Rapport de Projet : Système de Gestion d’une Bibliothèque

# Contexte et Objectif

Le but de ce projet est de développer une application Java permettant de gérer les opérations courantes d’une petite bibliothèque. L’application permettra d’enregistrer des livres, de gérer les emprunts et retours, et de consulter l’inventaire des ouvrages.

# Exigences Techniques

* **Langage :** Java (version 8 ou supérieure).
* **Méthodologie :** Approche orientée objet.
* **Outils :** Un IDE comme Netbeans, Eclipse ou IntelliJ IDEA.
* **Documentation :** Chaque groupe doit fournir un rapport expliquant l’architecture, les choix techniques, et un manuel d’utilisation.
* **Versioning (Optionnel) :** Utiliser Git pour le suivi des modifications et la collaboration.

# Presentation Generale

* + **Nom de l’application :** Library Manager
  + **Technologies :** java, javaFX , SQLite , JDBC
  + **Architecture :** MVC (Modele-vue-controller )

# Phase de conception

## Analyse des besions

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

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

Interface Utilisateur Console

* 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.
* Persistance des Données (Optionnel pour les débutants)

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

## Conception de l’architecture MVC

## 

* Controllers/ : gestions des interactions utilisateurs
* Models/: entite metier ( Books , User , Loan , Category , etc)
* Views/: interfaces Fxml
* Databases/: operations CRUD et gestion des tables
* Utils/ : Classes Utilitaires ( ex : Currentuser pour la gestion des sessions)

## Modelisation de la base de donnees

La base de donnee a ette initialement modelise a partir des classes du diagrammes de classes mais a été mis a jour plutard pour satisfaire les besions de developpement.

1. table books (livres)

* book\_id: identifiant unique (clé primaire, auto-incrémenté)
* title: titre du livre (obligatoire)
* author: auteur (obligatoire)
* isbn: numéro isbn (unique)
* description: description du livre
* category\_id: référence à la catégorie (clé étrangère)
* published\_year: année de publication
* copies\_total: nombre total d'exemplaires (>0)
* copies\_available: nombre d'exemplaires disponibles (≥0)
* created\_at: date de création
* image\_path: chemin de l'image

1. table category (catégories

* category\_id: identifiant unique (clé primaire, auto-incrémenté)
* name: nom de la catégorie

1. table loans (emprunts)

* loan\_id: identifiant unique (clé primaire, auto-incrémenté)
* book\_id: référence au livre (clé étrangère)
* user\_id: référence à l'utilisateur (clé étrangère)
* borrowed\_at: date d'emprunt
* due\_at: date de retour prévue
* returned\_at: date de retour effective
* status: état de l'emprunt ('ongoing', 'returned', 'overdue')
* number\_of\_book: nombre de livres empruntés

1. table users (utilisateurs)

* user\_id: identifiant unique (clé primaire, auto-incrémenté)
* username: nom d'utilisateur (unique)
* password\_hash: mot de passe hashé
* role: rôle ('admin' ou 'member')
* created\_at: date de création
* phone: numéro de téléphone
* birthdate: date de naissance
* gender: genre ('male' ou 'female')
* address: adresse
* email: email (unique)

# Phase d’implementation

## Developpement des modeles de donnees

Les modeles de donnee ont été ressortir a partir du diagramme de classe uml ainsi on a les modeles suivants :

- Books.java

- Attributes: book\_id, title, author, isbn, category\_id, published\_year,

copies\_total, copies\_available, image\_path, description

- User.java

- Attributes: user\_id, fullName, email, password, role, phone, birthdate,

address, gender

- Loan.java

- Attributes: loan\_id, book\_id, user\_id, borrowed\_at, due\_at, returned\_at,

status, book\_name, user\_name, numberOfBook

- Category.java

- Attributes: category\_id, name

- Notification.java

- Attributes: message, date

- Model.java

- Singleton class managing ViewFactory

- Historique.java

- Planned for future implementation

## Creation des interfaces utilisateurs

Les interfaces ont été cree a l’aide scenebuilder qui est logiciel qui permet de créer des vues fxml de manieres gratuites ainsi on a les fichiers fxml suivant :

