass, sum(prixvente*qte) as Montant
       from COMMANDE
       natural join DETAILCOMMANDE
       natural join LIVRE
       natural join THEMES
       natural join CLASSIFICATION
       where year(datecom) = 2024
       group by floor(iddewey/100))
select iddewey, nomclass, Montant, (Montant / CAT) * 100 as pourcentage
from CAParTheme
cross join CA
group by iddewey;
```

Dans cette requête, l'utilisation d'une vue temporaire est nécessaire, c'est pour cela qu'on utilise le mot clé "with". Ou par client, évolution des ventes selon le type (en ligne ou en magasin), quantité de livres vendus par auteur ou selon la ville des clients.

```
select villecli as ville, count(idcli) as qte
from CLIENT
natural join COMMANDE
natural join DETAILCOMMANDE
natural join LIVRE
natural join ECRIRE
natural join AUTEUR
where nomauteur = "René Goscinny"
group by villecli;
```

Une autre calcule la valeur du stock de livres par magasin

```
select nommag as magasin, sum(qte*prix) as valeurStock
from MAGASIN
natural join POSSEDER
natural join LIVRE
```

```
group by nommag;
, et une requête génère des instructions d'insertion dans la base DELETE
FROM POSSEDER WHERE isbn = "9782844273765";
```

Enfin, certaines requêtes comparent les performances des auteurs d'une année sur l'autre

```
SELECT nommag, numcom, datecom, nomcli, prenomcli, adressecli, codepostal,
villecli, numcom ,isbn, titre, qte, prixvente, (prixvente*qte) as total
FROM COMMANDE
NATURAL JOIN DETAILCOMMANDE
NATURAL JOIN CLIENT
NATURAL JOIN LIVRE
NATURAL JOIN MAGASIN
WHERE year(datecom) = 2020 and month(datecom)= 02
ORDER BY nommag, numcom, isbn;
```

, ou listent les commandes effectuées sur un mois précis, comme février 2020. Ces requêtes utilisent des jointures, des filtres temporels, des regroupements et des fonctions d'agrégation.

C. Le choix des insertions

Une insertion est l'opération qui consiste à ajouter de nouvelles données dans une base de données.

Par exemple dans cette insertion:

```
insert into CLIENT (idcli, nomcli, prenomcli, adressecli, codepostal,
villecli) values (1, 'Rodriguez', 'Fatima', '188 chemin de la Forêt',
'45000', 'Orléans')
```

Nous ajoutons un client a la base de donnée, permettant donc à ce client d'exister et donc de commander etc en ajoutant son id, son nom, prénom et toutes les informations nécessaire à la base

Dans cette commande si:

```
insert into LIVRE(isbn, titre,nbpages,datepubli,prix) values
    ('9782205054750', 'Les cavernes', 48, 2003, 8.81),
```

Nous ajoutons le livre a la base afin que celles ci puissent être disponible dans les stocks et ainsi être commandé par un client. Cela permet donc de gérer la quantité en stock, changer les maisons d'éditions etc de manière rapide

Cette commande :

```
insert into COMMANDE(numcom, datecom, enligne, livraison, idcli, idmag) values
    (1,str_to_date('1/1/2020','%d/%m/%Y'),'N','M',356,5),
```

Permet quant à elle d' ajouter une commande, et donc de relier un livre à une personne, et de créer un ticket.

Les commandes sont similaires pour toutes les insertions, nous changeons juste le nom de la table et les attributs que nous ajoutons.

Cette commande :

```
insert into ECRIRE(isbn,idauteur) values ('9782205054750', 'OL7572575A')
```

permet de lier l'auteur d'identifiant <OL7572575A> au livre <'9782205054750'> permettant de savoir que l'auteur a écrit ou coécrit le livre <'9782205054750'>.

La commande suivante :

```
insert into CLASSIFICATION(iddewey, nomclass) values (000,
'Informatique, généralités')
```

permet d'ajouter la classification <Informatique généraliste> d'identifiant 000.

La commande :

```
insert into EDITER(isbn,idedit) values ('9782011801333', 43)
```

permet d'indiquer que le livre <9782011801333> a été écrit ou coécrit par l'éditeur d'identifiant 43.

Dans cette commande :

```
insert into THEMES(isbn,iddewey) values ('9782070633708', 840)
```

Nous lions le livre d'identifiant <9782070633708> au thème d'identifiant 840, représentant ainsi que ce livre fait partie de la classification 840.

III. Analyse UML de l'application

Un diagramme UML (Unified Modeling Language) est un diagramme permettant la représentation d'un programme et de son fonctionnement sous la forme d'un diagramme, facilitant sa compréhension. Il peut se présenter sous différentes formes : diagramme de classe permettant de voir les principales composantes d'une application, les attributs et méthodes la composant, ainsi que les principales relations entre elles (héritage,

implémentation, attributs). Il peut aussi se présenter sous la forme d'un diagramme de séquence détaillant de fonctionnement d'une fonctionnalité en montrant les différents appels de fonctions, et leur retour en utilisant au fur et à mesure de l'exécution de l'application et de la fonctionnalité. De plus, il existe les diagrammes de cas d'utilisation, permettant de visualiser les différents acteurs participant lors de l'utilisation de la fonctionnalité.

