Projet Bibliotheque

RAPPORT POO

Sous la Supervision du Dr NOULAPEU

LISTE DES MEMBRES

NYEMB LOUIS PERIER 22G01033

TAWE KUITCHE MARIUS PEGUY 22G00383

PEWELI ASSALA ANAYICE 22G00515

ASSOMO MARIELLE CASSANDRA DE PARIS 22G00016

DIWONGI SONGUE ARIAN XAVIER 24G0182

TCHEMI EUGÈNE YVAN ULRICK 22G00385

MPACKO DARREN NGOLE 24G0115

KOUEMKONG NKAMGA FRANCK NELSON 22G00203

TIDO KEUBOU LEAL BRAËL 22G00393

BELLE MANDENGUE MAEVA ERICKA 24G01080

SOMMAIRE

INTRODUCTION

Ce rapport présente le développement d'une application Java pour la gestion d'une bibliothèque, réalisée dans le cadre du projet de programmation orientée objet. L'objectif principal était de créer un système permettant de gérer efficacement l'inventaire des livres, les emprunts et les retours au sein d'une petite bibliothèque.

Le projet initial prévoyait une interface console, mais notre équipe a choisi de développer une interface graphique afin d'améliorer l'expérience utilisateur et de mettre en pratique les concepts avancés de programmation Java. Cette décision a été motivée par la volonté de proposer une solution plus intuitive et conviviale pour les utilisateurs finaux.

L'application permet d'effectuer toutes les opérations essentielles à la gestion d'une bibliothèque, notamment l'ajout, la modification et la suppression de livres, l'enregistrement des emprunts et des retours, ainsi que la consultation de l'inventaire et de l'historique des transactions.

ANALYSE DES BESOINS

OBJECTIFS DU PROJET

- Développer une application de gestion de bibliothèque complète et fonctionnelle.
- Appliquer les principes de la programmation orientée objet
- Implémenter une interface graphique intuitive et ergonomique
- Assurer la persistance des données entre les sessions

EXIGENCES FONCTIONNELLES

1. Gestion des Livres

- Ajouter un nouveau livre avec ses informations complètes (titre, auteur, ISBN, année de publication)
- Modifier les informations d'un livre existant
- Supprimer un livre de l'inventaire
- Rechercher un livre selon différents critères (titre, auteur, ISBN)
- Afficher l'ensemble des livres disponibles

2. Gestion des EmpruntsConception et Architecture

- Enregistrer l'emprunt d'un livre par un utilisateur
- Enregistrer le retour d'un livre
- Consulter les livres actuellement empruntés
- Visualiser l'historique des emprunts par livre et par utilisateur
- Gérer les dates d'échéance et les retards

3. Gestion des Utilisateurs

- Enregistrer un nouvel utilisateur
- Modifier les informations d'un utilisateur
- Consulter la liste des utilisateurs
- Visualiser les emprunts en cours par utilisateur

