Rapport projet Library

ADJERIT Marwan Jalis, AGOMADJE Bruneck Ulrich,
BERTHIER Théophile, BEWEKEDI Benjamin,
CHARLES Jean-Smith

UE Projet, Projet génie logiciel

24/05/2022

Table des matières

1 Introduction

1.1 Présentation du livrable :

Créer une application permettant de gérer un stock de livres, tout en considérant le fait que des clients puissent les emprunter.
Cette même application doit comprendre une base de données et une interface graphique, il doit être possible de chercher et afficher un livre.
Il doit être possible d'enregister les données des utilisateurs, des livres et des emprunts et utiliser un api pour récupérer les données concernant les articles.

2 Présentation

2.1 Structure du projet

Afin de se donner une idée de la forme que va prendre le projet, il est nécessaire de s'appuyer sur des diagrammes pour définir le fonctionnement global de l'application. Pour commencer, nous avons réalisé le diagramme d'activité qui modélise les interactions d'un visiteur mais aussi du libraire avec l'application :

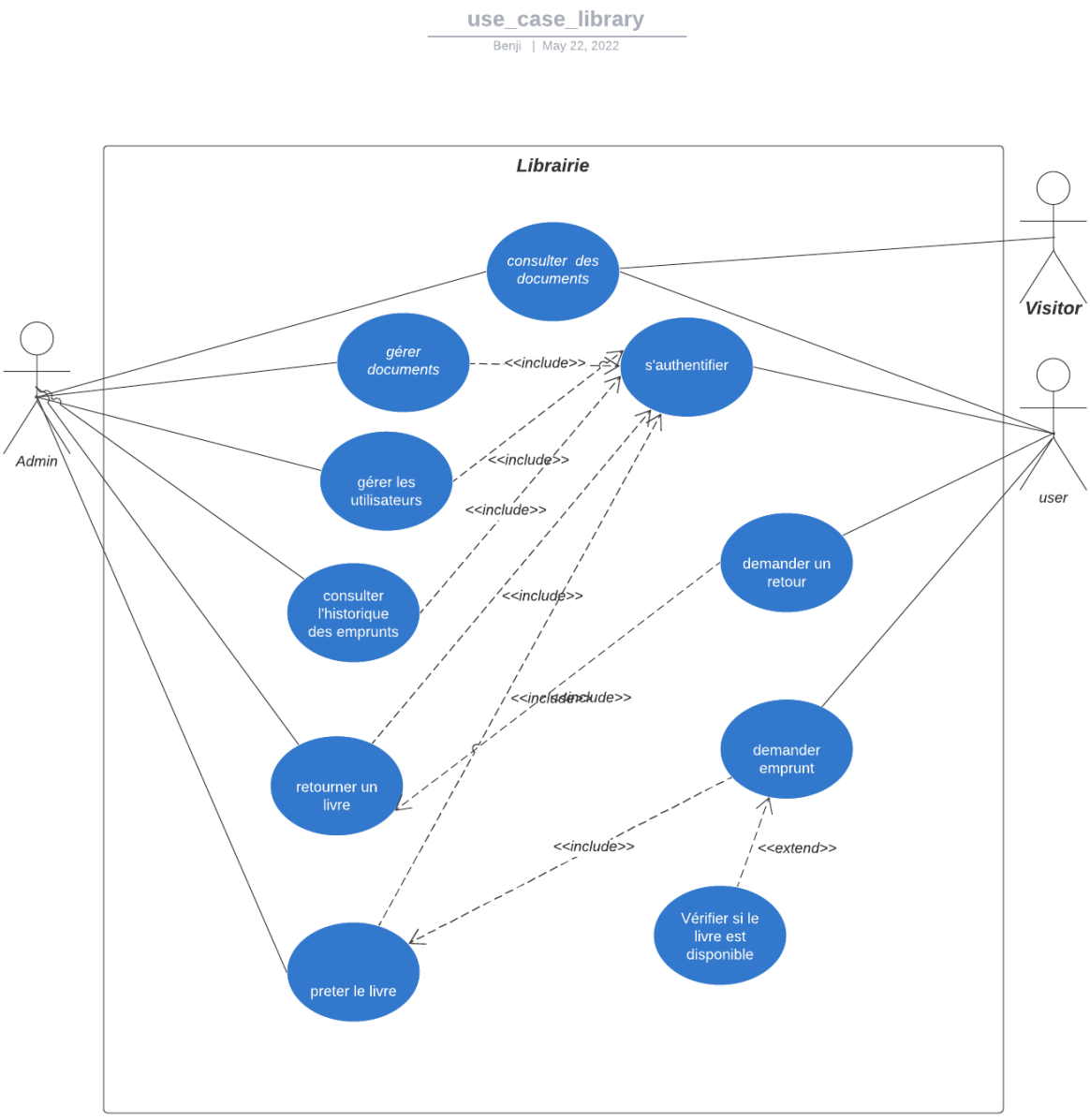


Figure 1 – Diagramme d'activité de la librairie

Par la suite, nous avons également réalisé le diagramme de classe, très utile pour la structure du projet en java car il nous donne une idée des classes, variables et méthodes à implémenter.