Pour cette SAE, nous avons réalisé les diagrammes UML de la figure **2** :

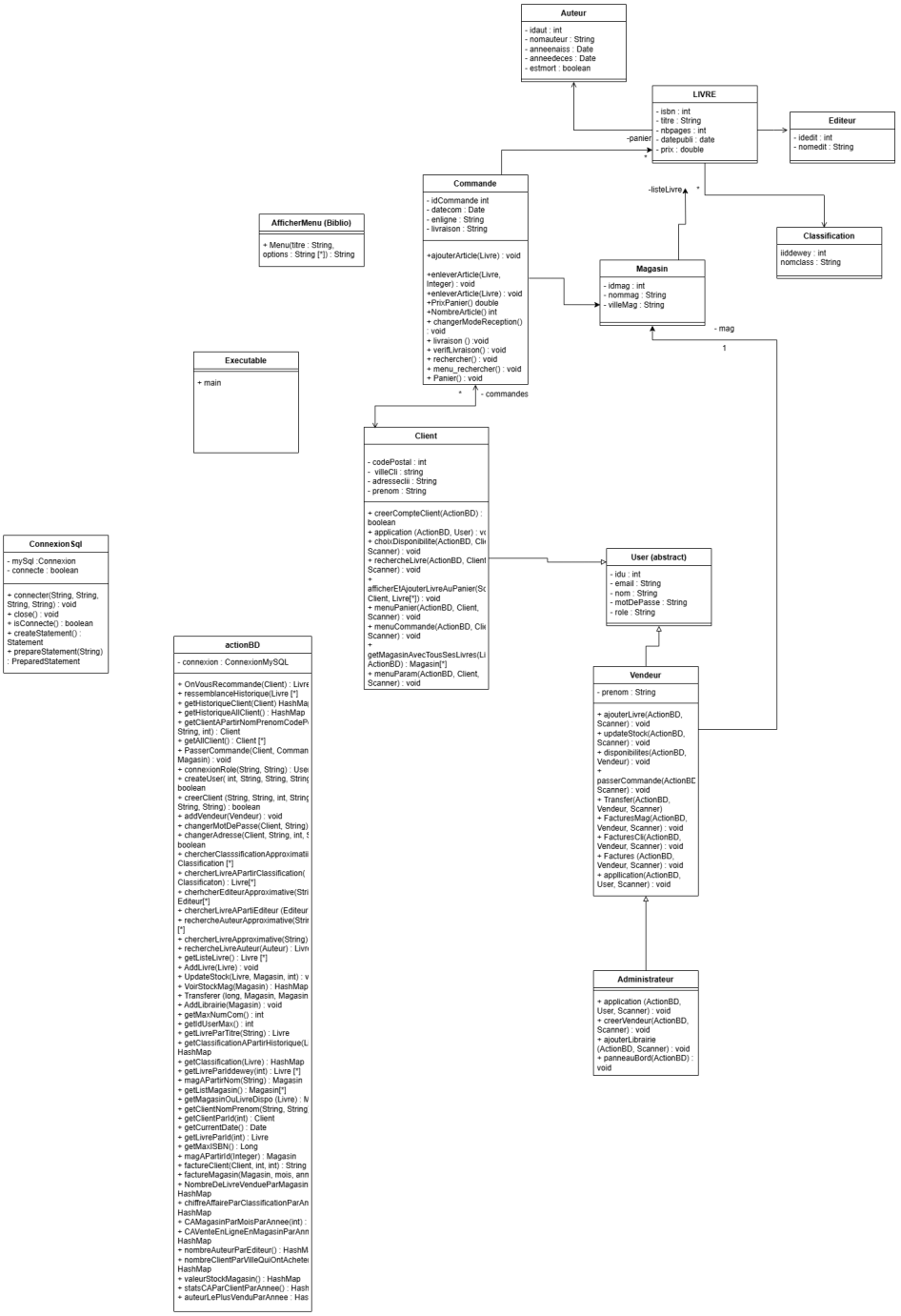

Figure 2 - Diagramme de Classe de l'application

A. Diagramme de classe

Le diagramme de classe de la figure 2 montre les différentes classes composant notre application.

Premièrement, nous avons une classe User permettant de poser les bases pour tous les types d'utilisateurs avec des informations importantes telles que son identifiant unique, son prénom, son nom ainsi que le magasin auquel il est actuellement rattaché, afin de pouvoir facilement commander dans son magasin préféré pour un utilisateur client, mais aussi pour indiquer le magasin de référence d'un vendeur, ainsi que le magasin actuellement en administration par les utilisateurs administrateurs. De plus, nous stockons son rôle afin de pouvoir créer la classe correspondante. Plusieurs classes héritent de cette classe User : la classe Client ainsi que la classe Vendeur, mais aussi indirectement, la classe Administrateur.

De son côté, la classe client ajoute des attributs par rapport à la classe User. Premièrement, nous stockons son code postal, sa ville de résidence ainsi que son adresse afin de faciliter la recherche du magasin le plus proche, mais aussi d'offrir un choix rapide pour l'adresse de livraison dans la cas où il passerait une commande en souhaitant se faire livrer. Nous stockons aussi son historique de commandes afin de pouvoir utiliser ces informations pour la fonction onVousRecommande(), mais aussi à des fins de consultations par l'utilisateur.

Le client a aussi accès à des méthodes. Toutes ses méthodes sont statiques. La première méthode permet de créer un nouveau compte client. Ensuite, nous avons une méthode “Application” faisant office de menu pour le client. De ce menu, le client peut rechercher des livres et les commander, mais aussi changer ses paramètres et obtenir ses recommandations. Toutes ces fonctionnalités découlent des fonctions présentes dans application, et des menus proposés dans celle-ci.

Pour le vendeur, l'application lui permet d'avoir accès à certaines fonctionnalités. Premièrement le vendeur a l'option d'ajouter un livre au catalogue de livres global, de plus il peut librement changer le stock d'un livre parmi le catalogue dans son magasin. Il peut aussi consulter le catalogue des livres disponibles dans son magasin, ainsi que leur quantité présente dans les stocks. Mais le vendeur peut aussi passer une commande à la place d'un client, ainsi que demander le transfert d'un livre depuis un autre magasin vers le sien, afin de pouvoir le vendre à un client qui aurait émis la requête de le commander. Enfin, le vendeur a l'option d'éditer les factures de son magasin, ou bien d'un client. Le vendeur n'ajoute pas d'attributs par rapport à la classe User. Il fonctionne néanmoins de la même manière que Client, en utilisant uniquement des méthodes statiques, ainsi qu'une fonction application permettant d'obtenir un menu avec toutes les options qui s'offrent à lui.

Notre UML possède une classe Administrateur qui hérite directement de la classe Vendeur. En effet, nous avons émis l'idée qu'un administrateur a la possibilité d'effectuer les mêmes actions qu'un vendeur, mais aussi la possibilité d'administrer les différents magasins de la chaîne de librairies. Cet administrateur ne possède pas d'attributs supplémentaires à la classe Vendeur, et la classe User.

Cependant, l'administrateur a accès à des outils d'administration. Premièrement, il peut créer des utilisateurs vendeurs et leur assigner un magasin de référence. De plus, il a la possibilité d'ajouter des librairies au réseau. Il possède aussi la capacité d'accéder au tableau de bord de la chaîne de magasin composée des graphiques réalisés au cours de la SAE Exploitation d'une base de données, ainsi que de l'accès au palmarès des auteurs.

Notre application possède une classe commande. Celle-ci permet de gérer une commande individuelle. Elle a pour attribut l'identifiant de la commande, le mode de réception choisi ainsi qu'un panier composé de plusieurs Livres (instances de la classe Livre) La commande est associée à un Magasin de référence dans le cas où l'utilisateur souhaite retirer la commande en magasin.

Cette classe permet de simplifier la création d'une commande dans la base de données.