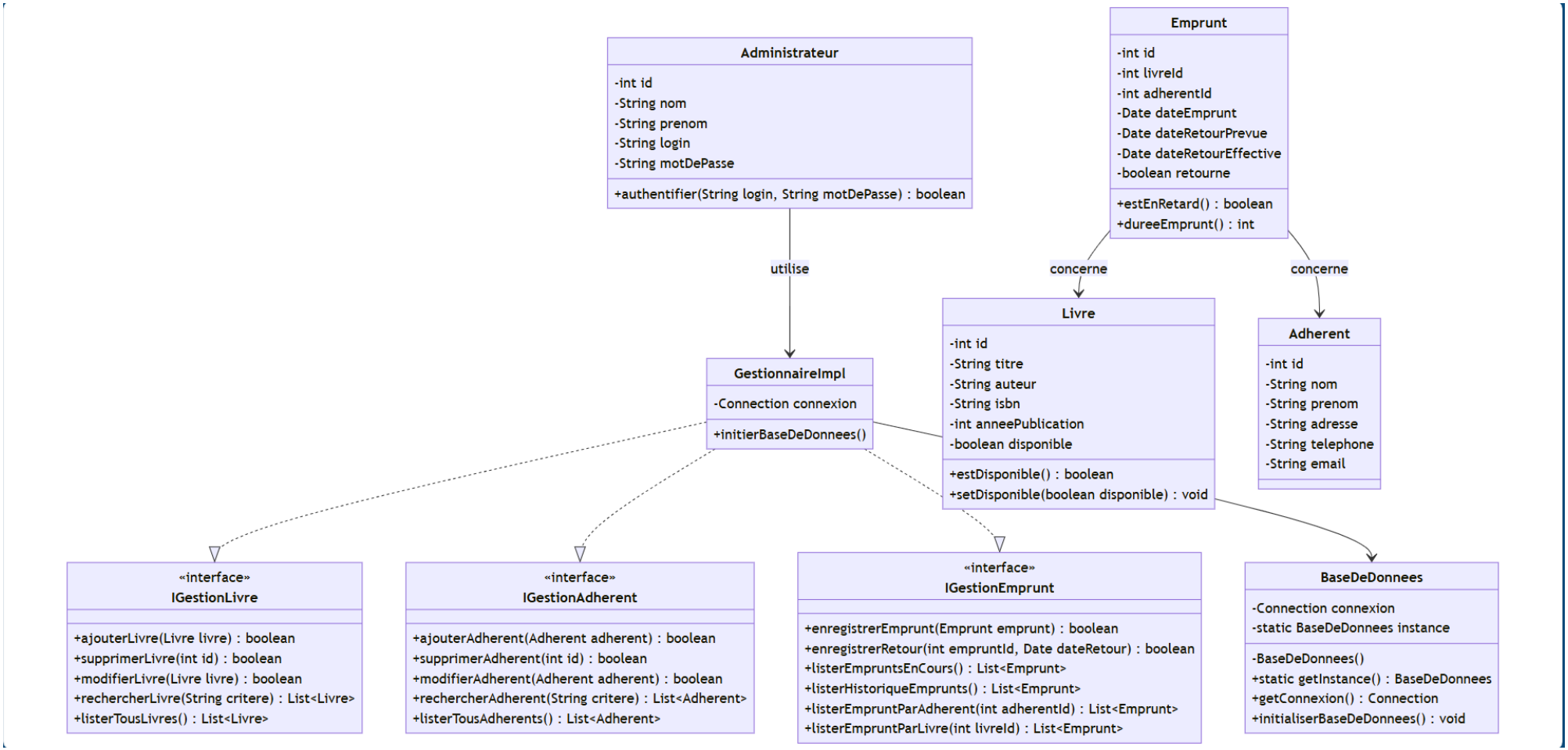
EXIGENCES NON-FONCTIONNELLES

- Interface utilisateur intuitive et réactive
- Temps de réponse rapide pour les opérations courantes
- Stockage persistant des données
- Gestion appropriée des erreurs et validation des entrées
- Documentation complète du code et de l'utilisation

CONCEPTION ET ARCHITECTURE

DIAGRAMME DE CLASSES

Notre conception s'articule autour de plusieurs classes principales qui modélisent les entités du domaine et leurs relations. Le diagramme de classes ci-dessous illustre l'architecture générale du système.


Administrateur
-int id
-String nom
-String prenom
-String login
-String motDePasse
+authentifier(String login, String motDePasse) : boolean
Emprunt
-int id
-int livreId
-int adherentId
-Date dateEmprunt
-Date dateRetourPrevue
-Date dateRetourEffective
-boolean retourne
+estEnRetard() : boolean
+dureeEmprunt() : int
utilise
concerne
concerne
GestionnaireImpl
-Connection connexion
+initierBaseDeDonnees()
Livre
-int id
-String titre
-String auteur
-String isbn
-int anneePublication
-boolean disponible
+estDisponible() : boolean
+setDisponible(boolean disponible) : void
Adherent
-int id
-String nom
-String prenom
-String adresse
-String telephone
-String email
«interface»
IGestionLivre
+ajouterLivre(Livre livre) : boolean
+supprimerLivre(int id) : boolean
+modifierLivre(Livre livre) : boolean
+rechercherLivre(String critere) : List<Livre>
+listerTousLivres() : List<Livre>
«interface»
IGestionAdherent
+ajouterAdherent(Adherent adherent) : boolean
+supprimerAdherent(int id) : boolean
+modifierAdherent(Adherent adherent) : boolean
+rechercherAdherent(String critere) : List<Adherent>
+listerTousAdherents() : List<Adherent>
«interface»
IGestionEmprunt
+enregistrerEmprunt(Emprunt emprunt) : boolean
+enregistrerRetour(int empruntId, Date dateRetour) : boolean
+listerEmpruntsEnCours() : List<Emprunt>
+listerHistoriqueEmprunts() : List<Emprunt>
+listerEmpruntParAdherent(int adherentId) : List<Emprunt>
+listerEmpruntParLivre(int livreId) : List<Emprunt>
BaseDeDonnees
-Connection connexion
-static BaseDeDonnees instance
-BaseDeDonnees()
+static getInstance() : BaseDeDonnees
+getConnexion() : Connection
+initialiserBaseDeDonnees() : void

Les principales classes du système sont:

1. **Livre**: Représente un ouvrage avec ses attributs (titre, auteur, ISBN, année de publication, statut, etc.)
2. **Utilisateur**: Modélise un emprunteur avec ses informations personnelles
3. **Emprunt**: Associe un livre, un utilisateur et des dates (emprunt, retour prévu, retour effectif)
4. **Bibliothèque**: Classe centrale qui gère l'ensemble des fonctionnalités métier
5. **DAO (Data Access Objects)**: Classes responsables de l'accès aux données dans la base SQLite
6. **Services**: Classes qui implémentent la logique métier en s'appuyant sur les DAOs

CAS D'UTILISATION

Le diagramme de cas d'utilisation ci-dessous présente les principales fonctionnalités offertes par le système et les interactions possibles avec les utilisateurs.

(A la suite)

