IntelliLib – Système Moderne de Gestion de Bibliothèque

Rapport Technique

Auteurs : El Afifi Youssef - Boukakar Ayoub

Cours : Programmation Java Avancée

Établissement : Ecole Nationale des Sciences Appliquées Al Hoceima

Superviseur :M. Abdelkhalak Bahri

Date : December 27, 2025

Contents

Résumé

IntelliLib est un système de gestion de bibliothèque moderne développé avec JavaFX et Spring Boot. Il offre une gestion sécurisée, modulaire et efficace des ressources de la bibliothèque via une interface conviviale. Le système met en œuvre un contrôle d'accès basé sur les rôles avec trois types d'utilisateurs (Administrateur, Bibliothécaire et Membre), le traitement des documents et la génération de rapports analytiques. Construit avec Java 21 et SQLite, il démontre une architecture MVC propre et une intégration solide du backend avec Spring Security. Le résultat est une solution évolutive et performante pour la gestion des opérations de bibliothèque, numériques et physiques.

1. Introduction

1.1. Contexte

Les bibliothèques ont longtemps été des institutions essentielles pour le stockage, la préservation et la diffusion des connaissances. Cependant, avec l'expansion rapide des ressources d'information et la diversité croissante des services de bibliothèque, les méthodes traditionnelles de gestion manuelle des catalogues, des membres et des transactions sont devenues inefficaces, sujettes aux erreurs et chronophages. Les systèmes papier ou partiellement numérisés entraînent souvent des redondances de données, des enregistrements incohérents et un accès limité, notamment lorsqu'il s'agit de gérer de grandes collections et plusieurs utilisateurs simultanément.

La transition mondiale vers la transformation numérique a créé le besoin de solutions de gestion intelligentes, automatisées et orientées utilisateur. Un système de gestion de bibliothèque robuste simplifie non seulement les tâches administratives quotidiennes telles que le prêt de livres, l'inscription des membres et la gestion des amendes, mais permet également un accès en temps réel aux informations et améliore l'expérience utilisateur. En tirant parti de technologies modernes telles que JavaFX et Spring Boot, les bibliothèques peuvent fonctionner avec une plus grande efficacité, une meilleure précision et une qualité de service améliorée.

1.2. Objectif

Le **IntelliLib – Système moderne de gestion de bibliothèque** est conçu pour répondre aux limites des opérations de bibliothèque traditionnelles en fournissant une plateforme numérique complète, modulaire et sécurisée. Le système intègre les fonctionnalités d'administration de la bibliothèque, de gestion des utilisateurs et de gestion des transactions d'amendes dans une application de bureau unifiée et facile à utiliser.

Les objectifs d'IntelliLib sont de:

- Automatiser et rationaliser les processus principaux de la bibliothèque tels que le prêt et le retour de livres, l'inscription des membres et le suivi des amendes.

- Fournir un système d'authentification sécurisé avec contrôle d'accès basé sur les rôles pour garantir la gestion appropriée des permissions.
- Gérer automatiquement les transactions d'amendes, suivre les livres en retard et appliquer les pénalités.
- Offrir une base évolutive qui supporte des extensions futures, incluant le traitement de documents, des tableaux de bord analytiques et des recommandations assistées par IA.

1.3. Portée

Le système IntelliLib couvre un cycle complet de gestion des opérations de bibliothèque, structuré autour de multiples rôles utilisateurs avec des privilèges et restrictions spécifiques. L'application offre des fonctionnalités pour les administrateurs, les bibliothécaires et les membres, garantissant que chaque rôle interagit avec le système selon ses responsabilités.

Les principaux domaines fonctionnels incluent:

- **Authentification et autorisation des utilisateurs:** Système de connexion sécurisé utilisant Spring Security avec des identifiants chiffrés et gestion de session.
- **Gestion des livres et ressources:** Opérations CRUD complètes sur les livres, catégories et enregistrements de prêt.
- **Gestion des transactions d'amendes:** Calcul et suivi automatique des amendes pour livres en retard, incluant les notifications.
- **Traitement et analyse des documents (plan futur):** Prévision pour la gestion de documents avec OCR et extraction de texte (non encore implémenté).
- **Gestion des utilisateurs et membres:** Outils pour créer et lier des comptes utilisateurs aux profils membres, gérer les états de compte et suivre les activités.
- **Système de notification et reporting:** Alertes automatisées pour les livres en retard, journaux système et résumés analytiques.

1.4. Structure du projet

Le projet IntelliLib adopte une architecture modulaire et en couches, combinant la puissance de **JavaFX** pour une interface graphique interactive avec la robustesse de **Spring**

Boot pour les services backend et la gestion des données. Le système suit le modèle **Model–View–Controller (MVC)**, garantissant maintenabilité, évolutivité et séparation claire des responsabilités.

Les principaux composants comprennent:

- **Couche frontend (présentation):** Implémentée avec JavaFX 21.0.3 et des mises en page basées sur FXML, offrant une interface intuitive pour les utilisateurs. Chaque vue est connectée à un contrôleur correspondant pour la gestion des événements.
- **Couche backend (logique applicative):** Alimentée par Spring Boot 3.2.6, cette couche gère la logique métier, la validation et l'orchestration des services entre l'interface utilisateur et la couche de données.
- **Couche de données (persistance):** Utilise SQLite 3.46.0 avec Hibernate ORM pour le mapping objet-relationnel et des opérations efficaces sur la base de données. Cela garantit que les données sont stockées, récupérées et mises à jour de manière sécurisée et cohérente.
- **Module de sécurité:** Utilise Spring Security pour l'authentification, l'autorisation et le contrôle des sessions avec hachage de mot de passe (BCrypt).
- **Modules Amendes et Utilitaires:** Gèrent les transactions d'amendes, la journalisation des activités, la gestion des fichiers et les fonctionnalités de reporting.
- **Module Vision (plan futur):** Conçu pour l'analyse de documents basée sur OCR et l'extraction de texte, à intégrer dans une version future.

En combinant ces technologies, IntelliLib fournit un écosystème numérique flexible et puissant pour la gestion des bibliothèques. La modularité de l'architecture permet l'intégration future avec des services cloud, des API REST et des plateformes mobiles, garantissant la pertinence et l'adaptabilité à long terme du système.

2. Architecture du Système

2.1. Architecture de Haut Niveau

2.1.1. Couches Architecturales

Le système implémente une architecture en trois tiers avec une séparation claire des responsabilités:

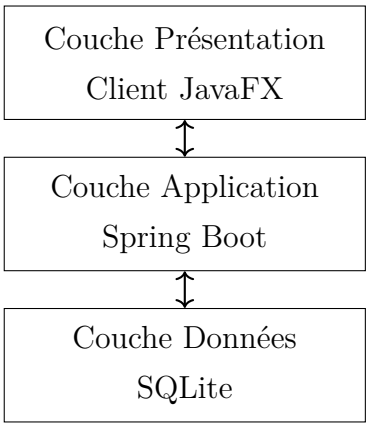

Figure 2.1: Architecture en trois tiers

2.1.2. Flux d'Interaction des Composants

Le système suit le modèle Model-View-Controller (MVC) avec un flux de données clair entre les composants:

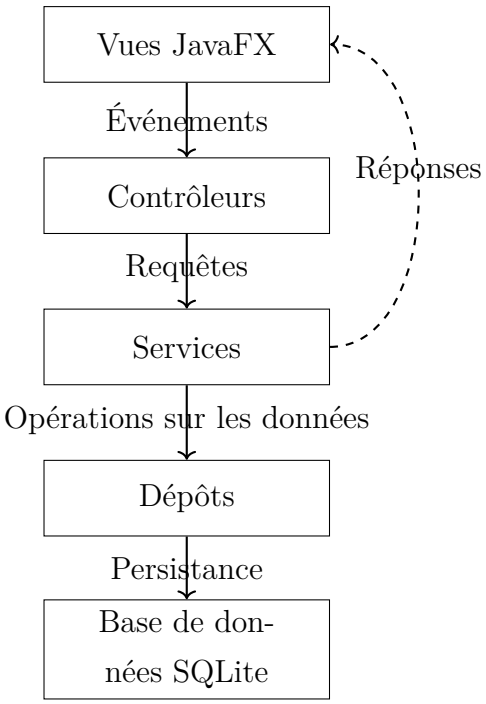

Figure 2.2: Flux d'Interaction des Composants

2.2. Répartition Détaillée des Composants

2.2.1. Couche Présentation

Point d'entrée de l'application: MainApp.java

La classe `MainApp` sert de point d'entrée à l'application IntelliLib. Elle étend `javafx.application.Application` et intègre le frontend JavaFX avec le backend Spring Boot.