L'application est composée d'une classe magasin. Un magasin possède un identifiant de magasin unique, ainsi qu'un nom et une ville d'implantation. Les magasins sont des objets sans méthode.

Une composante importante de notre application est la classe Livre. Celle-ci permet de créer un objet à l'image de la table Livre de notre base de données. Ainsi, elle a pour attributs l'isbn (identifiant) du livre, le titre, le nombre de pages, la date de publication et le prix du livre. Une méthode de livre permet d'obtenir le prix associé à ce livre, particulièrement utile pour connaître le prix total du panier, ainsi que le prix individuel du livre à titre consultatif.

Afin de se rapprocher du MCD de la base de données, nous avons opté pour une architecture similaire pour la classe Livre. En effet, elle est raccordée aux classes Auteur, Éditeur, et classification pour stocker les entrées de la table sous formes d'objets java. Ceux-ci simplifient les communications avec la base de données, ainsi que son exploitation dans les différentes fonctionnalités (par exemple, les consultations du catalogue affichent aussi son/ses auteurs). Ces classes "annexes" sont des attributs de la classe livre.

Nous avons réalisé une classe AfficherMenu qui se présente sous la forme d'une bibliothèque, n'ayant qu'une méthode statique permettant d'afficher des menus dans un terminal de manière modulable (le titre du menu est passé en paramètre, ainsi que les options disponibles dans celui-ci). Cette classe n'est qu'une classe d'affichage, elle ne gère en aucun point la logique associée aux options. En effet, les entrées de l'utilisateur sont traitées par d'autres fonctions, selon les besoins spécifiques de ces fonctions.

La classe la plus importante de notre application est la classe Action BD. Cette classe agit comme intermédiaire entre le reste de l'application et la base de données. Elle nous permet d'y faire des insertions, modifications, mais aussi de la consulter. Cette classe possède pour unique attribut une connexion à la base de données afin de réaliser les opérations.

Cette classe possède une multitude de fonctionnalités. Ces fonctionnalités incluent mais ne se limitent pas à : passer une commande, recuperer la liste de livres, ajouter un livre, mettre à jour le stock, récupérer le stock d'un unique magasin, transférer un livre, ajouter des utilisateurs (clients, vendeurs), mais aussi des librairies, obtenir les informations du tableau de bord et charger un utilisateur depuis la base de données.

Enfin, la classe connexionSql permet d'initialiser la connexion avec le serveur de la base de données, ainsi que de créer des requêtes à envoyer au serveur. Elle a pour attribut la connexion au serveur ainsi qu'un attribut est connecte permettant de savoir si le serveur de base de données est connecté.

Cette classe dispose de méthodes permettant d'ouvrir la connexion, ainsi qu'une autre permettant de se déconnecter du serveur. La classe possède aussi une méthode permettant de savoir si le programme est connecté au serveur de base de données. Enfin, les deux dernières méthodes permettent de créer des requêtes à transmettre au serveur.

B. Diagrammes de séquence

Diagramme de séquence de la méthode PasserCommande():

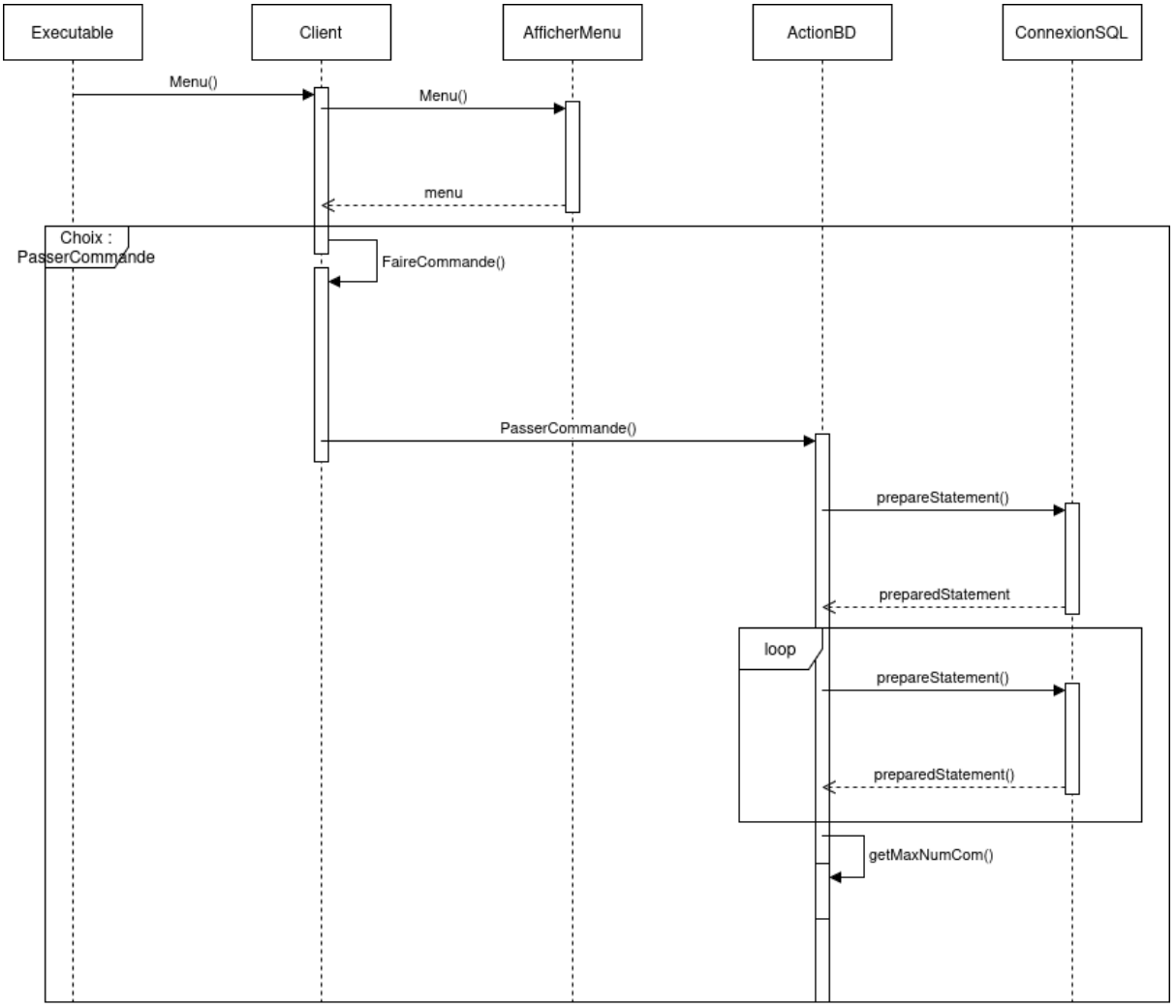

Figure 3 - Diagramme de séquence PasserCommande()

La figure 3 représente le diagramme de séquence lorsqu'un utilisateur (Client ou vendeur commandant pour un client) passe une commande. L'utilisateur commence par exécuter la fonction menu, lui permettant d'effectuer un choix parmi les actions disponibles. Si cet utilisateur choisit de passer une commande, la classe client va appeler sa propre fonction FaireCommande(). Celle-ci affiche un menu permettant à l'utilisateur de renseigner les informations nécessaires et utiles pour la base de données, permettant à celle-ci d'enregistrer la commande. Les informations sont passées en paramètre de la fonction sous la forme d'un objet Livre, ainsi qu'un objet Client et un objet Magasin. A partir de ces objets, la fonction PasserCommande de la classe ActionBD va extraire les informations nécessaires à la création d'une commande. Elle va alors préparer une requête afin d'insérer la commande dans la table COMMANDE de la base de données. Nous récupérons alors depuis celle-ci le plus grand numéro de commande afin d'attribuer un numéro unique à cette dernière commande. Ensuite, pour chaque livre commandé, on crée des requêtes

permettant son insertion dans la table DETAILCOMMANDE. Ainsi, nous avons ajouté la commande a la base de données à partir des choix de l'utilisateur.

