**REPUBLIQUE DU CAMEROUN**

**Paix-travail-patrie**

**\*\*\*\*\*\*\***

**MINISTERE DE L’ENSEIGNEMENT SUPERIEUR**

**\*\*\*\*\*\*\***

**UNIVERSITE DE DOUALA**





**REPUBLIC OF CAMEROUN**

**Peace-Work-Fatherland**

**\*\*\*\*\*\*\***

**MINISTRY OF HIGHER EDUCATION**

**\*\*\*\*\*\*\***

**UNIVERSITY OF DOUALA**

PROJET POO : DOCUMENTATION TECHNIQUE

SYSTEME DE GESTION D’UNE BIBLIOTHEQUE

**ANNEE ACADEMIQUE 2024-2025**

**Sous l’encadrement de :**

**Dr. NOULAPEU**

***Projet réalisé par :***

|  |  |  |
| --- | --- | --- |
| **Noms** | **Prénoms** | **Matricules** |
| **ASSONGUE TATANG** | **Muriel** | **22G00400** |
| **ATEKOUMBO** | **Excel Dexte Joris** | **24G01074** |
| **DONFACK KEUNANG** | **Vladimir** |  |
| **LAMBOWO** | **Marius** |  |
| **NGANSOP NGOUABO** | **Fredi Loïk** | **22G00307** |
| **NAMEUGNE** |  |  |
| **NOLACK KAWUNJIBI** | **Frange Parker** | **22G00335** |
| **NZEUTEM DOMMOE** | **Eunice Felixtine** | **22G00347** |
| **SEUMEGNE** |  |  |
| **TSEFACK** | **Calvin Klein** | **22G00540** |

# **SOMMAIRE**

[**I.** **ANALYSE DES BESOINS** 3](#_Toc198564954)

[**II.** **ENVIRONNEMENT DE DEVELOPPEMENT** 5](#_Toc198564955)

[**III.** **ARCHITECTURE LOGICIELLE** 6](#_Toc198564956)

[**IV.** **CONCEPTION UML** 7](#_Toc198564957)

[**V.** **STRUCTURE DU PROJET** 8](#_Toc198564958)

[**VI.** **DESCRIPTION DES MODULES** 8](#_Toc198564959)

[**VII.** **CONFIGURATION DE LA BASE DE DONNEES** 10](#_Toc198564960)

# **ANALYSE DES BESOINS**

## **Identification des utilisateurs**

### **Bibliothécaire :** Le bibliothécaire a accès à toutes les fonctionnalités de l’application y compris la gestion des documents, des emprunts et le suivi des emprunts.

### **Administrateur :** Les administrateurs ont accès à toutes les fonctionnalités de l’application, y compris la gestion des utilisateurs, des rôles et des paramètres de l’application.

## **Fonctionnalités requises**

### **Gestion des Livres**

* **Ajout d’un livre** : Saisir les informations de base (titre, auteur, ISBN, année de publication).
* **Suppression et modification***:* Permettre la mise à jour ou la suppression d’un livre existant.
* **Consultation de l’inventaire***:* Afficher la liste complète des livres disponibles.

### **Gestion des Emprunts**

* **Empreint d’un livre** : Enregistrer l’emprunt par un utilisateur (nom ; date d’emprunt).
* **Retour de livre** : Enregistrer le retour à mettre à jour le statut du livre.
* **Historique***:* Consulter l’historique des emprunts pour chaque livre ou utilisateur.

### **Interface Utilisateur**

* **Un menu principal** permettant de naviguer entre les fonctionnalités (gestion des livres, gestion des emprunts, consultation).
* **Gestion des entrées utilisateur** avec des validations de saisie.