Authentication:

- Login.fxml : Login interface

- Registration.fxml : New user registration

Main Interfaces:

- Dashboard.fxml : Main application dashboard

- Sidebar.fxml : Navigation sidebar component

- Dash.fxml : Dashboard content

Resource Management:

- Books.fxml : Book management interface

- BookCard.fxml : Individual book display component

- BookDetails.fxml : Detailed book information

- EditBook.fxml : Book editing interface

- Users.fxml : User management interface

- Borrows.fxml : Loan management interface

- BorrowsCard.fxml : Individual loan display component

- ConfirmBorrow.fxml : Loan confirmation interface

Support:

- Help.fxml : Help and contact information

## Implementations des controlleurs

Pour chaque vue fxml ci-dessus est associe un controlleur du meeme nom pour gerer la communication entre la vue et la base de donnee

## Integration de la base de donnees

* Interface Repository<T> définissant les opérations CRUD standard
* Classes dédiées par entité (BooksTable, UserTable, etc.)
* Requêtes préparées pour la sécurité
* Transactions pour maintenir l'intégrité des données

# Difficultes Rencontres

## Techniques

- Conflits lors des push :

Modifications concurrentes sur les mêmes fichiers (ex : DatabaseUtils.java, contrôleurs) généraient des conflits fréquents.

Solutions :

- Adoption d'une stratégie de feature branches

- Revue de code via pull requests

- Usage de git rebase pour un historique linéaire propre

- Répartition des tâches :

La distribution initiale du travail manquait de clarté.

Solutions :

- Une liste de tâches numérotée était envoyée au début de chaque sprint ; chaque membre choisissait un numéro

- Communication via WhatsApp pour les tâches et coordination

- Réunions quotidiennes ou ponctuelles sur Google Meet ou dans des groupes WhatsApp dédiés

Implémentation du Modèle MVC

- Confusion des responsabilités :

Logique métier parfois incluse dans les contrôleurs

Solutions :

- Refactoring pour clarifier le rôle des classes (modèle vs contrôleur)

- Centralisation des informations globales dans CurrentUser

- Utilisation de structures claires pour séparer l'accès aux données et la logique métier

- Dépendances entre contrôleurs :

Partage de données entre les vues (ex. état de connexion, sélection d'éléments) Solutions :

- Dépendances gérées via des méthodes static ou accès à des attributs static

- CurrentUser jouait le rôle de session globale

## Fonctionnelles

Communication entre les Composants de l'Interface

- Mises à jour non synchronisées :

Les modifications dans une vue n'étaient pas reflétées ailleurs.

Solutions :

- Utilisation de la classe utilitaire CurrentUser pour stocker l'utilisateur actif

- Méthodes et attributs static dans les contrôleurs pour partager l'état ou accéder aux vues

- Rafraîchissements explicites déclenchés selon les besoins dans les vues concernées

# Perspectives d’amelioration

- Tests automatisés : intégration de tests unitaires (JUnit) et d'intégration (TestFX)

- Amélioration du workflow Git : adoption de GitFlow pour uniformiser le cycle de développement

- Surveillance et logs : ajout de journaux d'activité avec Log4j pour diagnostiquer les anomalies

- Nouvelles fonctionnalités envisagées :

- Réservations de livres

- Notifications par e-mail

- Interface mobile

- Export de données

- Gestion des amendes

# Conclusion

Le projet Library Manager a permis de construire une application robuste, évolutive et orientée utilisateur. Les difficultés rencontrées ont été surmontées grâce à une organisation rigoureuse, une utilisation pertinente de l'architecture MVC, et des outils collaboratifs simples mais efficaces (Git, WhatsApp, Google Meet). Le système reste extensible et constitue une base solide pour de futurs développements en équipe.

# 