Responsabilités principales:

- Initialiser le contexte Spring Boot au démarrage
- Configurer `FXMLLoaderUtil` pour activer l'injection de dépendances dans les contrôleurs
- Charger la scène FXML principale pour la fenêtre de l'application
- Gérer la fermeture correcte du contexte Spring à la sortie de l'application

Responsabilités des Contrôleurs FXML

Les contrôleurs FXML agissent comme pont entre **les interactions utilisateur** et les **services backend**. Leurs principales responsabilités incluent:

- **Gestion des événements UI:** Capturer les actions des utilisateurs (clics, sélection de menu, soumission de formulaires) et déclencher les requêtes backend correspondantes
- **Validation des entrées:** Vérifier que les saisies utilisateur sont complètes, correctement formatées et cohérentes avant d'appeler les services
- **Coordination des services:** Appeler les méthodes de la couche service pour exécuter la logique métier (gestion des utilisateurs, prêts de livres, traitement des amendes)
- **Gestion de l'état UI:** Mettre à jour dynamiquement l'interface, afficher les notifications et refléter les changements de l'application
- **Conscience de la session:** Maintenir les informations de session et appliquer l'accès UI basé sur les rôles

2.2.2. Couche Application (Backend)

Couche Contrôleur

Les contrôleurs backend traitent les requêtes provenant de la couche présentation et coordonnent avec la couche service pour exécuter la logique métier. Le flux d'information suit cette séquence:

1. **Action utilisateur:** L'utilisateur interagit avec la vue JavaFX (ex. clic sur un bouton)
2. **Gestion par le contrôleur:** Le contrôleur FXML correspondant capture l'événement, valide les entrées et appelle le service approprié
3. **Traitement par le service:** Les services exécutent la logique métier requise, comme le calcul des amendes ou la mise à jour des prêts
4. **Interaction avec le dépôt:** Les services communiquent avec les dépôts pour stocker ou récupérer les données dans la base SQLite
5. **Mise à jour de la réponse:** Les services renvoient les résultats au contrôleur qui met à jour l'UI en conséquence

La couche contrôleur est organisée en trois grandes catégories comme indiqué dans le Tableau 2.1.

Catégorie	Contrôleurs	Responsabilités principales
Admin	ManageBookController	Opérations CRUD sur les livres, gestion des inventaires
	ManageBorrowController	Gestion des prêts, suivi des dates d'échéance
	ManageCategoryController	Gestion hiérarchique des catégories
	ManageFinesController	Calcul et traitement des amendes
	ManageMemberController	Enregistrement et gestion des membres
	ManageUserController	Gestion des comptes et rôles utilisateurs
Member	BrowseBooksController	Recherche et découverte de livres
	MyBorrowingsController	Gestion personnelle des prêts
	MyFinesController	Consultation et paiement des amendes personnelles
	RecommendationsController	Suggestions personnalisées de livres
Authentication	LoginController	Authentification des utilisateurs
	RegisterController	Enregistrement des nouveaux utilisateurs
	ProfileController	Gestion des profils utilisateurs

Table 2.1: Organisation de la couche Contrôleur

Couche Service

La couche Service représente la logique métier centrale d'IntelliLib. Chaque classe de service encapsule un domaine fonctionnel spécifique et agit comme médiateur entre la couche contrôleur et la couche dépôt. Les services garantissent que toutes les règles métier sont appliquées de manière cohérente et que l'intégrité des données est maintenue avant toute interaction avec la base de données.

Principaux services:

- **UserService:** Gère toutes les opérations liées aux utilisateurs, y compris l'inscription, l'authentification, l'autorisation et la gestion des profils. Intégré à Spring Security pour gérer les mots de passe chiffrés et les rôles utilisateurs.
- **BookService:** Gère le cycle complet des livres — enregistrement, catégorisation, recherche et gestion des inventaires. Assure également la cohérence des relations de catégories.

- **BorrowService:** Contrôle le processus de prêt, y compris l'émission, le renouvellement, le retour et le suivi des retards. Interagit étroitement avec `BookService` et `FineService`.
- **FineService:** Calcule automatiquement les amendes selon le nombre de jours de retard et les règles définies. Met à jour les entités `Borrow` et `Member` pour maintenir la synchronisation.
- **MemberService:** Gère la création, le lien et la mise à jour des profils membres. Assure que chaque utilisateur avec le rôle MEMBER possède une entité membre valide.
- **CategoryService:** Gère la catégorisation des livres et fournit des utilitaires pour le filtrage et l'organisation hiérarchique.
- **RecommendationService:** Fournit des recommandations de livres aux membres selon l'historique de prêt et les préférences de catégorie.
- **NotificationService:** Envoie des notifications pour les livres en retard, les actions sur le compte et les alertes système.

L'organisation de la couche service est détaillée dans le Tableau 2.2.

Catégorie	Service	Responsabilités principales
Gestion des utilisateurs	UserService	Gère l'inscription, l'authentification et la mise à jour des profils. Intégré à Spring Security pour le contrôle d'accès basé sur les rôles.
	MemberService	Gère les profils des membres et les lie aux comptes utilisateurs. Assure la cohérence des données entre les entités User et Member.
Opérations de bibliothèque	BookService	Gère l'inventaire des livres, la catégorisation et la recherche. Assure la cohérence des associations livre-catégorie.
	CategoryService	Maintient la hiérarchie des catégories et fournit des utilitaires pour les recherches filtrées de livres.
Prêts et amendes	BorrowService	Gère le cycle de prêt complet, incluant emprunt, retour et renouvellement. Met à jour la disponibilité des livres et les enregistrements des membres.
	FineService	Calcule les amendes pour les livres en retard et enregistre les transactions d'amendes pour chaque membre. Maintient la synchronisation entre Borrow et FineTransaction.
Services supplémentaires	NotificationService	Envoie des notifications automatiques dans l'application, telles que des alertes de retard, confirmations d'inscription et messages système.

Table 2.2: Organisation de la couche Service

2.2.3. Modèle de Données

Aperçu des entités

Le système IntelliLib utilise un modèle de données complet composé de sept entités principales représentant les concepts métiers fondamentaux de la gestion de bibliothèque. Ces entités sont conçues avec une normalisation appropriée et un mapping des relations pour assurer l'intégrité des données et des opérations efficaces.

Relations entre entités

Les relations entre les entités forment un réseau complexe qui soutient les flux opérationnels de la bibliothèque. Les relations principales sont représentées dans le diagramme de relations d'entités (ERD) de la Figure 2.3.

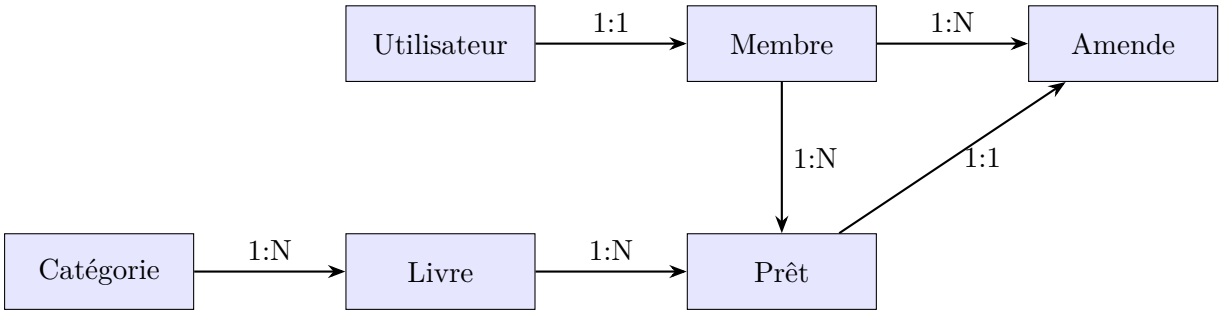

Figure 2.3: Diagramme des relations d'entités

Description des entités

- **Entité Utilisateur**: Sert de composant central pour l'authentification et l'autorisation, implémente l'interface UserDetails de Spring Security et gère les identifiants, le contrôle d'accès basé sur les rôles, l'état du compte et la gestion de session.
- **Entité Membre**: Représente les usagers de la bibliothèque avec des capacités avancées de gestion incluant informations personnelles, cycle de vie du membre, suivi financier, surveillance comportementale et validation de l'éligibilité au prêt.
- **Entité Livre**: Gère la collection de la bibliothèque pour les ressources physiques et numériques, incluant les informations bibliographiques, la gestion des inventaires, la gestion des ressources numériques, la catégorisation et le suivi temporel.
- **Entité Prêt**: Orchestration complète du cycle de prêt avec suivi des transactions, calcul des amendes, gestion des statuts, suivi des renouvellements et intégration avec le système d'amendes.