***Projet réalisé par :***

|  |  |  |
| --- | --- | --- |
| **Noms** | **Prénoms** | **Matricules** |
| **ASSONGUE TATANG** | **Muriel** |  |
| **ATEKOUMBO** | **Excel Dexte Joris** | **24G01074** |
| **DONFACK KEUNANG** | **Vladimir** |  |
| **LAMBOWO** | **Darius** |  |
| **NGANSOP NGOUABO** | **Fredi Loïk** | **22G00307** |
| **NAMEUGNE** |  |  |
| **NOLACK KAWUNJIBI** | **Frange Parker** | **22G00335** |
| **NZEUTEM DOMMOE** | **Eunice Felixtine** | **22G00347** |
| **SEUMEGNE** |  |  |
| **TSEFACK** | **Calvin Klein** | **22G00540** |

### **Persistance des Données**

* Utiliser des fichiers texte ou un format simple (par exemple, CSV) pour sauvegarder et restaurer les données entre plusieurs exécutions de l’application.

## **Modalités d’accès et d’utilisation de l’application**

### **Accès à l’application**

* **Lancement**: L’application se lance depuis un terminal en exécutant la classe principale (par exemple : Main.java)
* **Authentification**. Chaque utilisateur doit se connecter avec ses identifiants. Selon le rôle (administrateur, bibliothécaire ou emprunteur), différentes options s’affichent.

### **Utilisation par profil**

* **Administrateur**
  + Gère les comptes des bibliothécaires (ajout, modification, suppression).
  + Consulte les rapports globaux (livres disponibles, statistiques d’emprunts).
* **Bibliothécaire**
  + Ajoute, modifie et supprime des livres.
  + Gère les copies de livres et les emprunts/retours.

# **ENVIRONNEMENT DE DEVELOPPEMENT**

Dans ce projet, nous avons utilisé plusieurs outils pour garantir une gestion efficace du code, une interface utilisateur intuitive et une base de données robuste. Voici un aperçu des technologies utilisées :

* **Langage de programmation : Java 17**
* **IDE : NetBeans 19**

Environnement de développement intégré offrant des fonctionnalités comme la gestion de projets, le débogage, l’auto-complétions de code et la gestion de bibliothèques externes.

I Intègre JavaFX, facilitant le développement d’interfaces graphiques modernes.

* **JavaFX 17**

Bibliothèque graphique évoluée pour la création d’interfaces utilisateurs.

Permet de séparer la vue (FXML) de la logique métier (Java), favorisant une structure MVC claire.

CSS intégré pour la personnalisation des interfaces.

* **SceneBuilder 19**

Outil de conception graphique permettant de créer des interfaces JavaFX en mode WYSIWYG (What You See Is What You Get).

Génère des fichiers FXML, facilitant l’interaction entre la vue et le contrôleur.

* **Base de données : MySQL 8.0**

Système de gestion de base de données relationnel (RDBMS) robuste et open-source.

Utilisé pour stocker toutes les informations sur les utilisateurs, les livres, les emprunts et les notifications.

* **Connexion : JDBC 4.3**

API standardisée pour la connexion à une base de données relationnelle.

Gère la création de connexions, l’exécution des requêtes SQL et la gestion des transactions.

* **XAMPP** :

Outil regroupant Apache, MySQL, PHP et Perl.

MySQL est utilisé comme serveur de base de données local pour le projet. Interface phpMyAdmin pour la gestion des tables, des utilisateurs et des requêtes SQL.

Ces outils permettent de structurer le projet de manière modulaire, de séparer clairement les différentes couches logiques et de maintenir une cohérence tout au long du cycle de développement.