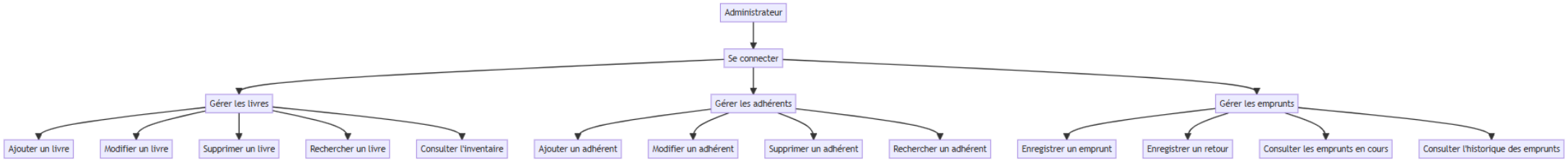
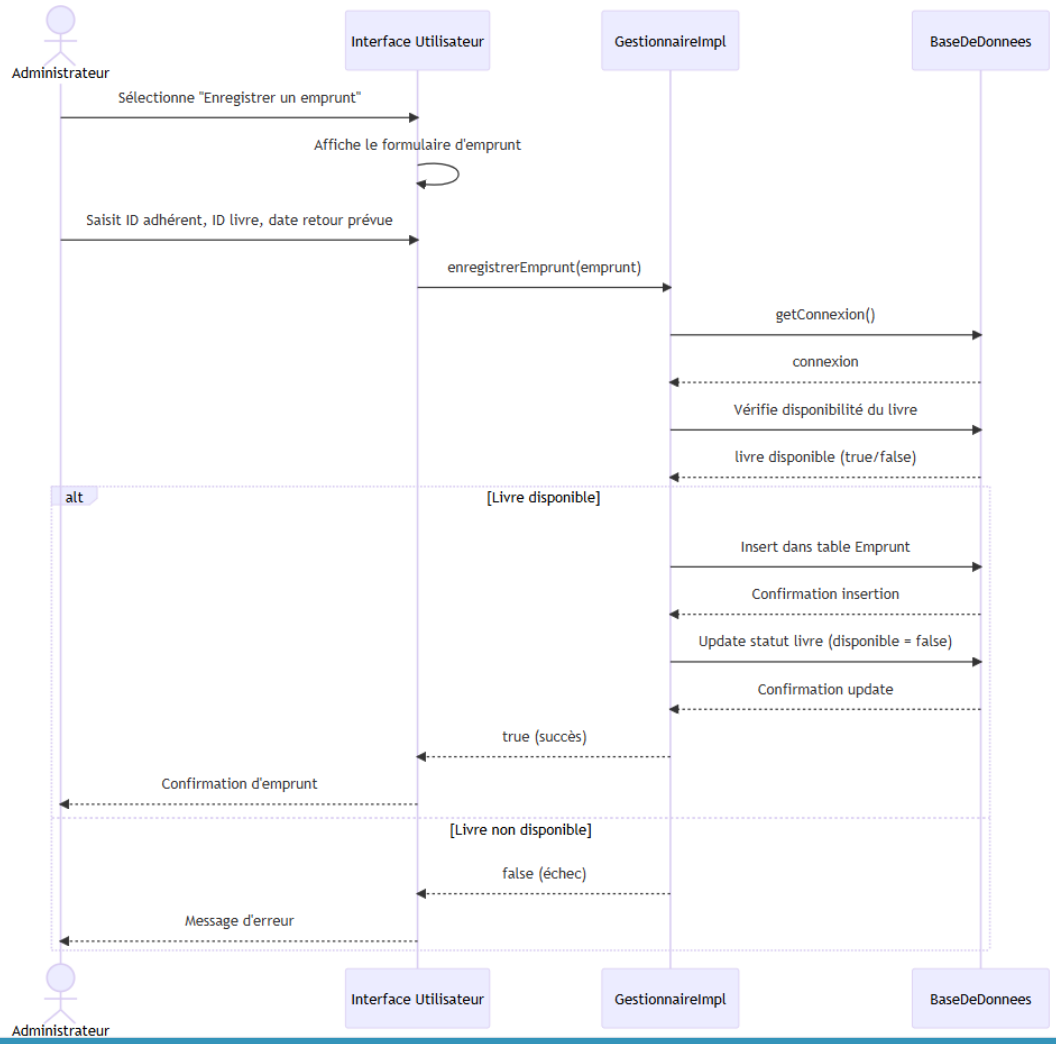

Administrateur
Se connecter
Gérer les livres
Gérer les adhérents
Gérer les emprunts
Ajouter un livre
Modifier un livre
Supprimer un livre
Rechercher un livre
Consulter l'inventaire
Ajouter un adhérent
Modifier un adhérent
Supprimer un adhérent
Rechercher un adhérent
Enregistrer un emprunt
Enregistrer un retour
Consulter les emprunts en cours
Consulter l'historique des emprunts

DIAGRAMME DE SEQUENCE

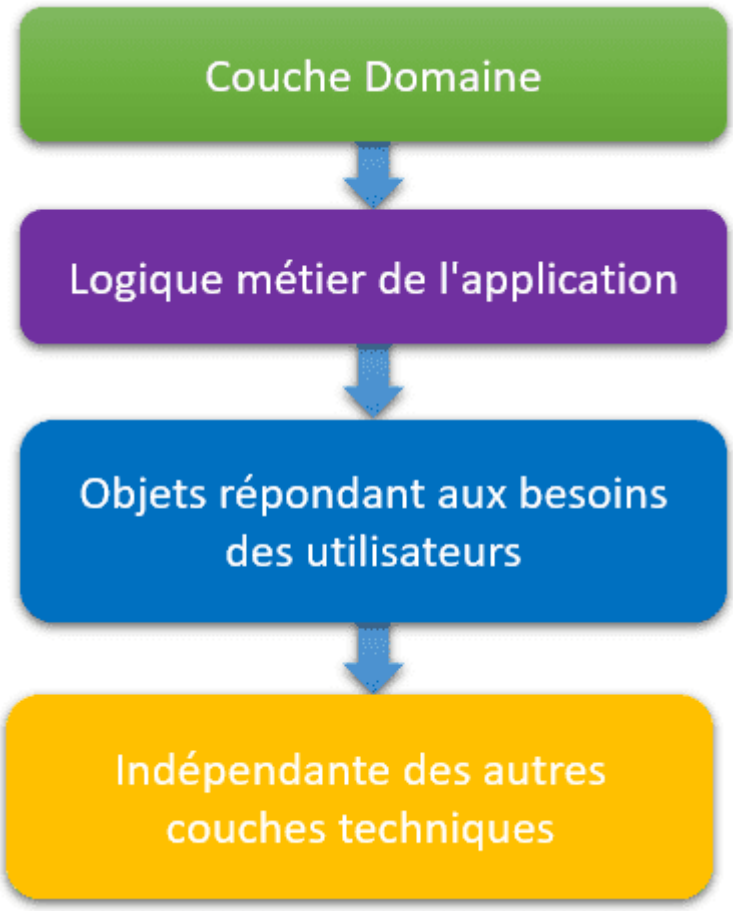


ARCHITECTURE EN COUCHES

Nous avons adopté une architecture en couches pour structurer notre application, ce qui permet une séparation claire des responsabilités:

- **Couche Présentation (UI)**: Les interfaces graphiques JavaFX et leurs contrôleurs
- **Couche Service**: Implémente la logique métier et coordonne les opérations
- **Couche DAO**: Assure l'accès aux données persistantes dans la base SQLite
- **Couche Métier**: Les classes qui modélisent le domaine de l'application

Cette architecture facilite la maintenance et l'évolution de l'application, tout en permettant une meilleure réutilisation du code.



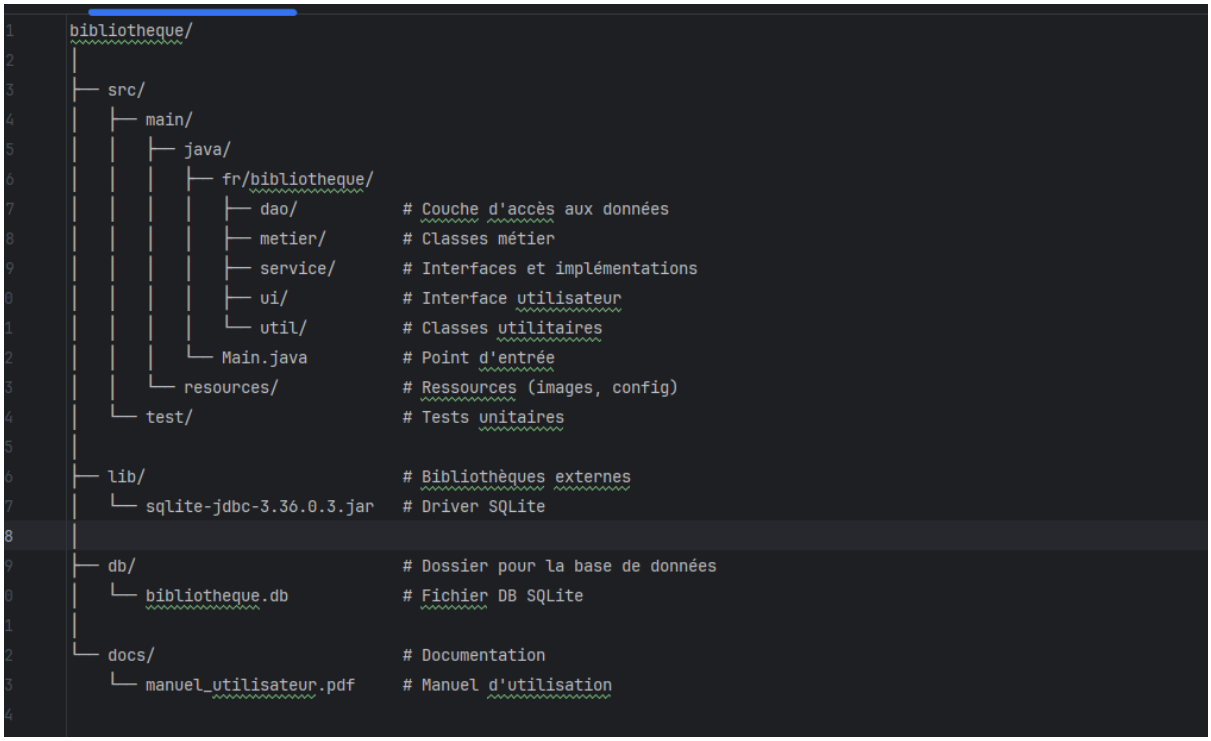
IMPLEMENTATION

TECHNOLOGIES UTILISEES

- **Langage**: Java 11
- **Interface Graphique**: JavaFX
- **Persistance des Données**: Base de données SQLite
- **IDE**: Eclipse
- **Gestion de Version**: Git/GitHub

STRUCTURE DU PROJET

L'organisation du code source suit une architecture en couches claire avec séparation des responsabilités. La structure de packages reflète cette approche:



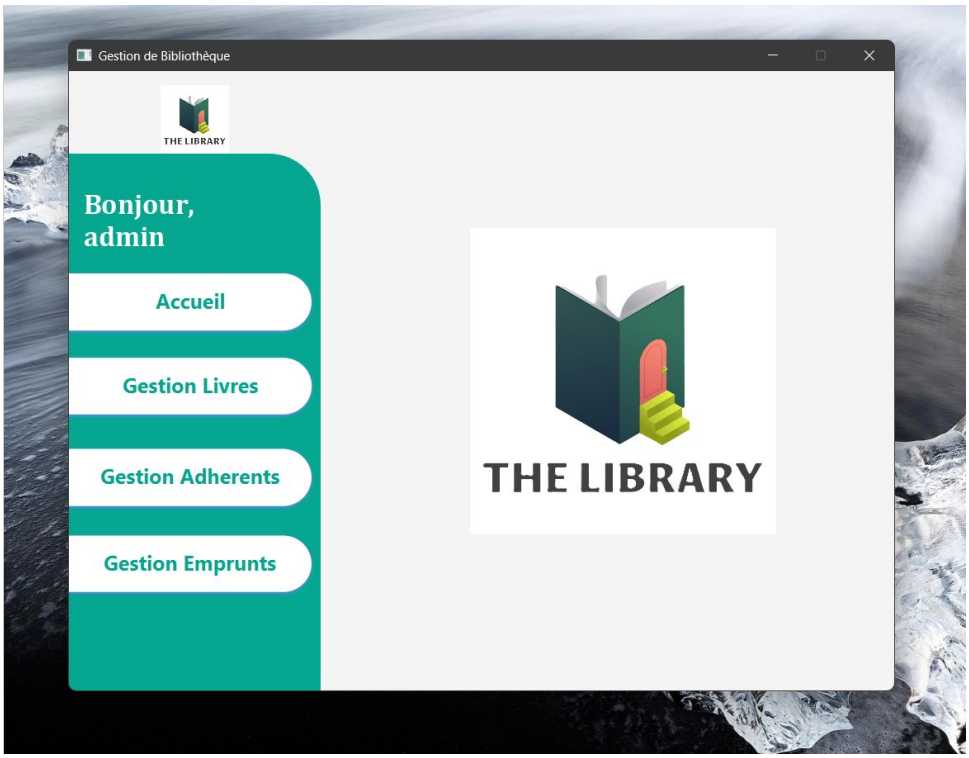
Cette organisation facilite la maintenance du code et permet une séparation claire entre l'interface utilisateur, la logique métier et l'accès aux données.

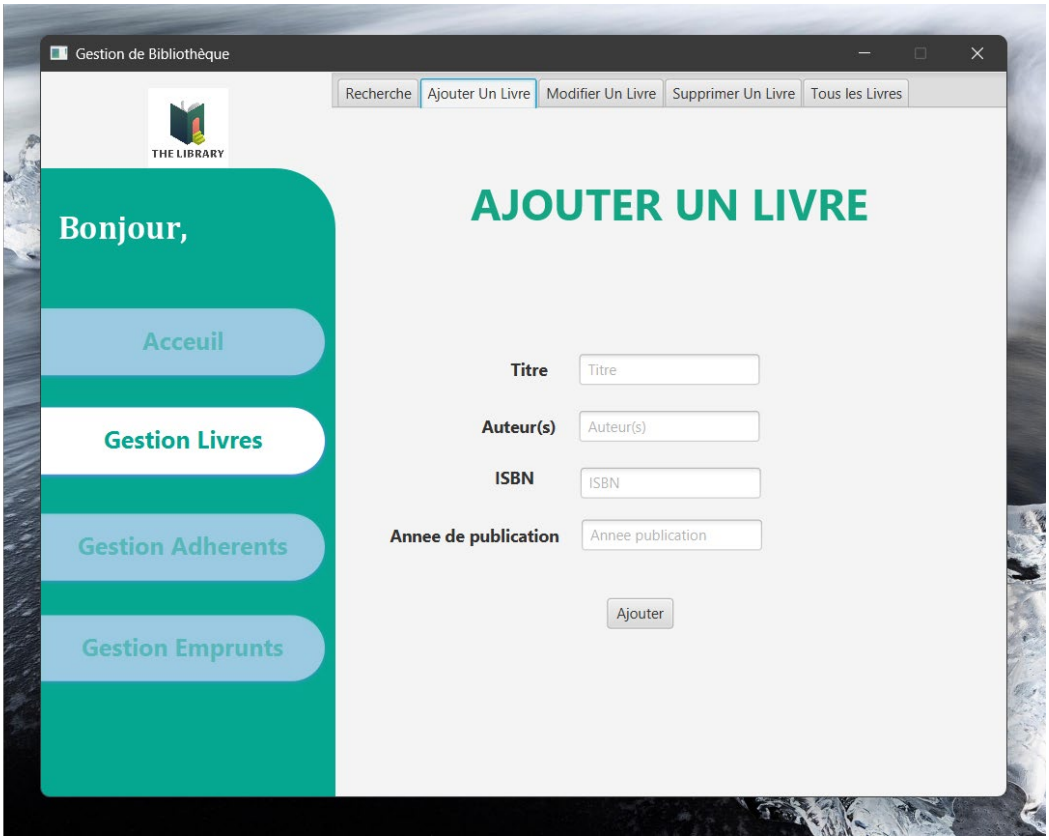
Contrairement à la consigne initiale qui préconisait une interface console, nous avons développé une interface graphique complète utilisant JavaFX. Ce choix nous a permis de créer une application plus conviviale et intuitive pour les utilisateurs finaux.