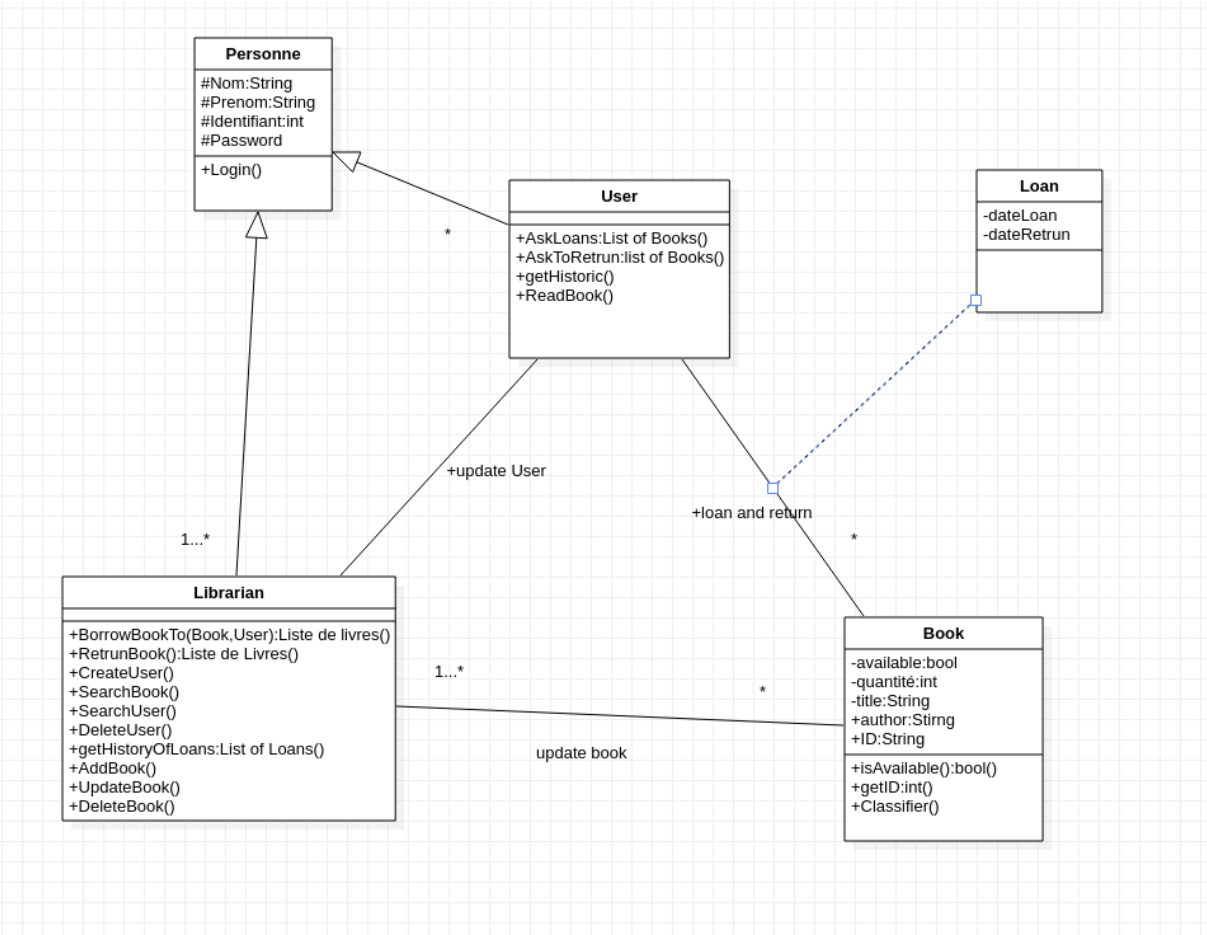


Figure 2 – Diagramme de classe de la librairie

Cela nous permet facilement de comprendre que nous aurons à créer la classe livre, la classe User et Librarian tous deux héritant de la classe personne.
-User peut : demander/retourner une location, regarder son historique.
-Librarian peut : créer/supprimer un user, chercher/prêter un livre, rechercher un User, ajouter/modifier/supprimmer un livre, regarder l'historique des prêts.
Ils ont tous deux un nom, un prénom, un identifiant, un mot de passe et la possibilité de se conneter.

Bien évidemment ce diagramme résume seulement dans la globalité l'application mais nous allons voir par la suite que bien plus de classes composent ce projet.

2.2 Partie Back-end

Comme dit précédemment, dans cette partie nous allons montrer les différentes classes avec leurs méthodes ainsi que la façon dont elles interagissent avec la base de données.

Book, Librarian, Loan et User

Ce sont les trois principales classes du projet, commençons par la classe book :

Book

Elle possède les variables suivantes :

```
private String title;
private String author;
private int idBook;
private int isbn;
private int idUser;
```

```
@Override
public String toString() {
    return "Book{" +

            ", title='" + title + '\'' +
            ", author='" + author + '\'' +
            ", ID='" + idBook + '\'' +