# **ARCHITECTURE LOGICIELLE**

L'architecture choisie pour ce projet est basée sur le modèle MVC (Modèle-Vue-Contrôleur), enrichie d'un module DAO pour la gestion des accès à la base de données. Cette architecture a été adoptée pour les raisons suivantes :

* **Séparation des responsabilités** : Chaque composant a une fonction précise :
* **Modèle (Model**) : Représente la structure des données (ex : Livre, Emprunt, Utilisateur) et contient la logique métier.
* **Vue (View)** : Interface graphique développée avec JavaFX et SceneBuilder. Elle est décrite par des fichiers FXML et stylisée avec CSS.
* **Contrôleur (Controlle**r) : Gère les interactions entre l’utilisateur et le modèle. Chaque vue dispose de son propre contrôleur.
* **DAO (Data Access Object)** : Centralise l’accès aux données. Chaque entité dispose de sa propre classe DAO pour effectuer des opérations CRUD (Create, Read, Update, Delete).
* **Gestion des transactions** : Les DAO utilisent des transactions pour assurer la cohérence des données lors des opérations complexes (ex : emprunt d’un livre, retour d’un livre).
* **Scalabilité** :
* MVC permet d’ajouter facilement de nouvelles fonctionnalités ou modules sans affecter le reste de l’application.
* DAO centralise toutes les requêtes SQL, rendant la base de données modifiable sans toucher à la logique métier.

**Exemple de structure MVC + DAO** :

model/Book.java : Décrit la structure d’un livre (id, titre, auteur, ISBN).

view/BookView.fxml : Interface pour visualiser et gérer les livres.

controller/BookController.java : Gère les actions (ajout, modification, suppression de livres).

dao/BookDAO.java : Contient les requêtes SQL pour insérer, mettre à jour, supprimer et récupérer des livres.

Cette organisation garantit une application modulaire, testable et maintenable.

# **CONCEPTION UML**

La conception UML est une étape cruciale pour structurer les interactions entre les composants de l’application. Voici les principaux diagrammes utilisés :

* **Diagramme de classes** :
* Représente les entités principales du système : Livre, Emprunt, Utilisateur, Notification.
* Montre les relations d’héritage entre les classes (ex : Librarian et Administrator héritent d’User).
* Indique les associations entre les classes (ex : Emprunt est lié à Livre et à Utilisateur).

**Exemple** :

Classe Book : id\_book, title, author, isbn, publication\_year, copies (List of Copy).

Classe Loan : id\_loan, date\_loan, exp\_return\_date, act\_return\_date, borrower (FK), copy (FK).

Classe User : id\_user, name, password, role.

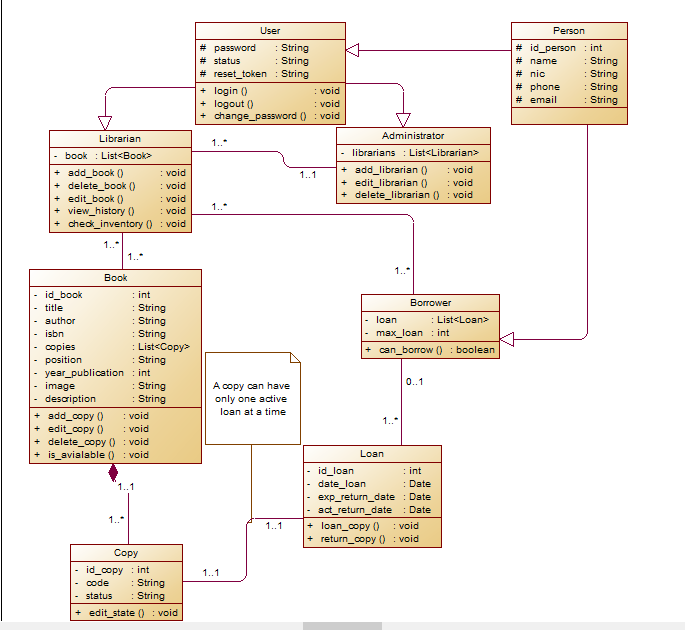


Figure 1 : Diagramme de classe

* **Diagramme de cas d’utilisation** :
  + Met en évidence les interactions entre les acteurs (Librarian, Administrator) et le système.
  + Cas d’utilisation principaux :
* Ajouter un livre.
* Emprunter un livre.
* Retourner un livre.
* Consulter l’historique des emprunts.
* Gérer les utilisateurs (ajout, modification, suppression).
* Décrivent le flux des actions pour des cas d’utilisation spécifiques.