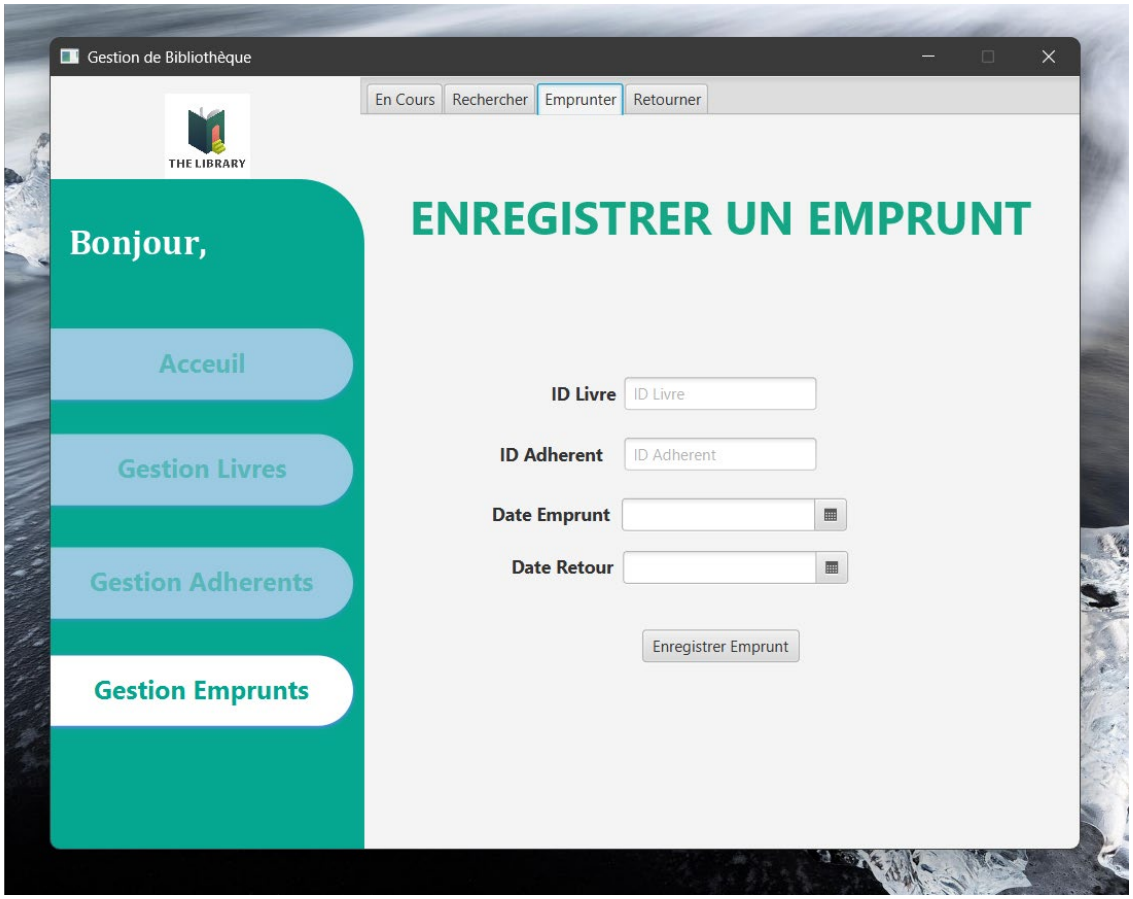
L'interface graphique est structurée autour d'un système d'onglets correspondant aux principales fonctionnalités du système:

1. **Gestion des Livres**: Interface pour ajouter, modifier, supprimer et rechercher des livres
2. **Gestion des Emprunts**: Interface pour enregistrer les emprunts et les retours
3. **Gestion des Utilisateurs**: Interface pour gérer les informations des emprunteurs
4. **Tableau de Bord**: Vue d'ensemble des statistiques de la bibliothèque

Les interfaces ont été conçues avec JavaFX et FXML, ce qui permet une séparation claire entre la présentation (fichiers FXML) et la logique de contrôle (classes Controller). Cette approche facilite la maintenance et l'évolution de l'interface utilisateur.


Gestion de Bibliothèque
THE LIBRARY
Bonjour,
admin
Accueil
Gestion Livres
Gestion Adherents
Gestion Emprunts
THE LIBRARY


Gestion de Bibliothèque
Recherche
Ajouter Un Livre
Modifier Un Livre
Supprimer Un Livre
Tous les Livres
THE LIBRARY
Bonjour,
Acceuil
Gestion Livres
Gestion Adherents
Gestion Emprunts
AJOUTER UN LIVRE
Titre
Titre
Auteur(s)
Auteur(s)
ISBN
ISBN
Annee de publication
Annee publication
Ajouter



Persistance des Données

Pour assurer la persistance des données entre les sessions d'utilisation, nous avons implémenté un système de stockage basé sur une base de données SQLite. Ce choix a été motivé par la légèreté de SQLite, sa facilité de mise en œuvre et sa robustesse.

La base de données est structurée autour de trois tables principales:

- Livres: Stocke les informations sur les livres
- Utilisateurs: Contient les données des utilisateurs
- Emprunts: Enregistre l'historique des emprunts avec les relations entre livres et utilisateurs

Un fichier de base de données SQLite (bibliotheque.db) est créé au premier lancement de l'application et stocké dans le répertoire db/. La connexion à la base de données est gérée par la classe SQLiteManager qui fournit les méthodes nécessaires pour les opérations CRUD (Create, Read, Update, Delete).

Pour interagir avec la base de données, nous utilisons le driver JDBC SQLite (version 3.36.0.3) qui est inclus dans le répertoire lib/ du projet.

Fonctionnalités Implémentées

Gestion des Livres

La fonctionnalité de gestion des livres permet aux bibliothécaires de gérer efficacement l'inventaire de la bibliothèque:

- **Ajout de Livre**: Un formulaire complet permet de saisir toutes les informations nécessaires (titre, auteur, ISBN, année de publication, genre, description, etc.)
- **Modification**: Les informations d'un livre existant peuvent être mises à jour à tout moment
- **Suppression**: Les livres qui ne font plus partie de la collection peuvent être retirés de l'inventaire
- **Recherche Avancée**: Un système de filtrage permet de rechercher des livres selon différents critères (titre, auteur, disponibilité, etc.)
- **Visualisation**: L'ensemble des livres est affiché dans un tableau avec la possibilité de trier selon différentes colonnes

Gestion des Emprunts

Le module de gestion des emprunts constitue le cœur fonctionnel de l'application:

- **Enregistrement des Emprunts**: Permet d'associer un livre à un utilisateur avec une date d'emprunt et une date de retour prévue
- **Retour de Livre**: Enregistre la date effective de retour et met à jour le statut du livre
- **Prolongation**: Possibilité de prolonger la durée d'un emprunt si nécessaire
- **Gestion des Retards**: Identification automatique des emprunts en retard avec notification visuelle
- **Historique**: Consultation de l'historique complet des emprunts pour chaque livre et chaque utilisateur

Gestion des Utilisateurs

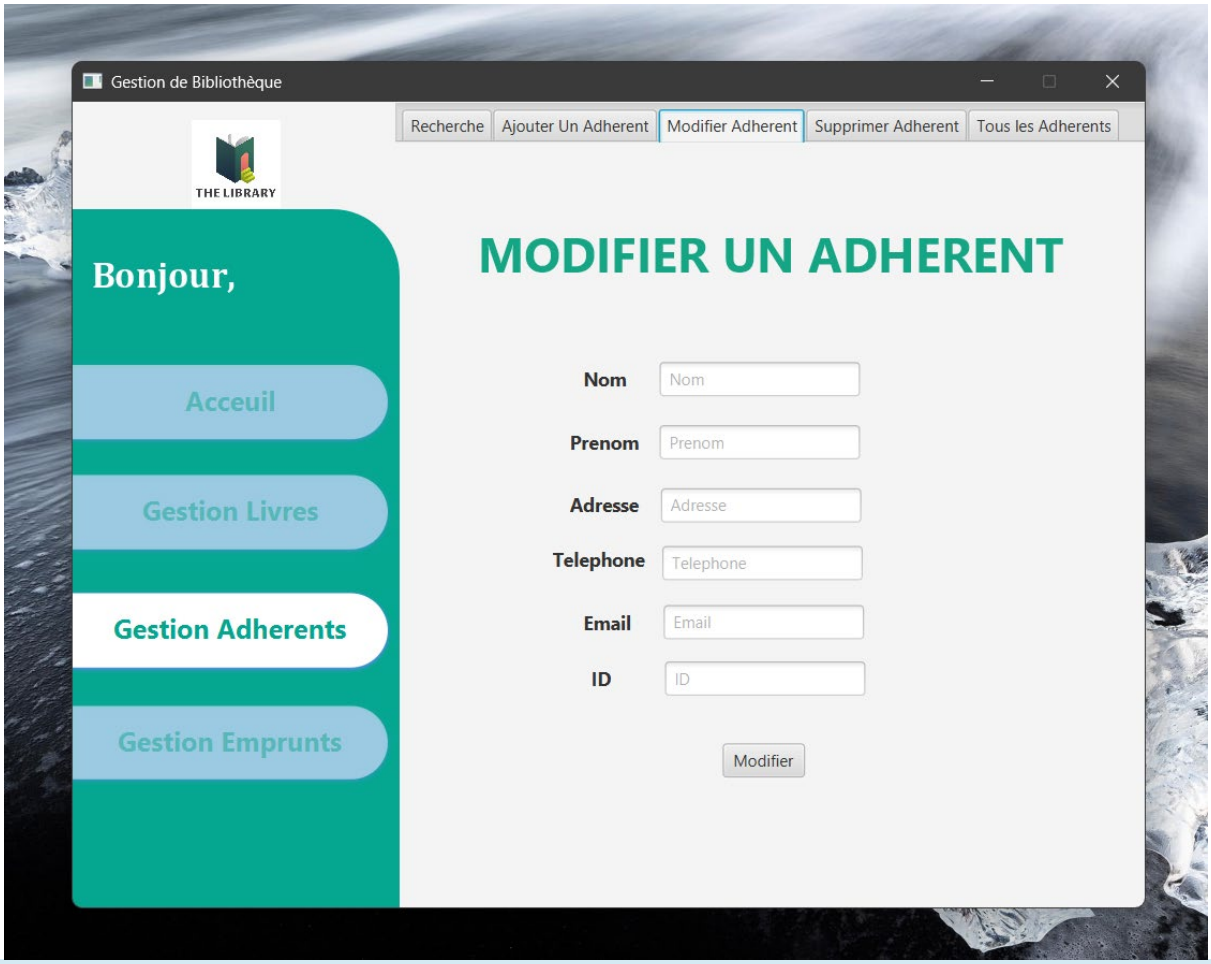
Ce module permet de gérer les informations relatives aux emprunteurs:

- **Inscription**: Enregistrement d'un nouvel utilisateur avec ses coordonnées complètes
- **Mise à Jour**: Modification des informations personnelles d'un utilisateur existant
- **Consultation des Emprunts**: Visualisation des emprunts en cours et de l'historique pour chaque utilisateur

- **Statut**: Gestion du statut des utilisateurs (actif, suspendu, etc.)

TESTS ET VALIDATION

Pour garantir la qualité et la fiabilité de notre application, nous avons mis en place une stratégie de tests à plusieurs niveaux:



TESTS UNITAIRES

Des tests unitaires ont été développés pour valider le comportement des principales classes du modèle (Livre, Utilisateur, Emprunt). Ces tests vérifient notamment:

- La création et la modification des objets
- La validation des données (ISBN valide, dates cohérentes, etc.)
- Le comportement des méthodes de gestion (emprunt, retour, etc.)