            '}';
}
```

Chacune à un getter et un constructeur, à noter que nous avons modifié la méthode toString() de cette classe.

User

Puis ensuite la méthode User nous avons les variables suivantes :

```
public class User {
    final String Name;
    final String LastName;
    protected int ID;
    protected String Password;
    private List<Book> Books;
```

Ici nous cherchons à rendre les variables Names et LastName final car c'est inchangeable pour chaque personne, l'Id et le password ne peuvent être qu'utilisés dans la classe ou ses dérivés. Nous avons également une liste Books de la classe Book pour les livres possédés par un utilisateur.
Nous avons ajouté en plus des getter et constructeurs les méthodes addBook (ajouter un livre à la liste), removeBook (en supprimer un) et GetListofBorrowedBooks (retourne tous les livres de la liste du User en cours)

Librarian

Cette classe hérite de User, il peut afficher les informations d'un User donné et peut également afficher sa liste d'emprunts.

```
public String getListofBorrowedBooksbyUser(User p1) { return "User:"+p1.Name+" "+p1.getListofBorrwedBooks();
```

Loan

Cette classe permet de gérer les emprunts en tenant compte de la date de retour.

Nous avons 7 variables de bases : L'id de User, l'id du livre, la date d'emprunt, la date limite d'emprunt, la date actuelle, le titre du livre et son statut.