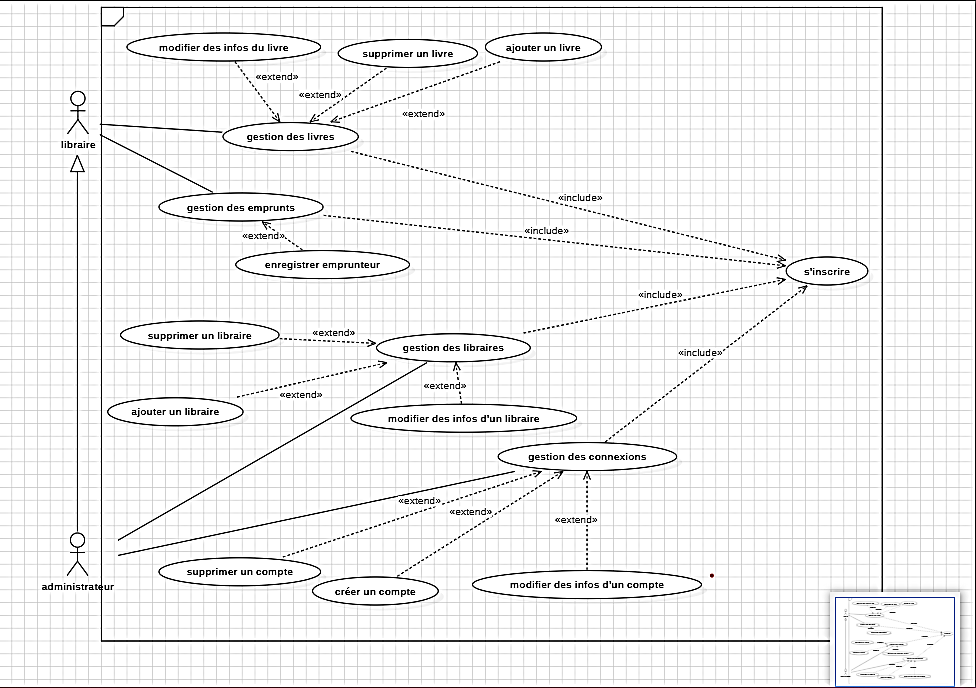


Figure 2 : Diagramme de cas d'utilisation 1

**Exemple** : Séquence pour le retour d’un livre :

1. L’utilisateur sélectionne un emprunt actif.

2. Le contrôleur appelle LoanDAO.returnLoan().

3. LoanDAO met à jour la date de retour et le statut du livre.

4. Le contrôleur notifie l’utilisateur du succès de l’opération.

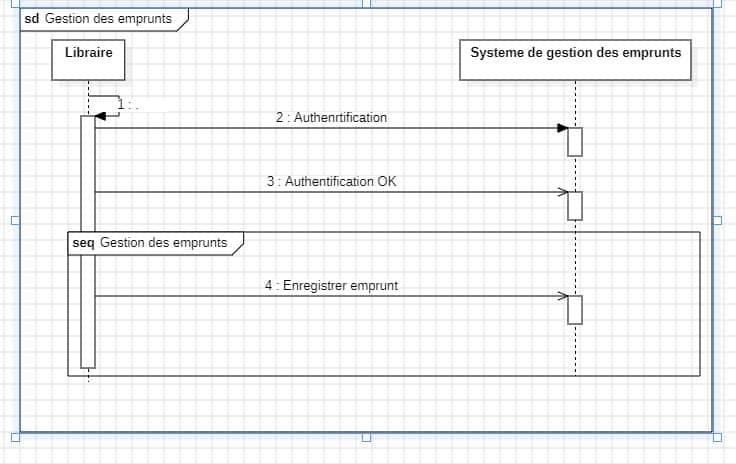