Diagramme de séquence de la méthode EditerFacture():

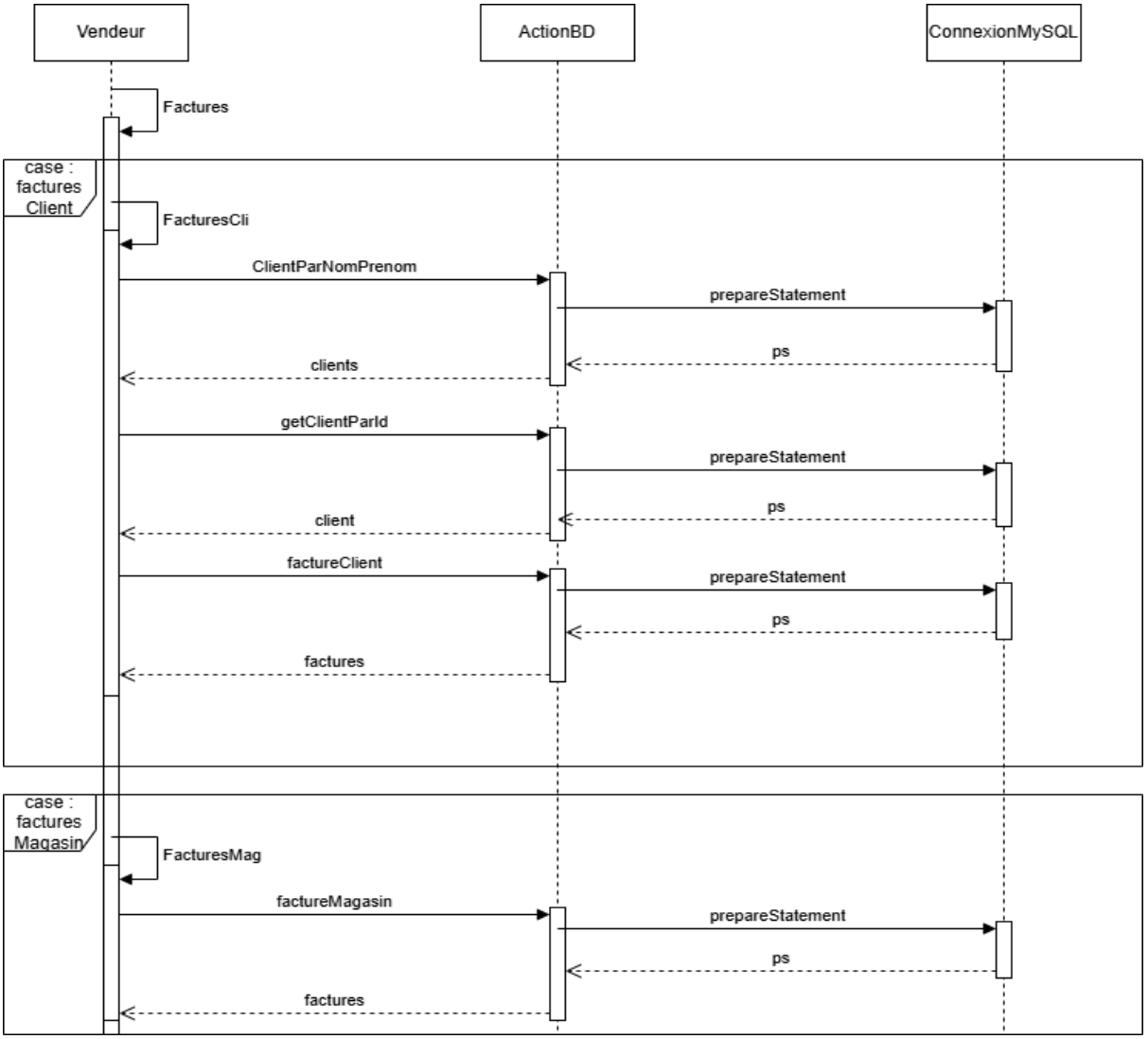

Figure 4 - Diagramme de séquence de editerFacture()

Description:
Ce diagramme (Figure 4) montre comment un vendeur édite une facture, soit pour un client, soit pour un magasin.
Dans le cas d'un client, le vendeur commence par rechercher le client avec son nom et prénom, récupère son ID, puis ses factures.
Dans le cas d'un magasin, il accède directement à la liste des factures du magasin.
Toutes ces actions passent par la classe ActionBD, qui prépare les requêtes SQL via ConnexionMySQL pour communiquer avec la base de données.
Ce processus permet d'afficher la facture à modifier ou à compléter.