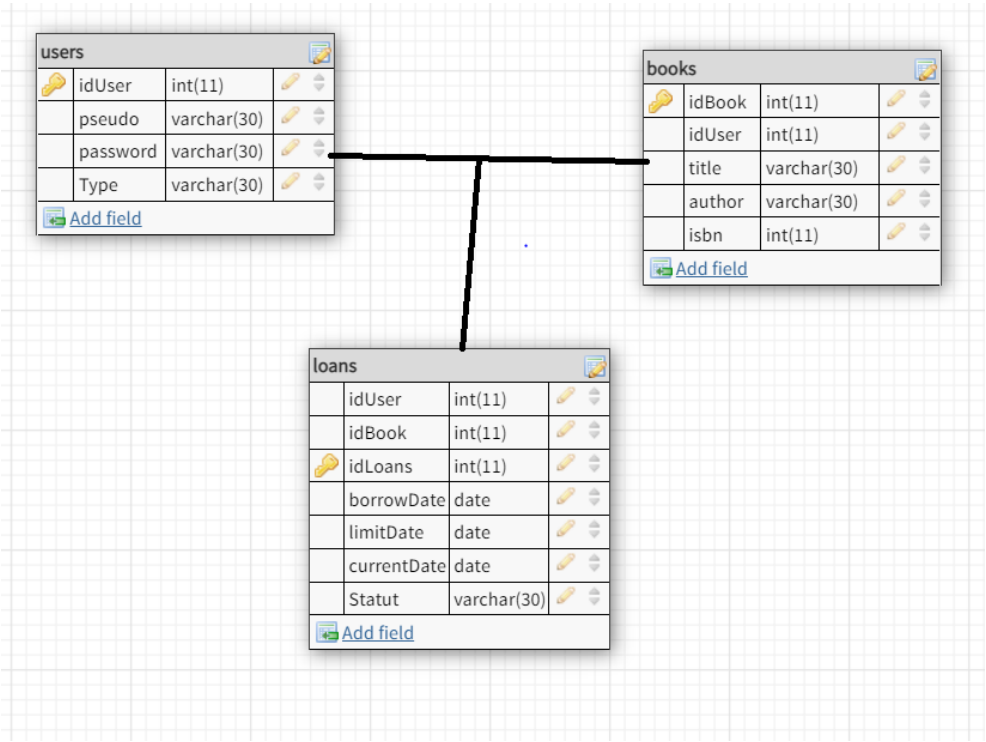
Il y a un getter et setter pour chacune des variables, en revanche nous utilisons la surcharge pour les constructeurs car il y a plusieurs cas de figures possibles.
En effet le minimun nécessaire pour instancier cette classe est idUser, idBook et borrowDate.

A part cela la classe ne possède pas de méthodes particulières.

Maintenant que nous avons défini les principales classes du projet, passons aux classes qui utilisent la base de données.

Les classes de la base de données

Voici dans un premier temps la relation entre les tables que nous utiliserons ainsi que leurs structures :



Connexion

Nous avons importé dans cette classe les packages Connexion et DriverManager pour ajouter les méthodes nécéssaire pour se connecter à une base de données.
La classe possède une méthode : connectorDB() qui va :
- Créer une nouvelle instance de com.mysql.jdbc.Driver, de ce fait nous pouvons créer des connexions mysql basé sur un URL
-Définir l'URL, et les identifiants pour ensuite effectuer la vraie connexion.

De plus, si une erreur se produit durant l'execution de cette méthode, nous avons un try catch qui retourne l'erreur dans ce cas et arrête donc la tentative de connexion.

```
Class.forName("com.mysql.jdbc.Driver");
System.out.println("Driver oki");
String url = "jdbc:mysql://localhost:3306/library";
String user = "root";
String password = "Zoo01022001**";
Connection cnx = DriverManager.getConnection(url, user, password);
System.out.println("Connexion bien établie ");
```

Management (a servi pendant la phase de test)

Cette n'est pas utilisé dans la version actuelle de l'application mais on a jugé bon d'en parler car elle a servi aux différents test et a parmi par la suite de penser sa version grapiphique.
Cette classe est assez conséquente et à quatres fonctionnalitées :
-La méthode register :
Elle prend en entrée un pseudo et un password, ensuite nous utilisons la méthode ConnectorDB pour se connecter à notre base de données.
Par la suite nous créeons la variable req qui est un string comportant les instructions SQL pour ajouter un User.
Nous allons donc ensuite tenter d'executer cette requête avec un statement, à noter que si il y a une erreur dans tous le procédé nous avons un try catch pour retourner ces erreurs.