Figure 3 : Diagramme de séquence 1

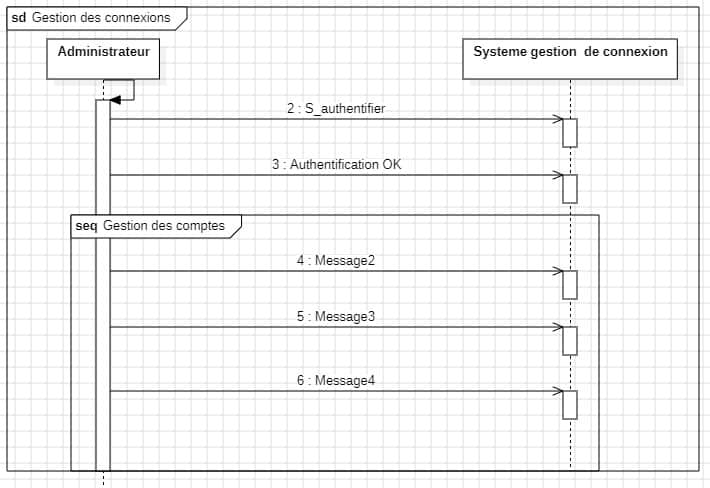


Figure 4 : Diagramme de séquence 2

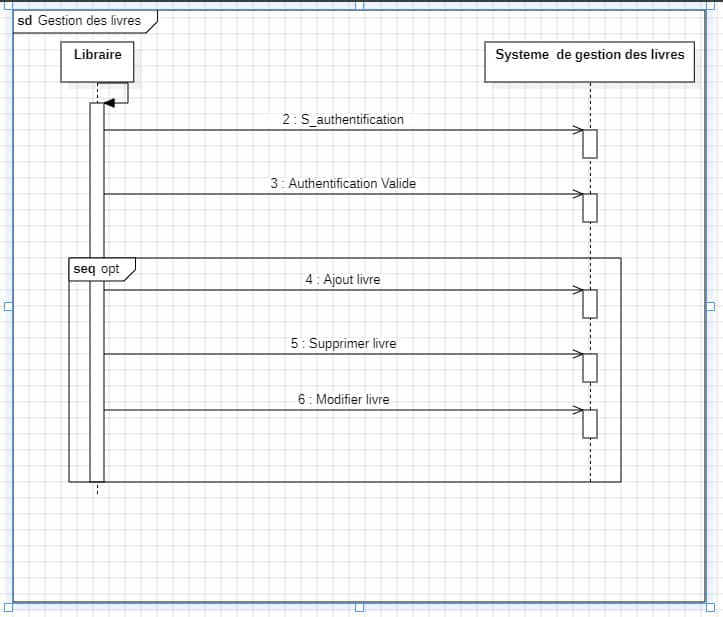


Figure 5 : Diagramme de séquence 3

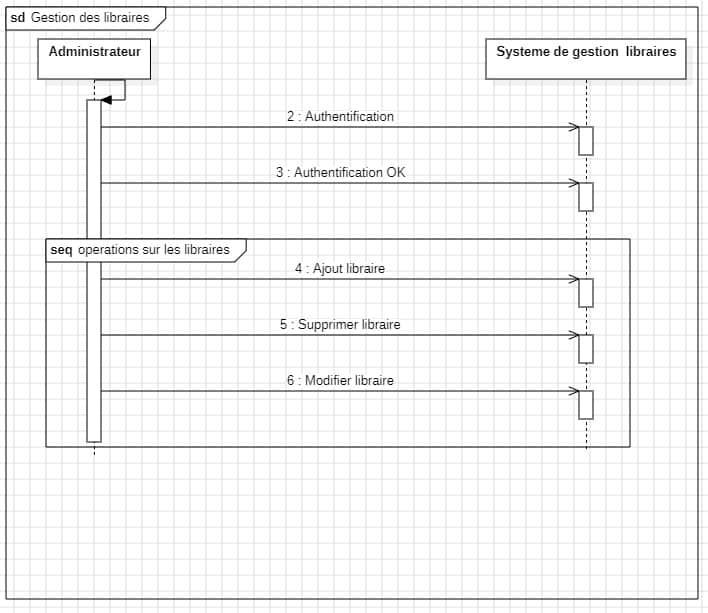


Figure 6 : Diagramme de séquence 4

Ces diagrammes permettent d’anticiper les interactions entre les différentes composantes et de structurer le code en conséquence.