- **Entité Amende**: Gère les aspects financiers des matériels en retard avec traitement multi-méthodes, gestion du cycle de transaction, génération de reçus et suivi d'audit.
- **Entité Catégorie**: Fournit une structure organisationnelle pour la collection grâce à un système de catégorisation hiérarchique, des métadonnées descriptives et un design évolutif.
- **Entité Activité**: Maintient des pistes d'audit complètes avec journalisation des actions utilisateur, suivi temporel, surveillance de sécurité via l'enregistrement des adresses IP et analyse des performances.

Types de relations

- **Relations Un-à-Un**:
  - **Utilisateur** $\leftrightarrow$ **Membre**: Chaque membre a exactement un compte utilisateur, et chaque compte utilisateur de type MEMBER est lié à un seul profil membre.
  - **Prêt** $\leftrightarrow$ **Amende**: Un prêt peut générer zéro ou une transaction d'amende (relation optionnelle pour les amendes annulées ou exemptées).
- **Relations Un-à-Plusieurs**:
  - **Catégorie** $\rightarrow$ **Livres**: Chaque catégorie peut contenir plusieurs livres, mais chaque livre appartient à une seule catégorie.
  - **Membre** $\rightarrow$ **Prêts**: Un membre peut avoir plusieurs enregistrements de prêt actifs ou historiques.
  - **Livre** $\rightarrow$ **Prêts**: Un livre peut être emprunté plusieurs fois par différents membres au cours de sa vie.
  - **Membre** $\rightarrow$ **Amendes**: Un membre peut accumuler plusieurs transactions d'amendes.
  - **Utilisateur** $\rightarrow$ **Activités**: Chaque compte utilisateur génère plusieurs enregistrements d'activité pour l'audit.
- **Relations Plusieurs-à-Plusieurs (résolues)**: La relation entre Membres et Livres est résolue via l'entité Prêt, qui sert de table de jonction avec des attributs temporels et financiers supplémentaires.

Encapsulation de la logique métier

Chaque entité encapsule une logique métier spécifique via des méthodes assurant l'intégrité des données et le respect des règles métier, comme détaillé dans le Tableau 2.3.

Entité	**Méthodes clés de la logique métier**
Membre	**canBorrow()**: Valide l'éligibilité au prêt **isBanExpired()**: Vérifie si une interdiction est expirée **hasUserAccount()**: Vérifie la liaison avec un compte utilisateur
Prêt	**isOverdue()**: Vérifie le statut temporel **calculateDaysOverdue()**: Calcule la durée de retard **calculateFine()**: Calcule la pénalité financière
Amende	**generateReceiptNumber()**: Génération automatique de reçus Validation et traitement du moyen de paiement
Livre	**toText()**: Représentation standardisée sous forme de chaîne Suivi et gestion de la disponibilité

Table 2.3: Méthodes de la logique métier par entité

Contraintes d'intégrité des données

Le modèle de données impose plusieurs contraintes:

- **Intégrité référentielle**: Suppressions en cascade pour les enregistrements dépendants, contraintes de clé étrangère avec indexation appropriée et contraintes de nullabilité pour les relations optionnelles.
- **Contraintes d'unicité**: Unicité des noms d'utilisateur et emails pour l'entité User, unicité des ISBN pour les livres, unicité des noms de catégorie, unicité des emails et téléphones des membres.
- **Règles de validation**: Validation des rôles (Admin, Bibliothécaire, Membre), statut d'amende (NONE, PENDING, PAID, WAIVED, CANCELLED), moyen de paiement (CASH, CREDIT_CARD, etc.) et statut de transaction (PENDING, COMPLETED, etc.).

Gestion temporelle des données

Le système utilise plusieurs mécanismes de suivi temporel:

- **Horodatages**: LocalDateTime pour un timing précis des événements (création d'utilisateur, connexions), LocalDate pour les événements datés (prêts, échéances, adhésion), et Long (Epoch) pour les audits nécessitant une précision milliseconde.
- **Valeurs par défaut**: Génération automatique d'horodatages à la création d'une entité, valeurs par défaut pour les nouveaux enregistrements et champs calculés avec valeurs par défaut logiques.

Considérations de performance

Le modèle de données est optimisé pour différents types d'opérations:

- **Optimisation en lecture**: Indexation des requêtes fréquentes (recherche de livres, consultation des membres), chargement différé pour les grandes collections pour minimiser l'empreinte mémoire, et chargement anticipé pour les relations fréquemment utilisées.
- **Optimisation en écriture**: Verrouillage minimal via contrôle de concurrence optimiste, opérations par lot pour le traitement en masse, et mises à jour efficaces des champs modifiés fréquemment.

Les stratégies d'optimisation des requêtes sont détaillées dans le Tableau 2.4.

Type de requête	**Stratégie d'optimisation**
Authentification utilisateur	Index composite sur (username, active)
Recherche de livres	Index full-text sur (title, author, ISBN)
Détection des retards	Index sur (due_date, returned) avec comparaison de dates
Calcul des amendes	Vue matérialisée pour les amendes fréquemment calculées
Audit des activités	Partitionnement par plage de dates pour les données historiques

Table 2.4: Stratégies d'optimisation des requêtes

Ce modèle de données complet fournit une base robuste pour toutes les opérations de gestion de bibliothèque tout en maintenant la flexibilité pour des améliorations futures et l'évolutivité pour des collections et des utilisateurs en croissance.

2.3. Flux et traitement des données

2.3.1. Flux d'authentification des utilisateurs

1. L'utilisateur soumet ses identifiants via LoginController
2. UserService valide les identifiants dans la base de données
3. Spring Security crée un token d'authentification
4. SessionManager établit la session utilisateur
5. L'utilisateur est redirigé vers le tableau de bord approprié à son rôle

2.3.2. Processus de prêt de livres

1. Le membre sélectionne un livre via BrowseBooksController
2. BookService vérifie la disponibilité (available_copies > 0)
3. BorrowService valide les limites de prêt du membre
4. Un enregistrement de prêt est créé avec la date d'échéance calculée
5. La disponibilité du livre est mise à jour atomiquement
6. Le membre est notifié du succès du prêt

2.4. Architecture de sécurité

2.4.1. Authentification et autorisation

- **Chiffrement des mots de passe**: Hachage BCrypt avec sel
- **Gestion des sessions**: HttpSession avec durée configurable
- **Protection CSRF**: Activée pour les opérations modifiant l'état
- **Limitation du taux**: Protection contre les attaques par force brute

2.4.2. Matrice de contrôle d'accès basé sur les rôles

Permission	Administrateur	Bibliothécaire	Membre
Créer/Modifier des utilisateurs	✓	Limité	×
Gérer les livres	✓	✓	×
Traiter Prêt/Retour	✓	✓	Seulement pour soi
Gérer les amendes	✓	✓	Voir/Payer uniquement
Voir les rapports	✓	Limité	Personnel uniquement
Gérer les catégories	✓	✓	×

Table 2.5: Matrice de contrôle d'accès basé sur les rôles

2.5. Stratégie de gestion des erreurs

2.5.1. Gestion des exceptions en couches

- **Couche présentation**: Affiche des messages d'erreur conviviaux et propose des options de récupération (réessayer, annuler, ou aide).
- **Couche contrôleur**: Capture les exceptions des services, les traduit en messages compréhensibles pour l'UI et maintient l'état de l'application.
- **Couche service**: Lance des exceptions métier spécifiques avec un contexte clair (ex. livre non disponible, membre banni).
- **Couche dépôt**: Gère les exceptions d'accès aux données, assure le rollback en cas d'échec et journalise les erreurs pour le diagnostic.