TESTS D'INTEGRATION

Des tests d'intégration ont été réalisés pour valider les interactions entre les différentes composantes du système, notamment:

- La communication entre le modèle et les contrôleurs

- La persistance et la restauration des données
- La synchronisation entre les différentes vues

TESTS FONCTIONNELS

Des scénarios de test complets ont été exécutés pour valider les fonctionnalités de l'application dans des conditions réelles d'utilisation:

- Enregistrement d'un nouveau livre
- Inscription d'un nouvel utilisateur
- Emprunt et retour de livres
- Recherche de livres selon différents critères
- Gestion des cas particuliers (utilisateur suspendu, livre déjà emprunté, etc.)

DIFFICULTES RENCONTREES ET SOLUTIONS

DIFFICULTES TECHNIQUES

1. **Implémentation de l'Interface Graphique avec JavaFX**
    - **Problème**: Navigation à vue dans JavaFX sans guide ou expérience préalable, difficulté à structurer correctement l'interface
    - **Solution**: Étude approfondie de la documentation et des tutoriels JavaFX, utilisation de Scene Builder pour la conception visuelle des interfaces FXML, approche incrémentale avec prototypage
2. **Gestion des Événements UI**
    - **Problème**: Coordination entre les différents contrôleurs JavaFX et mise à jour en temps réel des vues
    - **Solution**: Implémentation d'un système d'observateurs et d'écouteurs d'événements pour maintenir la cohérence des différentes vues
3. **Intégration avec SQLite**
    - **Problème**: Gestion des connexions à la base de données et des transactions, particulièrement lors d'opérations complexes
    - **Solution**: Implémentation d'un gestionnaire de connexion centralisé avec pattern singleton, utilisation de PreparedStatements pour éviter les injections SQL
4. **Gestion des Dates**
    - **Problème**: Manipulation complexe des dates pour le calcul des durées d'emprunt et des retards
    - **Solution**: Création d'une classe utilitaire dédiée aux opérations sur les dates et standardisation du format

Difficultés Organisationnelles

1. **Développement sans Guide Précis**
    - **Problème**: Difficulté à structurer le projet et à prioriser les tâches sans guidage détaillé
    - **Solution**: Établissement d'un plan de développement incrémental, définition de jalons clairement identifiés, revues régulières de l'avancement
2. **Équilibre entre Fonctionnalités et Délais**
    - **Problème**: Risque de dispersion face à la multitude de fonctionnalités possibles
    - **Solution**: Priorisation des fonctionnalités essentielles selon le cahier des charges, création d'un MVP (Minimum Viable Product) avant d'ajouter des fonctionnalités supplémentaires
3. **Apprentissage de Nouvelles Technologies**
    - **Problème**: Temps nécessaire pour maîtriser JavaFX et SQLite tout en avançant sur le développement
    - **Solution**: Sessions de formation autodidactes intensives, développement de petits prototypes pour tester et comprendre les concepts avant de les intégrer dans le projet principal

CONCLUSION ET PERSPECTIVES

BILAN DU PROJET

Ce projet nous a permis de mettre en pratique de nombreux concepts de la programmation orientée objet et de développer une application fonctionnelle répondant aux besoins de gestion d'une petite bibliothèque. L'implémentation d'une interface graphique, bien que non requise initialement, a considérablement enrichi l'expérience utilisateur et nous a permis d'explorer des aspects avancés du développement Java.

Les principales réussites du projet sont:

- Une architecture bien structurée suivant le modèle MVC
- Une interface utilisateur intuitive et complète
- Un système de persistance des données robuste
- Une couverture fonctionnelle répondant aux exigences initiales

AMELIORATIONS FUTURES

Plusieurs axes d'amélioration ont été identifiés pour de futures versions:

BASE DE DONNEES

- Remplacement du système de fichiers CSV par une véritable base de données relationnelle
- Implémentation d'un mécanisme de migration des données existantes

FONCTIONNALITES AVANCEES

- Système de réservation de livres
- Gestion des amendes pour les retards
- Intégration d'un module de statistiques avancées

INTERFACE UTILISATEUR

- Personnalisation des thèmes visuels
- Version responsive pour différentes tailles d'écran
- Version web accessible à distance

SECURITE

- Système d'authentification des bibliothécaires
- Gestion des droits d'accès selon les rôles

Guide d'Installation

1. Prérequis
   - Java Runtime Environment (JRE) 11 ou supérieur
   - Espace disque: minimum 100 Mo
2. Procédure d'Installation
   - Télécharger l'archive bibliotheque.zip
   - Extraire le contenu dans le répertoire souhaité
   - Vérifier que le répertoire lib/ contient bien le fichier sqlite-jdbc-3.36.0.3.jar
   - Exécuter le fichier bibliotheque.jar en double-cliquant dessus ou via la commande:

java -jar bibliotheque.jar

3. Premier Lancement
   - Lors du premier lancement, l'application créera automatiquement le répertoire db/ et le fichier de base de données bibliotheque.db
   - Aucune configuration supplémentaire n'est nécessaire

MANUEL UTILISATEUR

Le manuel utilisateur complet est disponible dans le fichier `manuel_utilisateur.pdf` inclus dans l'archive d'installation. Il détaille l'ensemble des fonctionnalités de l'application et les procédures à suivre pour chaque opération.

JAVADOC

La documentation complète du code source est disponible au format Javadoc dans le répertoire doc/ de l'archive d'installation. Elle peut être consultée en ouvrant le fichier index.html dans un navigateur web.

Références

- Oracle Java Documentation: https://docs.oracle.com/en/java/
- Java Swing Tutorial: https://docs.oracle.com/javase/tutorial/uiswing/
- Design Patterns en Java: "Head First Design Patterns" par Eric Freeman et Elisabeth Robson
- Java CSV Library: OpenCSV (http://opencsv.sourceforge.net/)