-La méthode connection :
Elle prend en entrée un pseudo et un password, le début du procédé est le même que la méthode précédente sauf que le but de notre requête ici est de vérifier qu'un User existe (on utilise donc un SELECT)
Nous cherchons l'ID de l'utilisateur puis nous cherchons le livre emprunté.
On a deux cas :
1- Le Type de la personne connecté est User :
On crée une nouvelle instance d'un User appelé user1 qui sera retourné par la méthode.
On à également la requête res2 qui est un ResultSet permettant d'afficher un à un les livres empruntés.

```
while (res2.next()) {
    // print the borrowed books
    System.out.println("Vous avez emprunté " + res2.getString( columnLabel: "title"));
}
```

2- Le Type de la personne est Librarian :
On crée user comme précédement sauf qu'il s'agit de la classe Librarian.

Par la suite, nous avons un catch(SQLException throwables) qui retourne l'erreur ainsi que le message "L'utilisateur n'existe pas"

-La méthode loanBooktoUserDB :
Prend en entré le pseudonyme de l'utilisateur ainsi que l'id d'un Book
En suivant la même logique de connection que les deux méthodes précédentes, cette dernière à deux requêtes SQL :
Une pour modifier dans books l'id du User qui le possède actellement et l'autre pour ajouter à 'loans' un couple (idUser, idBook) pour signaler qu'un User detient un Book en particulier.
On execute les deux requêtes, si c'est bon la console affiche "Emprunt réussi" sinon une erreur est retournée à l'aide du catch.

-La méthode ReturnBooktoUser :
Elle prend en entrée le titre du livre ainsi qu'un entier NULL
Le processus de connexion est toujours le même mais ici nous allons enlever l'id du détenteur du livre dans books, pour faire en sorte que le livre ne soit plus emprunté.

```
//SQL Request for deleting the current
String req = "UPDATE books SET idUser = "+NULL+"WHERE idBook = (SELECT idBook FROM books WHERE title="+title+");";
```

Si cela réussit la console affiche "Retour Réussi", sinon on retourne l'erreur.

UserSession
Cette classe est finale car on ne cherche pas à le redéfinir, elle n'a que 3 variables, userName, idUser et instance qui est du type de notre classe (UserSession).
Nous avons le constructeur pour userName et idUser et une méthode getInstance qui crée une nouvelle instance de UserSession s'il n'y en a pas puis la retourne dans tous les cas.
Il y a également la méthode cleanUserSession qui vide de champ de userName et met un id à 0.

MainBdd (a également uniquement servi pendant la phase de test)

Cette classe centralise les actions que peut choisir de faire l'utilisateur sur l'application en lien avec la base de données (se connecter etc...)
Pour ce faire elle doit importer les classes présentés précédemment Connexion, Management et User.
On ajoute à cela java.sql.* et java.util.Scanner pour recupérer les information entrées par l'utilisateur.

Le reste est une suite d'instruction visant à tester les différentes méthodes des autres classes (on peut même voir en commentaire certains appels à des méthodes).

2.3 Partie Front-end

Login Controller

Cette classe permet de gèrer la page de connexion de notre application (Login.fxml), nous avons 4 méthodes : LoginClicked() qui est une méthode évenement lié au bouton Log In, cette méthode appelle la méthode CheckLogin() cette méthode permet de vérifier si l'utilisateur a saisi le pseudo associé à un mot de passe, parmis les utilisateurs on distingue deux types : Librarian et User, selon qui s'est connecté,il sera redirigé vers une autre page : Librarian.fxml pour Librarian et dashboard.fxml pour User. On trouve également la méthode SignupClicked() lié au bouton Sign up qui nous renvoie vers une autre page : Signup.fxml

Nous avons réutilisé cette structure combinant une page graphique javafx au format .fxml associée à une classe "Controller" pour former et articuler l'ensemble de l'application.