# **STRUCTURE DU PROJET**

Le projet est structuré en plusieurs packages pour assurer une organisation claire des fichiers et faciliter la maintenance. Voici la structure complète :

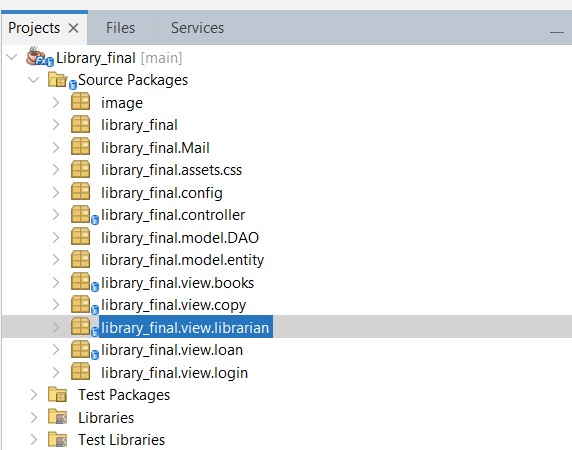


Figure 7 : Structure du projet 1

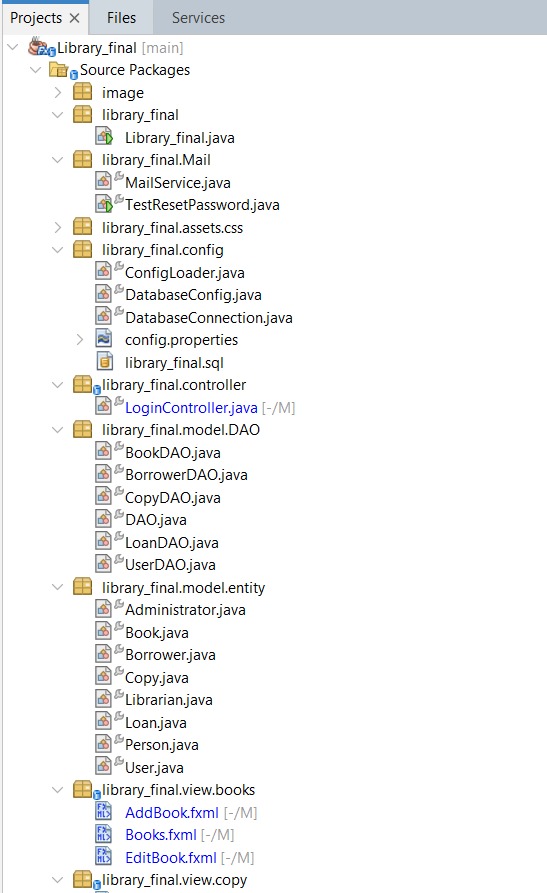


Figure 8 : Diagramme du projet 2

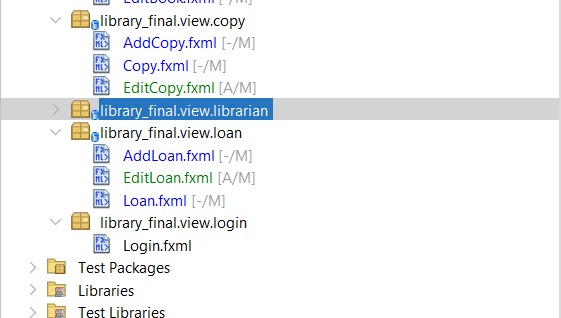


Figure 9 : Structure du projet 3

Cette organisation suit le modèle MVC et assure une séparation claire entre la logique métier, la gestion des données et l’interface utilisateur.