Diagramme de séquence de la méthode onVousRecommande():

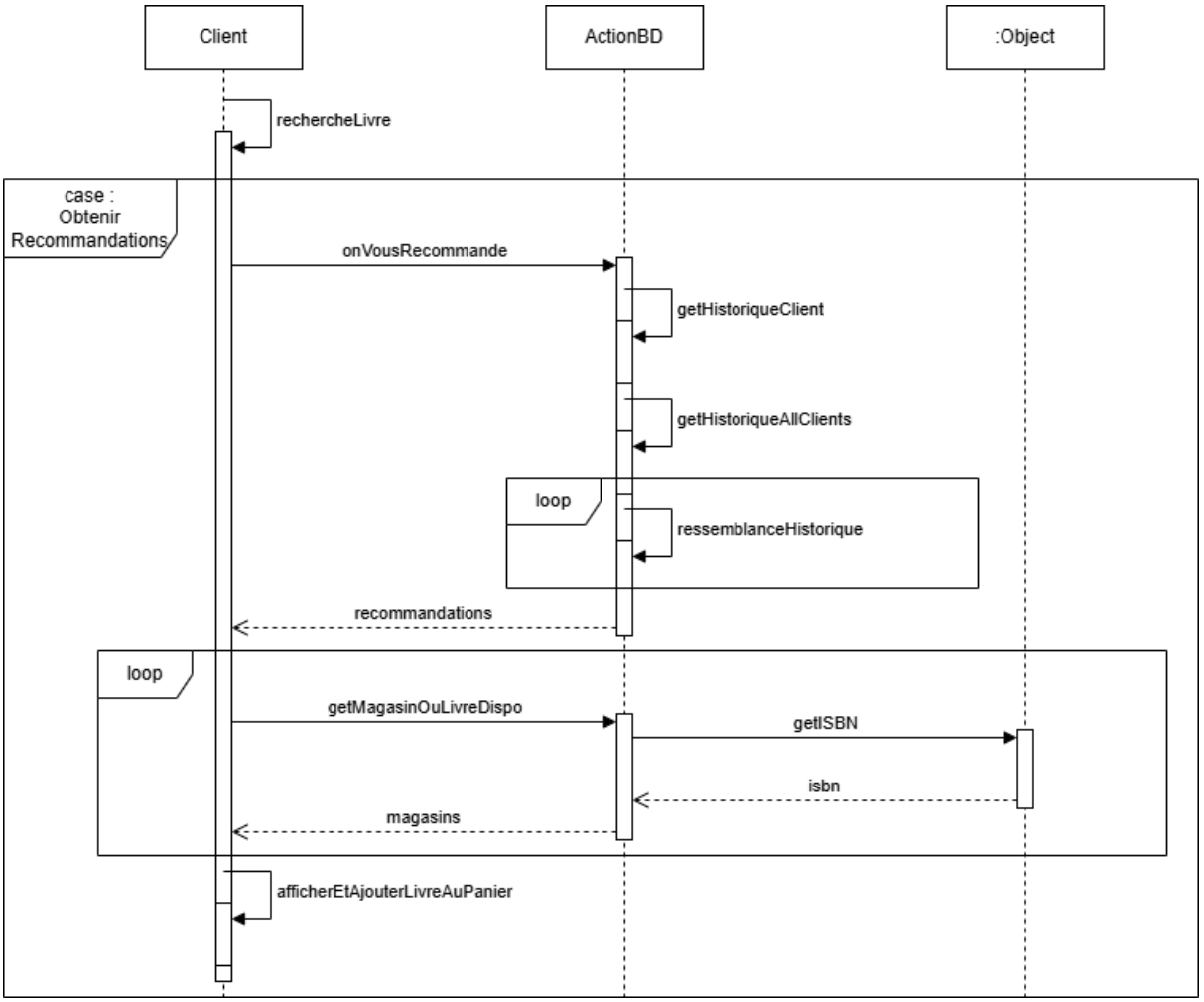

Figure 5 - Diagramme de séquence de onVousRecommande()

Description:

Ce diagramme de séquence (Figure 5) montre comment un client peut obtenir des recommandations de livres en fonction de son historique de lecture. quand le client recherche un livre, l'application appelle la méthode onvousrecommande() qui récupère l'historique du client et celui de tous les autres clients. ensuite, le système compare les historiques grâce à la méthode ressemblancehistorique() pour trouver des profils similaires. Cela permet de générer une liste de livres recommandés. Pour chaque livre, le système vérifie dans quels magasins il est disponible en utilisant son isbn. Une fois cette étape terminée, le client peut voir les recommandations et ajouter les livres au panier s'il le souhaite.

C. Diagramme de cas d'utilisation

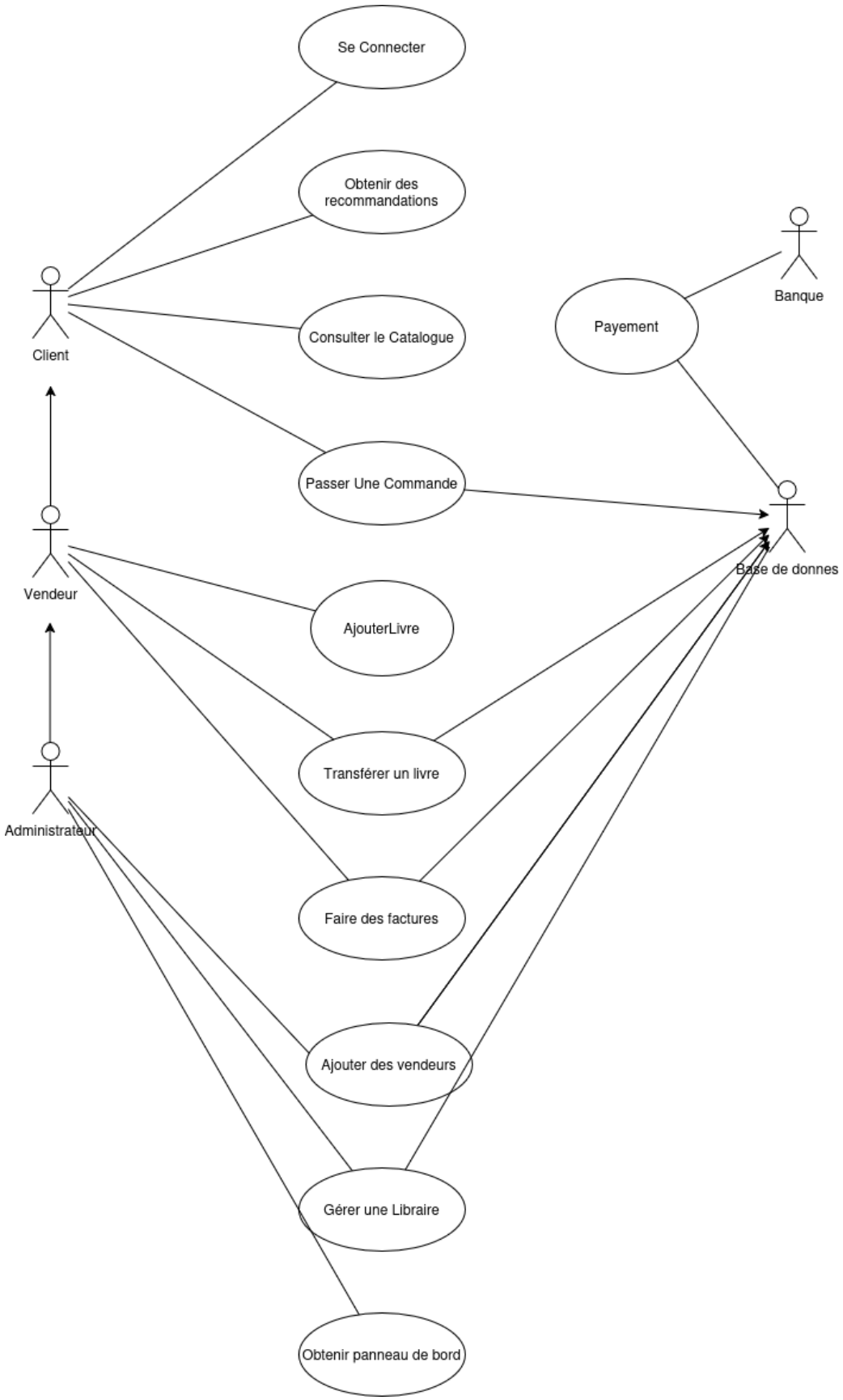

Figure 6 - Diagramme de cas d'utilisation

La figure 6 représente un diagramme de cas d'utilisation représentant les échanges entre les acteurs et les cas d'utilisations de notre application. Ce dernier montre qu'un client a la possibilité de se connecter, mais aussi de consulter le catalogue, de passer une commande, ainsi que d'obtenir des recommandations de livres. De plus, nous voyons que les vendeurs ont accès à des fonctionnalités similaires à celles des clients. En effet, il peut effectuer les actions de passer une commande, consulter le catalogue, ainsi que de se connecter. Mais il peut aussi modifier le stock, en ajoutant un livre ou en changeant la quantité présente dans le stock. Il a aussi la possibilité de demander le transfert d'un livre. Enfin, les utilisateurs administrateurs ont accès à toutes les fonctionnalités disponibles pour les utilisateurs vendeurs. S'ajoute à ces fonctionnalités, la capacité d'ajouter des utilisateurs, ainsi que des librairies. De plus, ils ont accès au panneau de bord. Il peuvent dernièrement faire des factures. Tous les cas d'utilisation cités précédemment pour tous les utilisateurs, font appel à l'acteur Base de Données qui s'occupe de la communication entre le programme et la base de données.

IV. Elaboration des Interfaces Homme Machine

V. Implémentation

Pour commencer l'implémentation, on a décidé de suivre notre schéma UML en créant toutes les classes qu'il contient. On a d'abord fait une version simple de chaque classe, avec les constructeurs, les getters et les setters pour tous les attributs dont on avait besoin ainsi que les methodes dont nous avions deja connaissance et la certitude qu'elles nous seraient utiles. Cette méthode nous a permis de garder une bonne organisation dès le départ, et d'éviter de devoir revenir en arrière plus tard en se rendant compte qu'il manquait une classe ou un attribut.

Ensuite, on a commencé à implémenter le code des méthodes utiles pour notre application, notamment dans la classe Commande. Par exemple, on a codé le calcul du nombre total d'articles dans une commande, ainsi que le prix total du panier. Ces méthodes nous permettent d'afficher des infos importantes directement dans l'appli, et elles seront pratiques pour l'utilisateur.

Une fois ces bases en place, on a commencé à faire une première version test de l'interface dans le terminal. Le but était de pouvoir tester le code plus facilement et de vérifier que les liens entre les classes fonctionnaient comme prévu.