Cette architecture fournit une base solide pour le système IntelliLib, équilibrant performance, maintenabilité et évolutivité tout en supportant les flux de travail complexes de la gestion moderne de bibliothèque. Le design modulaire permet une extension et une adaptation faciles aux besoins futurs.

3. Implémentation

3.1. Aperçu

Ce chapitre décrit l'implémentation du système IntelliLib, couvrant l'environnement de développement, l'architecture technique et la mise en œuvre parallèle des modules Membre et Administrateur avec leurs fonctionnalités et interfaces distinctes.

3.2. Environnement de Développement et Outils

Le système IntelliLib a été développé en utilisant un ensemble complet d'outils et de frameworks modernes pour assurer un développement robuste, des tests efficaces et un déploiement fluide.

Composant	**Technologie / Outil**
Langage de Programmation	Java 21
Interface Graphique	JavaFX avec FXML
Base de Données	SQLite 3.x
Mapping Objet-Relationnel	Hibernate ORM
Gestion des Dépendances et Build	Maven (Apache Maven)
Framework de Sécurité	Spring Security
Environnement de Développement Intégré	IntelliJ IDEA (recommandé)
Contrôle de Version	Git
Framework de Test	JUnit 5

Table 3.1: Environnement de développement et outils

3.3. Architecture Technique

Le système suit une architecture modulaire et en couches qui sépare les préoccupations de présentation, de logique métier et de persistance des données, avec un contrôle d'accès basé sur les rôles.

3.3.1. Architecture de la Couche Service

Service	Responsabilité	Méthodes clés
ActivityService	Journalisation des activités et analyse	logActivity(), getActivity-ChartData()
BookService	Gestion des livres et gestion des fichiers	saveBookWithFile(), update-Book()
BorrowService	Opérations d'emprunt et suivi	borrowBook(), returnBook()
FineService	Calcul et paiement des amendes	calculateDailyFines(), process-FinePayment()
MemberService	Gestion des profils des membres	createMember(), updateMem-ber()
UserService	Authentification et gestion des utilisateurs	registerUser(), updateUser()

Table 3.2: Responsabilités de la couche service

3.3.2. Implémentation de la Sécurité

Le **contrôle d'accès basé sur les rôles** est appliqué via la configuration Spring Security.

Listing 3.1: SecurityConfig.java - Règles d'accès HTTP

```
@Configuration
@EnableMethodSecurity
public class SecurityConfig {

     @Bean
     public PasswordEncoder passwordEncoder() {
         return new BCryptPasswordEncoder();
     }
}
```

3.4. Implémentation des Modules Parallèles

Fonctionnalités du Module Membre	**Fonctionnalités du Module Administrateur**
Navigation et recherche de livres	Administration complète du système
Emprunt et retour de livres	Gestion du catalogue de livres
Historique personnel des emprunts	Gestion et suivi des membres
Consultation et paiement des amendes	Calcul et administration des amendes
Gestion du profil	Gestion des comptes utilisateurs
Lecture de fichiers PDF pour livres numériques	Analyse et reporting du système
Accès limité aux données personnelles	Accès à toutes les données du système
Opérations en libre-service	Opérations en masse et automatisation

Table 3.3: Comparaison parallèle des fonctionnalités entre les modules Membre et Administrateur

3.4.1. Implémentation du Module Membre

Le module membre fournit des interfaces en libre-service pour les usagers de la bibliothèque.

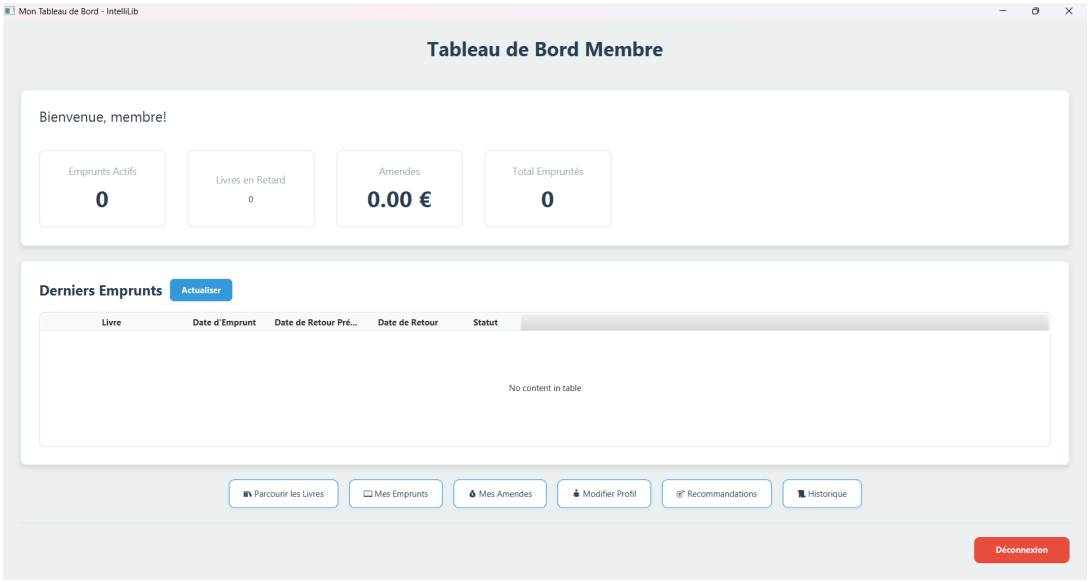

Figure 3.1: Interface du tableau de bord membre avec résumé personnel

Fonctionnalités du tableau de bord membre :

- Statistiques personnelles des emprunts

- Notifications de livres en retard
- Statut des amendes et options de paiement
- Accès rapide aux emprunts et à la navigation

Interface de navigation des livres :

Figure 3.2: Interface de navigation des livres avec recherche et filtrage

Flux d'emprunt pour les membres :

Listing 3.2: MemberController.java - Processus d'emprunt

```
@FXML
private void handleBorrowBook() {
    Book selectedBook = booksTable.getSelectionModel().
       getSelectedItem();
    Member currentMember = sessionManager.getCurrentMember();

    if (selectedBook != null && currentMember.canBorrow()) {
        borrowService.borrowBook(selectedBook.getId(),
           currentMember.getId());
        showSuccess("Livre␣emprunt ␣avec␣succ s␣!");
        refreshAvailableBooks();
    }
}
```

3.4.2. Implémentation du Module Administrateur

Le module administrateur fournit des capacités complètes d'administration du système.

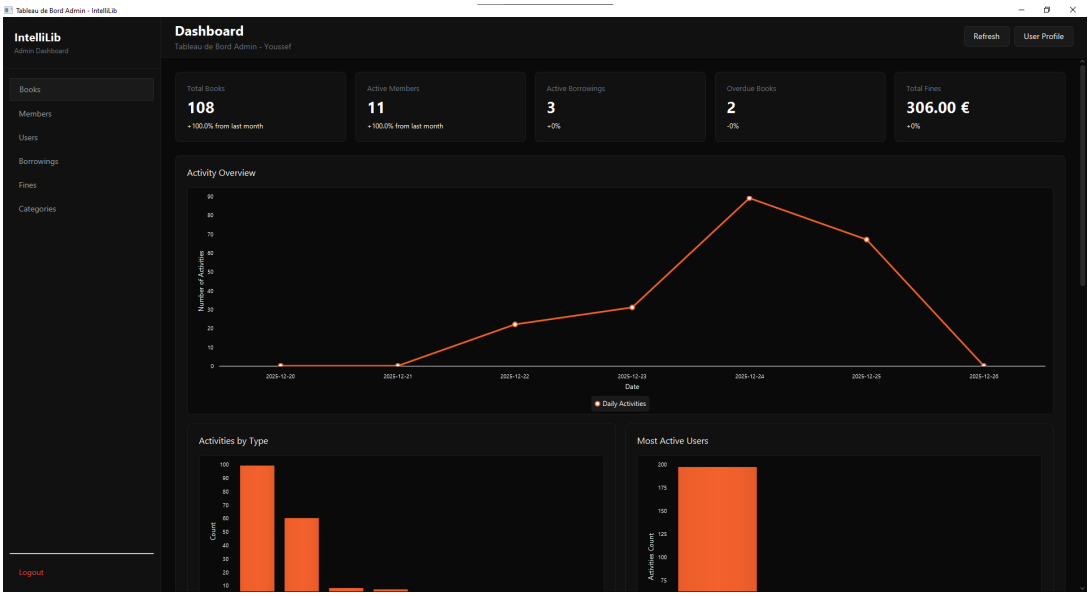

Figure 3.3: Tableau de bord administrateur avec analyse système globale

Fonctionnalités du tableau de bord administrateur :