# **DESCRIPTION DES MODULES**

L’application est composée de plusieurs modules, chacun ayant une fonction bien définie pour garantir la modularité, la réutilisabilité et la maintenabilité du code.

* **Image/**: Contient les images utilisées pour l’interface graphique.
* **Library\_final.config/** : Contient les classes permettant la configuration de la base de données et le chargement des paramètres externes.
* *DatabaseConfig.java* : Gère la connexion à la base de données MySQL à l’aide de JDBC.
* *DatabaseConnection.java*: Gère le test de connexion à la base de données.
* *ConfigLoader.java* : Lit les fichiers de configuration (config.properties) pour centraliser les paramètres comme l'URL de la base de données, le nom d'utilisateur et le mot de passe.
* *Library\_final*.*sql* : Contient la requête de création de la base de données.
* **Library\_final.model.entity/** : Représente les entités métiers du système (classes correspondant aux tables de la base de données).
* *Person.java* : Superclasse pour les utilisateurs, bibliothécaires et administrateurs.
* *User.java* : Gère les informations des utilisateurs (mot de passe, rôle, etc.).
* *Librarian.java* : Étend User et ajoute des méthodes spécifiques aux bibliothécaires (gestion des livres).
* *Administrator.java* : Étend User et ajoute des méthodes spécifiques aux administrateurs (gestion des utilisateurs).
* *Book.java* : Représente un livre avec ses attributs (titre, auteur, ISBN).
* *Copy.java* : Représente une copie d’un livre (code, statut).
* *Loan.java* : Gère les emprunts (date de prêt, date de retour, etc.).
* *Borrower.java* : Gère les emprunteurs et le nombre maximal de prêts.
* **Library\_final.model.DAO/** : Ce package centralise toutes les opérations CRUD (Create, Read, Update, Delete, Select).
* *PersonDAO.java* : Gestion des personnes (ajout, modification, suppression).
* *UserDAO.java* : Gestion des utilisateurs et de leurs rôles.
* *BookDAO.java* : Gestion des livres et des copies disponibles.
* *LoanDAO.java* : Gestion des emprunts (création, retour, historique).
* *BorrowerDAO.java* : Gestion des emprunteurs (vérification des emprunts en cours).
* **Library\_final.controller/** : Contient les classes qui gèrent les interactions entre l’utilisateur et le modèle (gestion des événements, validation des entrées).
* *LoginController.java* : Contrôle l'authentification des utilisateurs.
* *DashboardController.java* : Gère la navigation principale de l'application.
* *BookController.java* : Gère les opérations liées aux livres (ajout, suppression, édition).
* *UserController*.*java* : Gère les utilisateurs (ajout, suppression, édition).
* *LoanController.java* : Gère les emprunts (prêts, retours, consultation de l’historique).
* **library\_final.view.login/** : Contient les fichiers FXML définissant l'interface utilisateur.
* *Login.fxml* : Interface de connexion.
* **library\_final.view.loan/ :** 
  + *AddLoan.fxml* : Gère l’ajout des emprunts
  + *EditLoan.fxml* : Gère la modification des emprunts
  + *DeleteLoan.fxml* : Gère la suppression des emprunts
* **library\_final.view.librarian/ :** 
  + *AddLibrarian.fxml* : Gère l’ajout des bibliothécaire
  + *EditLibrarian.fxml* : Gère la modification des bibliothécaire
  + *DeleteLibrarian.fxml* : Gère la suppression des bibliothécaire
* **library\_final.view.copy/ :** 
  + *AddCopy.fxml* : Gère l’ajout des copies de livres
  + *EditCopy.fxml* : Gère la modification des copies de livres
  + *DeleteCopy.fxml* : Gère la suppression des copies de livres ;
* **Library\_final.Mail/** : Module dédié à l’envoi d’e-mails de notification ou de réinitialisation de mot de passe.
* *MailService.java* : Service pour envoyer des e-mails.
* *TestResetPassword.java* : Classe de test pour la fonctionnalité de réinitialisation de mot de passe.