Enfin, cette interface en mode terminal nous aide aussi à imaginer à quoi ressembleront les vraies fenêtres de l'application qu'on va bientôt développer avec JavaFX. Ça nous donne une première idée de la structure qu'on veut mettre en place pour l'interface graphique.

VI. Travail de groupe (cf. SAE Gestion de projet)

Au cours de cette SAE, nous avons collaboré en équipe de quatre, ce qui nous a permis de mettre en pratique nos compétences en travail d'équipe et en gestion de projet. L'ensemble du travail a été réalisé pendant les heures de SAE spécifiquement allouées à ce projet, ce qui nous a donné l'opportunité de nous concentrer pleinement sur notre mission sans distractions extérieures.

Nous avons structuré notre travail grâce à notre chef de projet, Raphaël, qui a orchestré toute l'organisation en attribuant à chacun une tâche et une partie bien définie de la SAE. Son leadership a été crucial pour maintenir le cap et s'assurer que nous avancions tous dans la même direction. Raphaël a non seulement défini les objectifs globaux du projet mais a également veillé à ce que chaque membre de l'équipe comprenne parfaitement son rôle et ses responsabilités.

Conformément à ses directives, nous avons élaboré le MCD collectivement grâce à l'outil Draw.IO, que nous avons connecté à google drive afin de travailler en simultané dessus. Nous avons décidé de le faire en groupe car il était essentiel que nous soyons tous d'accord sur les informations à inclure dans notre base afin de partir sur des bases solides. Ce processus a été important pour aligner nos visions et garantir que le projet soit développé de manière cohérente. Pour travailler efficacement et minimiser le temps perdu, chacun a créé différentes tables du MCD, et nous avons peaufiné les détails ensemble à la fin. Cette approche collaborative nous a permis de bénéficier de perspectives variées et d'améliorer la qualité de notre travail.

Par la suite, nous avons commencé à développer le projet en Java. Sur ce projet, Raphaël nous a également demandé de nous concentrer sur des tâches différentes, ce qui nous a permis de contribuer de manière significative au projet. Une personne gérait la connexion JDBC, une tâche cruciale pour assurer la communication entre notre application et la base de données. Deux autres commençaient l'implémentation des commandes, ce qui a été essentiel pour permettre aux utilisateurs d'interagir avec notre application. La dernière personne créait les jeux d'essais nécessaires ainsi que tout le reste, s'assurant que le projet soit complet et fonctionnel.

Pour partager le code que nous avons développé, nous avons utilisé GitHub avec un référentiel commun où nous avons tous codé sur notre propre branche (espace de travail personnel), en faisant valider notre travail par notre chef de projet (qui gérait GitHub) pour savoir si nous pouvions fusionner avec la branche principale. Grâce à cela, nous avons pu implémenter chacun de notre côté tout en ayant accès au code des autres grâce à des merges (action de fusionner une branche contenant une nouvelle implémentation à la branche principale) et des pulls(l'action de récupérer de la branche principale sur notre espace de travail local). De plus, cela nous a permis de corriger des erreurs en revenant en arrière si un problème survenait malgré tout sur la branche principale. Ce système de gestion de code nous a permis de travailler de manière efficace et de maintenir un haut niveau de qualité tout au long du projet.