- Statistiques et analyses système globales
- Surveillance des activités en temps réel
- Indicateurs de performance et graphiques
- Accès rapide à tous les modules de gestion

Gestion des Livres (ManageBookController)

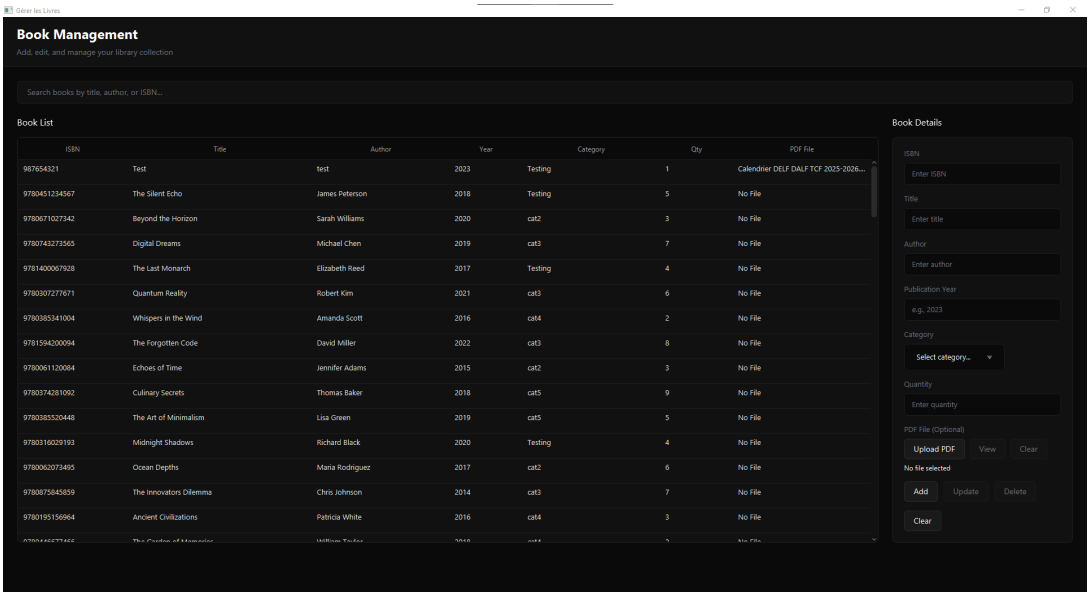

Figure 3.4: Interface de gestion des livres avec opérations en masse

Fonctionnalités de gestion des livres :

- Suivi de l'inventaire par ISBN
- Organisation par catégories
- Téléversement et aperçu des fichiers PDF
- Gestion des quantités et disponibilités
- Recherche et filtrage avancés

Listing 3.3: ManageBookController.java - Opérations sur les livres

```
@FXML
private void handleAddBook() {
    if (validateInput()) {
        Book book = createBookFromForm();
        Book savedBook = bookService.saveBookWithFile(book,
            selectedFile);
        activityLogger.logBookAdd(currentUser, savedBook.getTitle
            ());
        showStatus("Livre␣ajout ␣avec␣succ s␣!");
    }
}
```

Gestion des Emprunts (ManageBorrowController)

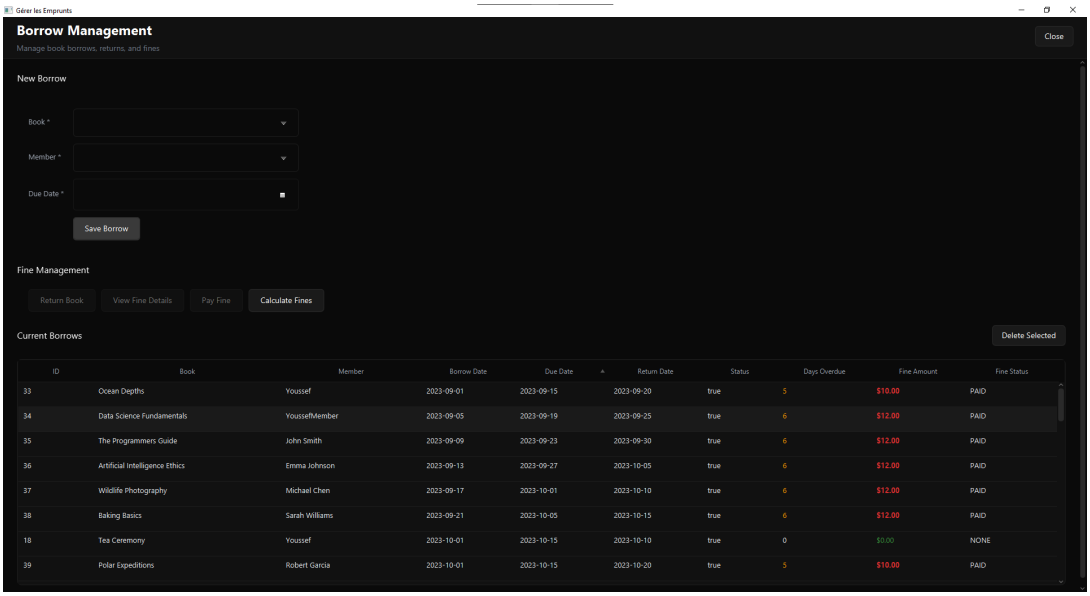

Figure 3.5: Interface de gestion des emprunts pour l'émission, le retour et le renouvellement

Fonctionnalités de gestion des emprunts :

- Gestion complète du cycle d'emprunt (émission, retour, renouvellement)
- Détection automatique des retards et calcul des amendes
- Validation de l'éligibilité des membres
- Traitement en masse des retours et annulation d'amendes

Listing 3.4: ManageBorrowController.java - Opérations sur les emprunts

```
@FXML
private void handleIssueBorrow() {
    Borrow borrow = createBorrowFromForm();
    borrowService.issueBorrow(borrow);
    activityLogger.logBorrow(currentUser, borrow.getBook().
       getTitle());
    showStatus("Emprunt␣ mis ␣avec␣succ s␣!");
}
```

Gestion des Catégories (ManageCategoryController)

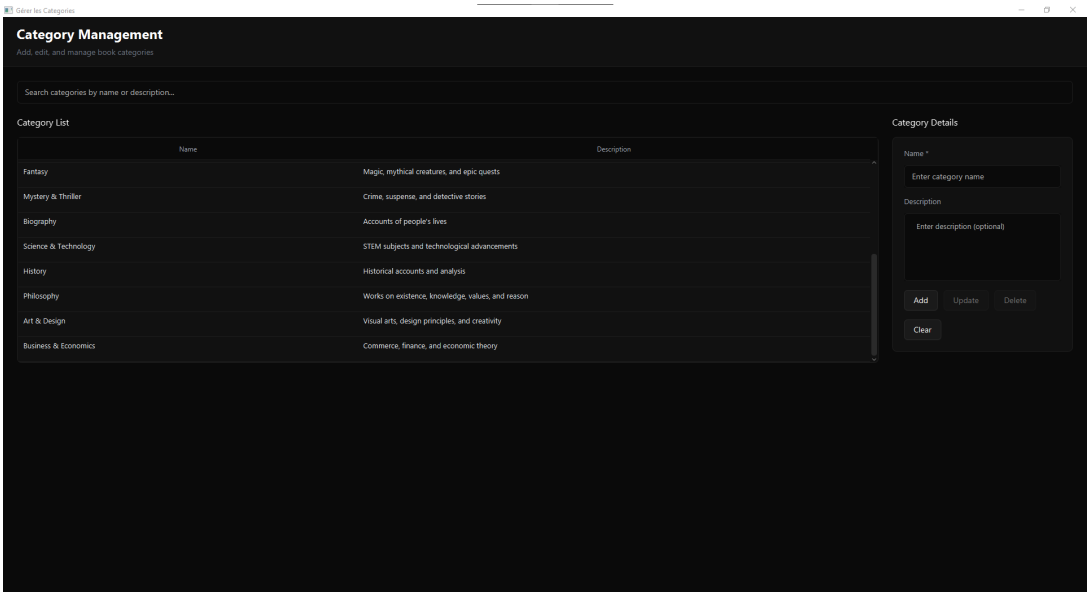

Figure 3.6: Interface de gestion des catégories avec opérations ajouter/modifier/supprimer