Signup Controller

Cette classe permet de gèrer la page d'inscription de notre application(Signup.fxml), elle contient 4 méthodes : SignupRegister() qui est une méthode lié au bouton Sign Up, cette méthode appelle la méthode UserRegistration() qui prend en paramètre le pseudo et le mot de passe de l'utilisateur, avec une requête SQL la méthode va insérer le nouvel utilisateur dans la base de données dans la table "users".

borrowBook Controller

Cette classe permet de gérer les emprunts des livres, elle contient les méthodes : borrowClicked ié à au bouton Borrow, elle appelle la méthode BorrowBook() qui prend en paramètre l'id du livre et l'id de l'utilisateur, elle met à jour le champ idUser qui prend la valeur de l'id de l'utilisateur dans la table "books" et elle ajoute le nouvel emprunt dans la table "loans".La méthode initialize() est une méthode qui permet de gèrer l'affichage des emprunts, c'est à dire à chaque nouvel emprunt, la page borrowBook.fxml se met à jour et affiche les emprunts.

BookManagement Controller

Cette classe permet la gestion des livres par le libraire, il peut mettre à jour un livre en cliquant sur le bouton Update qui appelle la méthode UpdateBook() qui prend en paramètre le titre, l'autheur et le numéro isbn, la méthode met à jour les valeurs des champs du livre passé en paramètre dans la table "books". Le libraire peut également ajouter un nouveau livre dans la base de données en cliquant sur le bouton Add,qui appelle la méthode addBooktoDB() qui prend en paramètre un titre, un autheur et numéro isbn, avec une requête SQL =, on va insérer le

nouveau livre dans la tbale ”books”. Enfin le libraire peut décider de supprimer un livre de la base de données en cliquant sur la bouton Delete, qui appelle la méthode deleteBooktoDB() qui prend en paramètre l’id du livre à supprimer

ReturnBook Controller

Cette classe permet de rendre un livre à la librairie, la méthode ReturnClicked() permet de mettre à jour l’id de l’utilisateur qui possède le book dans la table ’books’, mettre à jour la date et mettre le statut du livre à ”termine” dans la table ’loans’

LoanManagement Controller

Cette classe permet au libriare de gérer les emprunts, la méthode acceptation() donne l’autorisation à un utilisateur d’emprunter un livre, ainsi une date limite ainsi que le statut ”en cours” sont mis a jour dans la table ’loans’ et la console affiche ”Emprunt réussi”.
Dans le cas échéant, il est possible de refuser un emprunt avec la méthode denyLoan(), alors il retire le livre de la table ’loans’ et enleve l’id de l’user qui emprunte de la table ’books’.

LibrarianMenu Controller

Cette classe permet au libraire de naviguer entre les différentes scenes de l’application qui lui sont reservés, ainsi BookSceneChange() redirige vers la scene LibrarianDashboard, UserSceneChange() vers UserManagement, LoanSceneChange() vers LoanManagement et LoanListSceneChange() vers LoanList.

UserManagement Controller

Cette classe permet d’interagir avec la table ’users’, elle permet 3 choses :
-La méthode addUser() qui permet d’ajouter un utilisateur dans la table ’users’ à l’aide de ses informations.
-La méthode updateUser() qui modifie les informations relatives à ce dernier.
-La méthode deleteUser() qui supprimer un utitilisateur de la table ’user’

3 Conclusion

Pour conclure ce rapport, si il y a bien une chose que ce projet nous a permis d'apprendre, c'est la méthode d'organisation du travail.

En effet, une fois que nous nous sommes mis d'accord sur une structure de travail séparant le front-end et la back-end tout en effectuant des suggestions pour le front-end et en participant collectivement au débogage du code, notre productivité s'est multipliée.

Il s'agit là d'un point satifsfaisant qu'il nous faudra nous remémorer pour nos projets à venir.

Aussi, un des seuls regrets que nous ayons eu au cours du projet, est le fait que nous ne sommes pas parvenus à intégrer l'usage d'une API pour compléter notre base de donnée. En effet, intégrer une API même si elle est complémentaire à la base de donnée initiale, s'est réveler re une tâche ardue de par l'utilisation d'un format nouveau au projet, qu'est le JSON et les recherches que nous aurions dû faire car les API étaient une nouvelle notion pour nous. Cependant, nous avons fait un effort de recherche et il ne fait nul doute, qu'avec les connaissances amassées et inexploitées, nous pourrons intégrer cette technologie dans nos travaux futurs.