Numéro de tâche	Libellé	durée	Tâches antérieurs
A	Exploitation BD	5 semaines (10h)	
B	Gestion de Projet	4 semaines (6h)	
C	Projet Java	7 semaines (17h)	A
D	Graphes	7 semaines (14h)	A
E	Réseaux	3 semaines (6h)	A--C
F	Développement de l'app	1 semaine (35h)	A--C--D--E
G	Rédaction du rapport	8 semaines	A--B--C--D--E--F

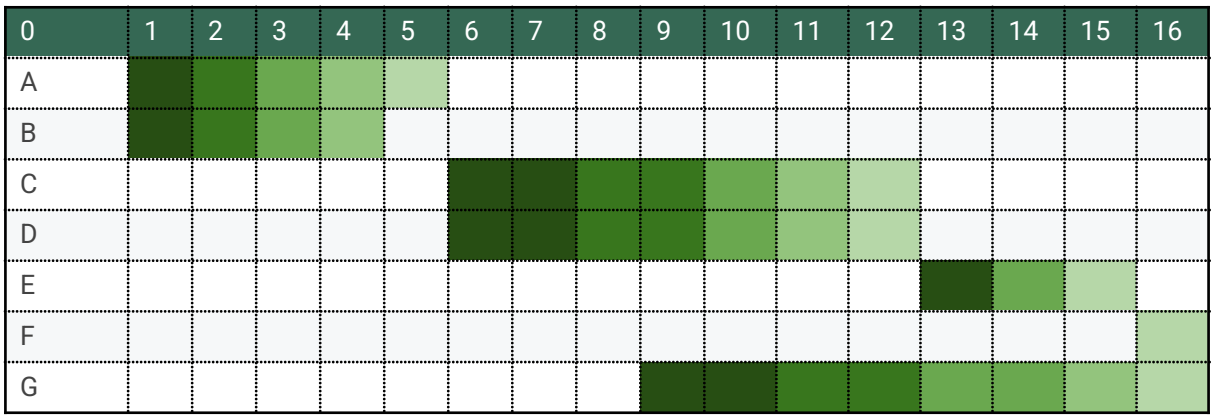

Figure 8 - Diagramme de Gantt

VII. Bilan du groupe

Nous avons réussi à implémenter toutes les fonctionnalités attendues dans la commande originelle, et ce pour chaque utilisateur. Toutes ces fonctionnalités ont réussi à passer les tests auxquels nous les avons soumis, ce qui semblerait indiquer que l'application fonctionne comme attendu. Cependant, nous n'avons pas réussi à finir les tests de toutes les fonctions. Nous nous sommes organisés de manière à ce que Raphaël s'occupe en partie du rapport, mais principalement de l'interaction avec la base de données et des tests. Corentin s'occupait des interfaces et interactions dans le terminal avec l'utilisateur ainsi que de certaines parties du rapport. Robin s'occupait partiellement de l'interaction avec la base de données, de certaines parties du rapport, ainsi que de l'interaction avec les utilisateurs au travers du terminal. Enfin, Joris s'occupait du rapport dans son ensemble et de l'édition de factures. Nous avons néanmoins modifié les tâches et sommes sortis de nos assignations pour aider les autres et compléter le projet dans les temps impartis. Au cours de cette SAE, nous avons tous appris et grandement évolué dans l'art de conduire un projet dans son ensemble. Nous avons aussi largement évolué dans nos compétences de programmation en Java, mais aussi dans nos compétences de collaboration et de communication, ainsi que dans l'interaction entre un programme Java et une base de données mysql. Nous trouvons tous que cette SAÉ nous a aidé à évoluer en tant que développeurs en nous apportant une expérience des plus concrète sur la gestion d'un projet integrale.

VIII. Annexes

Rapport individuel Robin FAUCHEUX

Au cours de cette SAE dédiée à la réalisation d'une application dans le terminal permettant l'exploitation d'un réseau de librairies, nous devions imaginer et implémenter en java une application complète rendant possible la gestion pour différents types d'utilisateurs du réseau de librairies Livre Express.

Réalisations personnelles

1. UML

J'ai tout d'abord participé à l'élaboration du diagramme de classe UML représentant notre application, en collaboration avec le reste du groupe. J'ai ensuite réalisé, avec l'aide de Corentin, le diagramme de Cas d'Utilisation. J'ai enfin réalisé les trois diagrammes de séquence.

2. Java

JDBC

- PasserCommande
- addVendeur
- addLivre
- updateStock
- voirStockMag
- transferer
- addLibrairie
- getMaxNumCom
- getIdUserMax
- getClientNomPrenom
- getClientParId
- getCurrentDate
- getLivreParId
- getMaxISBN
- factureClient (travaille la fonction de Joris)
- factureMagasin (retravaillé la fonction de Joris)

Commande

- livraison
- rechercher
- menu_rechercher
- panier

Vendeur

- ajouterLivre

- ipdateStock
- disponibilites
- passerCommande
- Tranfer
- FacturesMag
- FacturesCli
- Factures
- application

Administrateur

- application (Avec Corentin)
- ajouterLivrairie
- panneauBord (Reprise du code de Raphaël)

Afficher Menu

- Première version de la fonction, ensuite améliorée par Raphaël

3. Rapport

J’ai réalisé la partie analyse du diagramme de classe, ainsi que l’analyse du diagramme de séquence de PasserCommande, ainsi que l’analyse de diagramme de cas d’utilisation. De plus, j’ai réalisé l’introduction du rapport, ainsi que la section du bilan du groupe. J’ai aussi effectué l'introduction de l’analyse du MCD.

4. Précision

Je n'exclus pas la possibilité d' avoir effectué d’autres tâches au cours de cette SAE, mais si c’est le cas, je ne m’en rappelle pas ou elles ne sont pas très significatives.

Conclusion

Ce projet m’a permis de m'améliorer dans des compétences telles que JDBC grâce à la réalisation de fonctions interagissant avec la BD, Java en aidant au développement des interactions avec l’utilisateur, mais aussi car JDBC est une librairie Java. J’ai aussi évolué dans les compétences de communication et de collaboration en effectuant ce projet avec mes camarades.