Fonctionnalités de gestion des catégories :

- Ajouter, modifier, supprimer des catégories de livres
- Organisation par catégories pour un filtrage facile
- Intégration avec l'inventaire et la recherche

Listing 3.5: ManageCategoryController.java - Opérations sur les catégories

```
@FXML
private void handleAddCategory() {
    Category category = createCategoryFromForm();
    categoryService.saveCategory(category);
    showStatus("Cat gorie␣ajout e␣avec␣succ s␣!");
}
```

Gestion des Amendes (ManageFinesController)

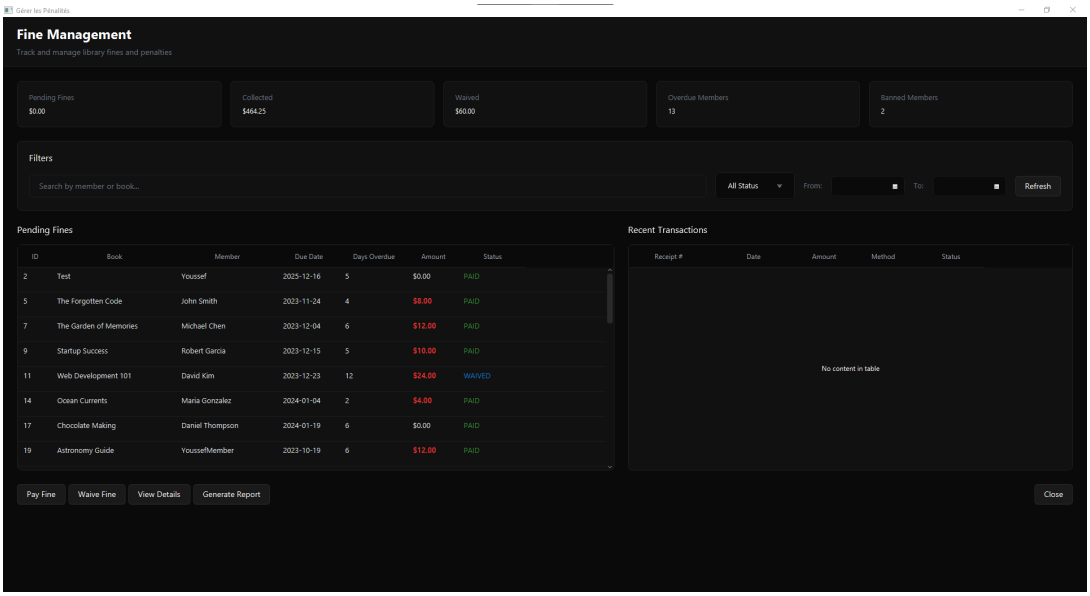

Figure 3.7: Interface de gestion des amendes pour les livres en retard et paiements

Fonctionnalités de gestion des amendes :

- Calcul automatique des amendes journalières
- Multiples méthodes de paiement (espèces, carte, en ligne)
- Workflow d'annulation d'amende avec suivi des raisons
- Rapports de transaction et gestion des suspensions de membres

Listing 3.6: ManageFinesController.java - Opérations sur les amendes

```
@FXML
private void handleProcessFine() {
    Fine fine = fineService.calculateFine(selectedBorrow);
    fineService.processPayment(fine);
    activityLogger.logFinePayment(currentUser, fine.getAmount());
    showStatus("Amende␣trait e␣avec␣succ s␣!");
}
```

Gestion des Membres (ManageMemberController)

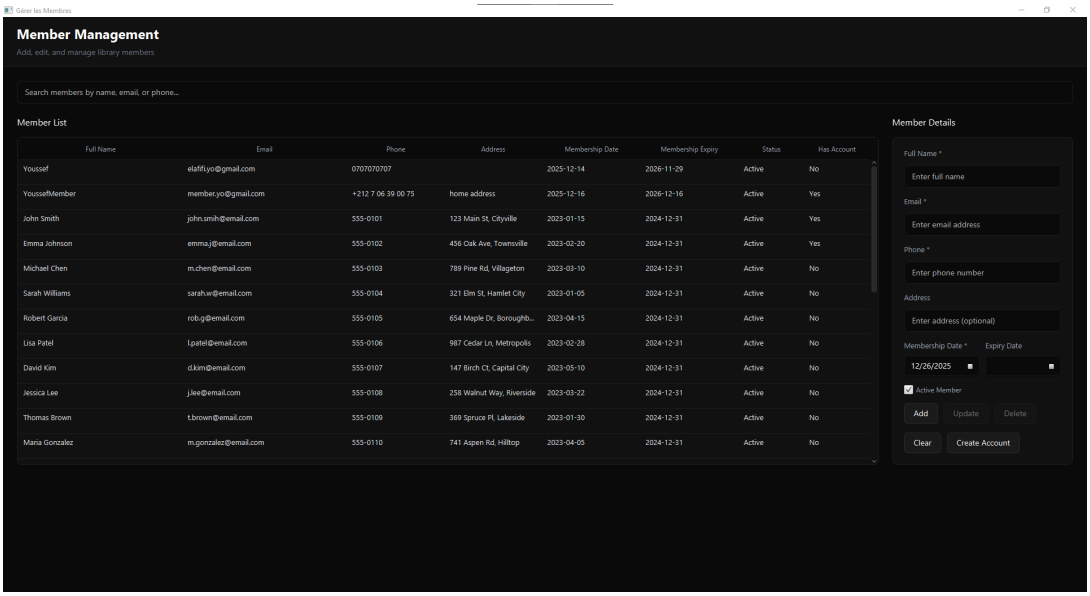

Figure 3.8: Interface de gestion des membres pour les profils et comptes

Fonctionnalités de gestion des membres :

- Créer et modifier les profils des membres
- Création de comptes utilisateurs à partir des membres
- Suivi du statut, expiration et amendes
- Opérations en masse et import/export de données

Listing 3.7: ManageMemberController.java - Opérations sur les membres

```
@FXML
private void handleCreateMember() {
    Member member = createMemberFromForm();
    memberService.createMember(member);
    showStatus("Membre␣cr  ␣avec␣succ s␣!");
}
```

Gestion des Utilisateurs (ManageUserController)

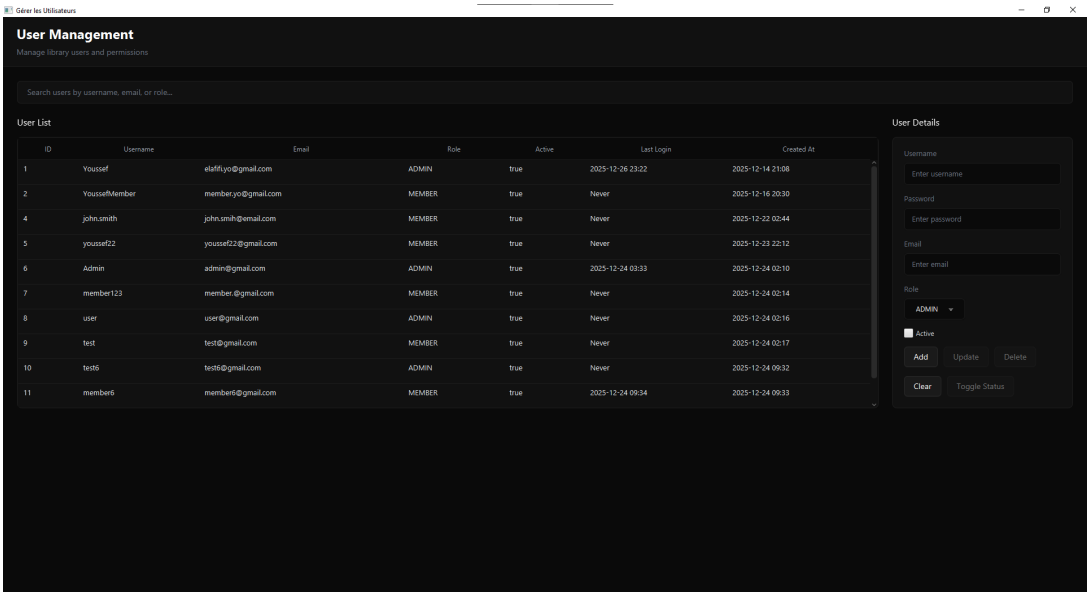

Figure 3.9: Interface de gestion des utilisateurs pour rôles et permissions

Fonctionnalités de gestion des utilisateurs :