Cette structure garantit une organisation modulaire, où chaque package a une responsabilité bien définie, permettant ainsi une gestion efficace des fonctionnalités et des données.

# **CONFIGURATION DE LA BASE DE DONNEES**

La base de données est structurée pour gérer efficacement les informations relatives aux livres, emprunts, utilisateurs et notifications. Elle est conçue pour minimiser la redondance et garantir l’intégrité des données à travers des relations bien définies.

**SGBD : MySQL 8.0**

* Système de gestion de base de données relationnel utilisé pour stocker et organiser les informations.
* Supporte les transactions ACID, les clés étrangères et les jointures complexes.
* Gestion des contraintes d’intégrité :
* Utilisation de clés primaires (id\_book, id\_user, id\_loan) pour identifier de manière unique chaque enregistrement.
* Clés étrangères (id\_user dans Loan, id\_book dans Copy) pour garantir l’intégrité référentielle.

**Extrait SQL représentatif** :

CREATE DATABASE Library;

USE Library;

CREATE TABLE IF NOT EXISTS Person(

id\_person INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

nic VARCHAR(50) NOT NULL UNIQUE,

phone VARCHAR(20) NOT NULL,

email VARCHAR(100)

)ENGINE=InnoDB;

CREATE TABLE IF NOT EXISTS Borrower(

id\_borrower INT UNSIGNED PRIMARY KEY,

max\_loan INT DEFAULT 3,

FOREIGN KEY (id\_borrower) REFERENCES Person(id\_person) ON DELETE CASCADE

)ENGINE=InnoDB;

CREATE TABLE IF NOT EXISTS `User`(

id\_user INT UNSIGNED PRIMARY KEY ,

password VARCHAR(255) NOT NULL,

status ENUM('Active', 'Inactive') DEFAULT 'Active',

reset\_token VARCHAR(255),

role ENUM('Administrator', 'Librarian') NOT NULL,

FOREIGN KEY (id\_user) REFERENCES Person(id\_person) ON DELETE CASCADE

)ENGINE=InnoDB;

CREATE TABLE IF NOT EXISTS Book(

id\_book INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(150) NOT NULL,

author VARCHAR(100),

isbn VARCHAR(50) UNIQUE NOT NULL,

publication\_year INT(4),

position VARCHAR(100),

image VARCHAR(255),

description TEXT,

copies INT DEFAULT 1,

status ENUM('Active', 'Inactive') NOT NULL DEFAULT 'Active'

)ENGINE=InnoDB;

CREATE TABLE IF NOT EXISTS Copy(

id\_copy INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

code VARCHAR (50) UNIQUE NOT NULL,

status ENUM('Available’, ‘Loaned', 'Lost', 'Inactive') NOT NULL DEFAULT 'Available',

id\_book INT UNSIGNED NOT NULL,

FOREIGN KEY (id\_book) REFERENCES Book(id\_book) ON DELETE CASCADE

)ENGINE=InnoDB;

CREATE TABLE IF NOT EXISTS Loan(

id\_loan INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

id\_copy INT UNSIGNED NOT NULL,

id\_borrower INT UNSIGNED NOT NULL,

id\_librarian INT UNSIGNED NOT NULL,

date\_loan DATE NOT NULL,

exp\_return\_date DATE NOT NULL,

act\_return\_date DATE DEFAULT NULL,

FOREIGN KEY (id\_copy) REFERENCES Copy(id\_copy) ON DELETE SET NULL,

FOREIGN KEY (id\_borrower) REFERENCES Borrower(id\_borrower) ON DELETE SET NULL,

FOREIGN KEY (id\_librarian) REFERENCES User(id\_user) ON DELETE SET NULL

) ENGINE=InnoDB;