Rapport individuel Vachey Joris

1. Introduction

Présente rapidement le contexte du projet :
Dans le cadre de la SAE 2.01, nous avons travaillé sur le développement d'une application Java pour le groupe de librairies Livre Express, afin de permettre aux vendeurs et aux clients de gérer les commandes et les stocks de livres. Ce projet s'inscrit dans la continuité de la SAE 2.04 où nous avions mis en place la base de données.

2. Mon rôle dans le projet

Conception des diagrammes UML
Implémentation du code de base pour les méthodes.
Participation à la création du rapport
implémentation des factures
Fichiers de Test

3. Compétences développées

AC11.01 : J'ai implémenté plusieurs classes et méthodes en Java de manière structurée.
AC11.02 : J'ai produit des conceptions UML cohérentes avec le besoin client.

4. Difficultés rencontrées

Ayant plus de mal que mes camarades en java, je me suis retrouvé à ne pas savoir quoi faire. J'ai donc choisi d'aider pour ce que j' ai pu en terme de code, tel que mettre le code necessaire pour qu' il n y ai pas de message d' erreurs lors des tests, les factures, faire l uml et le rapport.

5. Apports de la SAE

J'ai appris à structurer une application Java complète, en respectant les principes d'architecture logicielle.

J'ai renforcé mes compétences en communication technique grâce aux rapports..

6. Conclusion

Ce projet m'a permis de mieux comprendre les étapes concrètes de développement logiciel, de la conception jusqu'aux tests. J'ai aussi pris conscience de l'importance de la rigueur dans le code et la collaboration via Git. Cette SAE me donne envie de progresser en Java car la création d'une application est vraiment très intéressante.

Rapport Individuel SAE Java - Corentin LACOUME

Tâches réalisée:

Dans cette SAE, j'ai participé à la création d'une application en Java. Je me suis occupé de l'implémentation des menus, donc de la partie navigation de l'application. J'ai aussi fait l'explication du modèle conceptuel de données (MCD). J'ai apporté une aide ponctuelle sur l'explication de certaines requêtes SQL.

J'ai contribué au diagramme de cas d'utilisation, à la mise en page du rapport, et à l'intégration des diagrammes de séquence. J'ai également écrit une brève description des méthodes onvousrecommande() et editerfacture().

Cette SAE m'a permis de mieux comprendre les liaisons entre une application Java et une base de données MariaDB. Au début, j'ai eu du mal à saisir comment les données circulent entre les deux, mais en travaillant sur ce projet, j'ai pu voir concrètement comment les requêtes SQL étaient intégrées dans le code Java, ce qui m'a aidé à faire le lien.

Ce projet m'a aussi permis de consolider les bases en Java que j'ai apprises tout au long du semestre, en les appliquant dans un vrai projet structuré. Le fait d'avoir travaillé sur l'implémentation et l'organisation des différentes parties du programme m'a aidé à mieux maîtriser la logique du langage.

Rapport individuel – Raphaël BROUSSEAU

Dans le cadre de la SAE dédiée au développement d'une application pour LIVRE EXPRESS, notre groupe avait pour mission de concevoir un système complet de gestion de librairie en ligne, intégrant notamment la gestion des utilisateurs, des commandes, et des statistiques commerciales.

Nous avons débuté par une conception UML commune, à laquelle j'ai participé en apportant mes idées pour répondre aux exigences du cahier des charges.

Après cette phase de conception, le travail a été réparti équitablement entre les membres de l'équipe. Voici un récapitulatif des fonctionnalités que j'ai réalisées personnellement :

Connexion à la base de données

- Mise en place de la connexion JDBC via `ConnexionMySQL`.

Système de recommandation personnalisée

Développement complet du module de recommandation de livres basé sur l'historique d'achat des clients :

- `onVousRecommande(Client client)`
- `ressemblanceHistorique(List<Livre> historiqueClient1, List<Livre> historiqueClient2)`
- `getHistoriqueClient(Client client)`
- `getHistoriqueAllClient()`

Fonctions du tableau de bord administrateur

Codage de l'intégralité des fonctions statistiques utilisées pour les visualisations de l'administrateur :

- `NombreDeLivreVenduParMagasinParAns()`
- `chiffreAffaireParClassificationParAns(int annee)`
- `CAMagasinParMoisParAnnee(int annee)`
- `CAVenteEnLigneEnMagasinParAnnee(int anneeAExclure)`
- `nombreAuteurParEditeur()`
- `nombreClientParVilleQuiOntAcheterAuteur(Auteur auteur)`
- `valeurStockMagasin()`
- `statsCAParClientParAnnee()`
- `auteurLePlusVenduParAnnee(int anneeExclu)`

Menus clients et recherches avancées

Développement des menus clients ainsi que de toutes les fonctionnalités de recherche de l'application :

Recherches de livres

- `cherhcherClassificationApproximative(String nomClass)`
- `chercherLivreAPartirClassification(Classification classi)`

- `cherhcherEditeurApproximative(String nomEditeur)`
- `chercherLivreAPartiEditeur(Editeur editeur)`
- `rechercheAuteurApproximative(String nomauteur)`
- `cherhcherLivreApproximative(String nomApproximativeLivre)`
- `rechercheLivreAuteur(Auteur auteurRecherche)`

Recherches de clients

- `getClientAPartirNomPrenomCodePostal(...)`
- `getClientNonPrenom(...)`
- `getClientParId(...)`

Recherches de livres par identifiants

- `getLivreParTitre(...)`
- `getLivreParId(...)`
- `getLivreParIddewey(...)`

Recherches liées aux magasins

- `getMagasinOuLivreDispo(Livre livre)`
- `magAPartirNom(String nomMagasin)`
- `magAPartirId(Integer idmag)`
- `getMagasinParId(Integer idmag)`

Test des fonctions JDBC

- J’ai participé avec joris à la réalisation du fichier de test `TestBD.java`, qui comprend les tests unitaires de toutes les méthodes utilisant JDBC.

Menu Client

- J'ai réalisé la totalité des menus client cad :
- `Application()`
- ect

Conclusion

Cette SAE m’a permis de valider le niveau 1 de la compétence « Réaliser un développement d’application », notamment à travers les apprentissages critiques suivants : **implémenter des conceptions simples** et **faire des essais et évaluer leurs résultats en regard des spécifications**, en développant des fonctionnalités clés telles que le système de recommandation ou les statistiques du tableau de bord.
Elle m'a aussi premis travailler dans un cadre de développement collaboratif et structuré.