- Créer, modifier et supprimer des comptes utilisateurs
- Assigner des rôles (Administrateur, Bibliothécaire, Membre)
- Activer/désactiver les comptes
- Suivi des dernières connexions et paramètres de sécurité

Listing 3.8: ManageUserController.java - Opérations sur les utilisateurs

```
@FXML
private void handleUpdateUserRole() {
    User user = createUserFromForm();
    userService.updateRole(user);
    showStatus("R le␣utilisateur␣mis␣ ␣jour␣avec␣succ s␣!");
}
```

3.5. Architecture des Composants Principaux

3.5.1. Couche Modèle

Modèles côté Membre	**Modèles côté Administrateur**
Livre (accès en lecture seule)	Livre (CRUD complet)
Membre (profil personnel)	Membre (gestion complète des membres)
Emprunt (emprunts personnels)	Emprunt (suivi de tous les emprunts)
Amende (amendes personnelles)	TransactionAmende (toutes les transactions)
Utilisateur (compte personnel)	Utilisateur (tous les comptes)
Activité (activité personnelle)	Activité (activité globale)

Table 3.4: Comparaison de l'accès à la couche modèle

3.5.2. Couche Contrôleur

Contrôleurs côté Membre	**Contrôleurs côté Administrateur**
BookController (navigation/emprunt)	ManageBookController (gestion complète)
MemberController (profil personnel)	ManageMemberController (tous les membres)
DashboardController (personnel)	AdminDashboardController (système)
BorrowController (emprunts personnels)	ManageBorrowController (tous les emprunts)
FineController (amendes personnelles)	ManageFinesController (toutes les amendes)
ProfileController (auto-modification)	ManageUserController (tous les utilisateurs)

Table 3.5: Séparation de la couche contrôleur

3.6. Implémentation des Flux de Travail Parallèles

3.6.1. Flux côté Membre

Processus d'emprunt :

1. Naviguer ou rechercher des livres disponibles

2. Sélectionner un livre à emprunter
3. Le système valide l'éligibilité du membre
4. Le livre est enregistré comme emprunté par le membre
5. Le membre reçoit une confirmation
6. Le livre apparaît dans l'historique personnel

Processus de paiement des amendes :

1. Consulter les amendes personnelles dans le tableau de bord
2. Sélectionner une amende à payer
3. Choisir le mode de paiement
4. Confirmer les détails du paiement
5. Recevoir la confirmation de paiement
6. Le statut de l'amende est mis à jour immédiatement

3.6.2. Flux côté Administrateur

Gestion en masse des livres :

1. Accéder à l'interface de gestion des livres
2. Ajouter/modifier plusieurs livres avec import CSV
3. Assigner catégories et métadonnées
4. Télécharger le contenu numérique (PDF)
5. Mettre à jour les quantités en inventaire
6. Publier les changements dans le catalogue

Administration des membres :

1. Consulter tous les membres
2. Créer de nouveaux profils de membres
3. Lier/délier les comptes utilisateurs
4. Gérer le statut d'adhésion
5. Traiter les mises à jour en masse
6. Générer des rapports sur les membres

3.7. Schéma de Base de Données avec Vues Basées sur les Rôles

3.7.1. Accès aux données pour les Membres

Listing 3.9: Vue SQL pour l'accès aux données des membres

```
-- Les membres ne voient que leurs propres donn es
CREATE VIEW member_borrowings_view AS
SELECT b.* FROM borrows b
WHERE b.member_id = CURRENT_MEMBER_ID();

CREATE VIEW member_fines_view AS
SELECT f.* FROM fine_transactions f
WHERE f.member_id = CURRENT_MEMBER_ID();
```

3.7.2. Accès aux données pour les Administrateurs

Listing 3.10: Accès SQL pour les données administrateur

```
--   L  administrateur   voit toutes les donn es
SELECT * FROM borrows;
SELECT * FROM members;
SELECT * FROM fine_transactions;
SELECT * FROM activities;
```

3.8. Patterns d'Implémentation Technique

3.8.1. Patterns côté Membre

Binding des données personnelles :

Listing 3.11: MemberDashboardController.java

```
@FXML
private void initialize() {
    Member currentMember = sessionManager.getCurrentMember();

    borrowedBooksLabel.textProperty().bind(
        Bindings.createStringBinding(() ->
            String.valueOf(borrowService.countByMember(
               currentMember.getId()))
        )
```

```
    );

    currentFinesLabel.textProperty().bind(
        Bindings.createStringBinding(() ->
            String.format("$%.2f", fineService.getMemberFines(
               currentMember.getId()))
        )
    );
}
```

3.8.2. Patterns côté Administrateur

Agrégation des données système :

Listing 3.12: AdminDashboardController.java

```
@FXML
private void loadSystemStatistics() {
    long totalBooks = bookService.getTotalBooksCount();
    long activeMembers = userService.countActiveMembers();
    long activeBorrowings = borrowService.countActiveBorrowings()
       ;
    double totalFines = borrowService.calculateTotalFines();

    totalBooksLabel.setText(String.valueOf(totalBooks));
    activeMembersLabel.setText(String.valueOf(activeMembers));
    totalFinesLabel.setText(String.format("$%.2f", totalFines));
}
```

3.9. Implémentation de l'Interface Utilisateur

3.9.1. Composants UI côté Membre

Listing 3.13: member$_{d}ashboard.fxml$

```
<VBox xmlns="http://javafx.com/javafx"
      xmlns:fx="http://javafx.com/fxml"
      fx:controller="com.intellilib.controllers.member.
         MemberDashboardController">

    <HBox spacing="20">
```

```
        <VBox>
            <Label text="Mes␣Emprunts"/>
            <TableView fx:id="myBorrowingsTable">
                <columns>
                    <TableColumn text="Titre␣du␣Livre"/>
                    <TableColumn text="Date␣  d  chance  "/>
                    <TableColumn text="Statut"/>
                </columns>
            </TableView>
        </VBox>

        <VBox>
            <Label text="Mes␣Amendes"/>
            <TableView fx:id="myFinesTable">
                <columns>
                    <TableColumn text="Montant"/>
                    <TableColumn text="Date␣  d  chance  "/>
                    <TableColumn text="Statut"/>
                </columns>
            </TableView>
        </VBox>
    </HBox>
</VBox>
```

3.9.2. Composants UI côté Administrateur

Listing 3.14: admin$_d$*ashboard.fxml*

```
<VBox xmlns="http://javafx.com/javafx"
      xmlns:fx="http://javafx.com/fxml"
      fx:controller="com.intellilib.controllers.admin.
         AdminDashboardController">

    <HBox spacing="20">
        <VBox>
            <Label text="Vue␣ d ensemble ␣du␣syst me"/>
            <GridPane>
                <Label text="Total␣Livres:" GridPane.rowIndex="0"
                   />
```

```
                <Label fx:id="totalBooksLabel" GridPane.rowIndex=
                    "0" GridPane.columnIndex="1"/>

                <Label text="Membres␣Actifs:" GridPane.rowIndex="
                    1"/>
                <Label fx:id="activeMembersLabel" GridPane.
                    rowIndex="1" GridPane.columnIndex="1"/>
            </GridPane>
        </VBox>

        <VBox>
            <Button text="G rer␣les␣Livres" onAction="#
                manageBooks"/>
            <Button text="G rer␣les␣Membres" onAction="#
                manageMembers"/>
            <Button text="G rer␣les␣Amendes" onAction="#
                manageFines"/>
        </VBox>
    </HBox>
</VBox>
```

3.10. Sécurité et Contrôle d'Accès

3.10.1. Restrictions d'accès pour les Membres

Listing 3.15: Validation accès Membre

```
@Service
public class MemberAccessService {

    public boolean canAccessBorrow(Long borrowId, Long memberId)
       {
        Borrow borrow = borrowRepository.findById(borrowId).
           orElse(null);
        return borrow != null && borrow.getMember().getId().
           equals(memberId);
    }

    public boolean canAccessFine(Long fineId, Long memberId) {
```

```
        FineTransaction fine = fineRepository.findById(fineId).
            orElse(null);
        return fine != null && fine.getMember().getId().equals(
            memberId);
    }
}
```

3.10.2. Privilèges d'accès Administrateur

Listing 3.16: Validation accès Admin

```
@Service
public class AdminAccessService {

    @PreAuthorize("hasRole('ADMIN')")
    public List<Borrow> getAllBorrows() {
        return borrowRepository.findAll();
    }

    @PreAuthorize("hasRole('ADMIN')")
    public List<Member> getAllMembers() {
        return memberRepository.findAll();
    }

    @PreAuthorize("hasRole('ADMIN')")
    public void deleteMember(Long memberId) {
        memberRepository.deleteById(memberId);
    }
}
```

3.11. Bénéfices de l'Implémentation

3.11.1. Bénéfices Module Membre

Expérience Utilisateur :

- Interface en libre-service intuitive
- Mise à jour en temps réel des données personnelles
- Emprunt et retour simplifiés

- Suivi transparent des amendes

Gains d'Efficacité :

- Réduction de la charge des bibliothécaires
- Disponibilité 24/7 en libre-service
- Accès instantané aux données personnelles
- Notifications automatisées

3.11.2. Bénéfices Module Administrateur

Gestion du Système :

- Supervision complète du système
- Capacités d'opérations en masse
- Analyses et rapports avancés
- Workflows automatisés

Efficacité Opérationnelle :

- Console de gestion centralisée
- Surveillance en temps réel
- Calcul automatique des amendes
- Traçabilité complète des actions

3.12. Résumé de l'Implémentation

Le système IntelliLib implémente avec succès des modules Membre et Administrateur parallèles, répondant à des besoins distincts tout en maintenant une architecture cohérente :

Module Membre : Fonctionnalités en libre-service avec interfaces intuitives pour naviguer, emprunter et gérer les comptes personnels.

Module Administrateur : Administration complète du système avec outils avancés de gestion, analyses et automatisation.

Les deux modules partagent des fondations techniques communes :

- UI moderne basée sur JavaFX avec FXML

- Spring Security pour le contrôle d'accès basé sur les rôles
- Hibernate ORM pour la persistance des données
- Architecture orientée services pour la logique métier

L'approche d'implémentation parallèle garantit une expérience optimale pour les usagers et les administrateurs tout en assurant sécurité, scalabilité et maintenabilité du système.

4. Perspectives Futures

4.1. Améliorations de l'Expérience Membre

Le système IntelliLib peut être enrichi par plusieurs fonctionnalités avancées pour améliorer l'engagement des membres et la personnalisation.

4.1.1. Fonctionnalités de Personnalisation

- **Recommandations de Livres** : Suggestions intelligentes basées sur l'historique d'emprunts et les préférences des membres
- **Analyse de Lecture** : Tableaux de bord visuels montrant les habitudes et statistiques de lecture
- **Notifications Automatisées** : Alertes par e-mail et dans l'application pour les dates d'échéance, amendes et nouvelles arrivées

4.1.2. Fonctionnalités Sociales

- **Avis sur les Livres** : Avis et système de notation soumis par les membres
- **Défis de Lecture** : Éléments de gamification avec réalisations et classements
- **Clubs de Lecture Virtuels** : Plateforme pour discussions communautaires et listes de lecture partagées

4.2. Extensions de la Gestion de la Bibliothèque

Fonctionnalités avancées pour améliorer les opérations de la bibliothèque et la gestion des ressources.

4.2.1. Gestion des Ressources Numériques

- **Support des E-Books** : Intégration des formats EPUB, MOBI et autres

- **Annotation de Documents** : Outils intégrés pour surlignage et prise de notes
- **Support Multimédia** : Gestion des livres audio et ressources vidéo

4.2.2. Efficacité Opérationnelle

- **Intégration QR/Code-Barres** : Scan rapide pour emprunts et retours de livres
- **Opérations en Masse** : Import/export en masse des catalogues via Excel/CSV
- **Application Mobile Complémentaire** : Accès nomade pour membres et personnel

4.3. Avancées Techniques

Améliorations techniques futures pour accroître les capacités du système.

4.3.1. Intégration de l'IA

- **Analyse Prédictive** : Prévision de la demande et optimisation des collections
- **Assistant Chatbot** : Système d'aide basé sur l'IA pour les requêtes des membres
- **Recherche Intelligente** : Traitement du langage naturel pour une découverte améliorée

4.3.2. Fonctionnalités de Scalabilité

- **Déploiement Cloud** : Options d'hébergement pour les grandes institutions
- **API REST** : Capacités d'intégration avec d'autres systèmes
- **Support Multi-Plateforme** : Versions Web et mobile de l'application

Ces améliorations représentent l'évolution d'IntelliLib d'un système de gestion traditionnel vers une plateforme complète de bibliothèque numérique, tirant parti des technologies modernes pour améliorer l'engagement des utilisateurs et l'efficacité opérationnelle.

5. Conclusion

5.1. Résumé des Réalisations du Projet

Le projet IntelliLib a permis de développer un système complet et intelligent de gestion de bibliothèque répondant aux besoins opérationnels modernes tout en offrant une expérience utilisateur intuitive. Les principales réalisations incluent :

- **Implémentation complète MVC** : Séparation claire entre présentation, logique métier et persistance des données, assurant maintenabilité et extensibilité.
- **Modèles de données robustes** : Développement de modèles métier sophistiqués (Livre, Membre, Utilisateur) avec logique intégrée et persistance via Hibernate pour SQLite.
- **Interface utilisateur moderne** : Création d'une interface JavaFX intuitive et réactive avec FXML, offrant navigation fluide et accès basé sur les rôles.
- **Flux de travail essentiels** : Mise en œuvre des opérations critiques : emprunt et retour de livres, inscription des membres, gestion des amendes et gestion des documents numériques.
- **Cadre de sécurité** : Intégration de Spring Security pour l'authentification, l'autorisation et le contrôle d'accès basé sur les rôles.

5.2. Compétences Techniques Démontrées

Ce projet a permis de développer et de démontrer les compétences techniques suivantes :

- **Conception orientée objet** : Utilisation efficace de l'encapsulation, de l'héritage et du polymorphisme en Java
- **Conception de bases de données** : Conception et mise en œuvre de schémas normalisés avec relations appropriées
- **Développement UI/UX** : Création d'interfaces conviviales avec JavaFX et FXML

- **Implémentation de la sécurité** : Intégration de mécanismes d'authentification et d'autorisation
- **Tests logiciels** : Développement de suites de tests complètes pour assurer la qualité
- **Contrôle de version** : Utilisation efficace de Git pour le développement collaboratif et la gestion des versions

5.3. Potentiel Futur et Impact

IntelliLib établit une base solide pouvant évoluer dans plusieurs directions :

- **Valeur Éducative** : Projet comme étude de cas pour le développement d'applications Java modernes, démontrant l'implémentation réelle de design patterns et frameworks.
- **Impact Communautaire** : Fournir aux bibliothèques une solution de gestion digitale efficace, améliorant l'accès aux ressources et les services.
- **Fondation Technologique** : Architecture modulaire permettant une intégration future de technologies comme les recommandations IA, applications mobiles et services cloud.
- **Contribution Open Source** : Structure et implémentation du projet servant de référence pour d'autres développeurs travaillant sur des solutions similaires.

5.4. Réflexions Finales

Le développement d'IntelliLib constitue une réalisation majeure dans l'application des concepts théoriques de l'ingénierie logicielle à un problème concret. Le projet relie avec succès les besoins traditionnels de gestion de bibliothèque aux solutions technologiques modernes, démontrant comment le logiciel peut améliorer l'efficacité organisationnelle et l'expérience utilisateur.

La conception modulaire, le jeu de fonctionnalités complet et l'accent mis sur l'expérience utilisateur font d'IntelliLib une solution viable pour les bibliothèques souhaitant moderniser leurs opérations. Le projet illustre également l'importance des bonnes pratiques de développement logiciel : planification rigoureuse, mise en œuvre cohérente et tests approfondis, pour aboutir à un système robuste, maintenable et extensible.

À mesure que les bibliothèques évoluent à l'ère numérique, des systèmes comme IntelliLib joueront un rôle crucial pour garantir l'accessibilité, l'efficacité et la pertinence

des ressources pour leurs utilisateurs. Le projet atteint non seulement ses objectifs immédiats mais pose également les bases pour de futures innovations dans la technologie des bibliothèques.