Rapport de Projet : Smart Library

Réalisé par : Khadija El Merahy
Encadré par : Pr. H. Hrimech

Mai 2025

Contents

1 Objectifs du Projet

Le projet **SmartLib** vise à développer une application Java complète pour la gestion intelligente d'une bibliothèque universitaire. Les objectifs principaux sont :

- Gérer efficacement les livres, les étudiants et les emprunts.
- Automatiser le calcul des pénalités de retard (après 7 jours).
- Fournir des statistiques d'usage (livres les plus empruntés, retardataires fréquents).
- Offrir une interface utilisateur moderne et intuitive avec JavaFX.
- Assurer la persistance des données via Hibernate et MySQL.
- Respecter une architecture MVC claire et modulaire.

2 Logiciels Utilisés

Les logiciels suivants ont été utilisés pour le développement du projet :

- **NetBeans** : Environnement de développement intégré (IDE) pour le codage, le débogage et la gestion du projet.
- **MySQL Workbench** : Outil de conception et de gestion de la base de données MySQL.
- **SceneBuilder** : Logiciel pour la création visuelle des interfaces graphiques JavaFX (fichiers FXML).

3 Introduction

Le projet **SmartLib** est une application Java conçue pour gérer une bibliothèque universitaire. Elle permet la gestion des livres, des étudiants, des emprunts, des retours, des pénalités de retard, et des statistiques d'usage. L'application utilise JavaFX pour l'interface utilisateur, Hibernate avec MySQL pour la persistance des données, et suit une architecture MVC. Ce rapport détaille l'architecture, les diagrammes UML, les captures d'écran de l'interface, l'état des tests, et intègre le code source des fichiers clés.

4 Architecture du Projet

4.1 Structure des Dossiers

La structure du projet est organisée comme suit :

- `controller/` : Contrôleurs JavaFX (`AdminController.java`, `BorrowInterfaceController.jav`
  `LoginController.java`, `StudentDashboardController.java`).
- `Smartlib/` : Classe principale (`Smartlib.java`). Fichier CSS (`styles.css`).

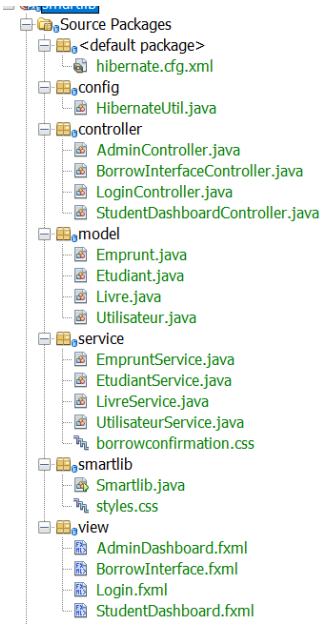


Figure 1: Structure des dossiers du projet SmartLib

- `model/` : Entités (`Emprunt.java`, `Etudiant.java`, `Livre.java`, `Utilisateur.java`).
- `default package/` : Configuration Hibernate (`hibernate.cfg.xml`, `HibernateUtil.java`).
- `service/` : Logique métier (`EmpruntService.java`, `EtudiantService.java`, `LivreService.java`, `UtilisateurService.java`).
- `view/` : Fichiers FXML (`AdminDashboard.fxml`, `BorrowInterface.fxml`, `Login.fxml`, `StudentDashboard.fxml`).

4.2 Architecture MVC

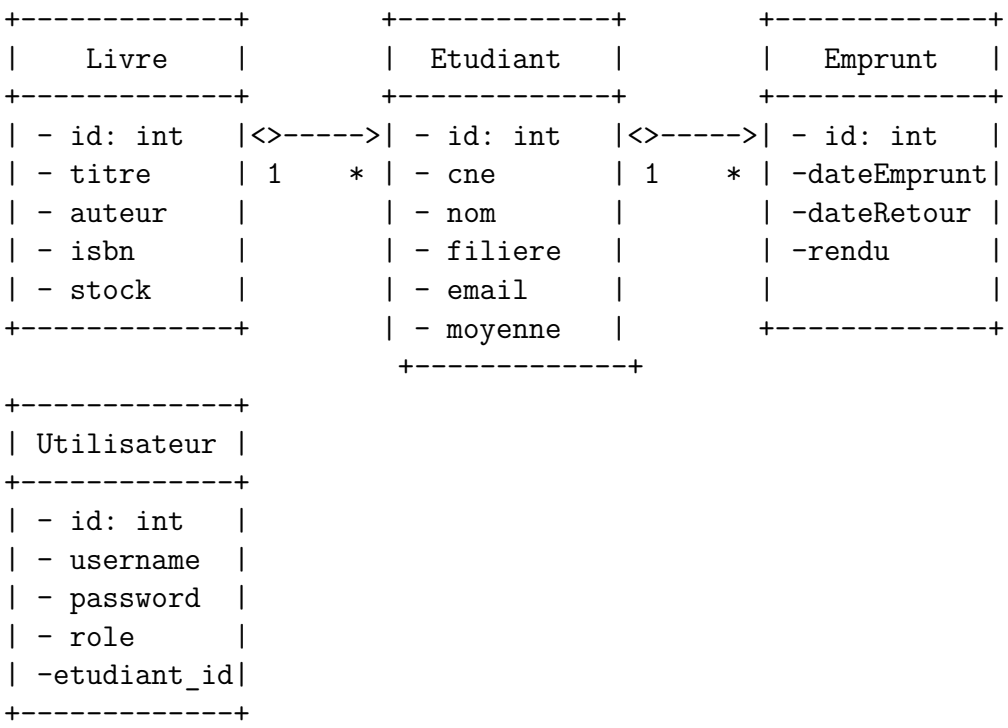
L'application suit le modèle MVC :

- **Model** : Les entités dans `model/` définissent les données.
- **View** : Les interfaces dans `view/` sont créées avec FXML et stylisées via CSS.
- **Controller** : Les contrôleurs dans `controller/` gèrent les interactions.

La couche `service` encapsule la logique métier, et `dao` gère l'accès à la base de données via Hibernate.

5 Diagramme UML des Classes

Voici un extrait du diagramme UML basé sur les entités :



Les relations incluent :

- `Emprunt` est lié à `Livre` et `Etudiant` via `@ManyToOne`.
- `Utilisateur` est une entité potentielle pour l'authentification.

6 Interface Utilisateur

6.1 Description

L'interface utilise JavaFX avec plusieurs vues :

- `Login.fxml` : Page de connexion avec un thème personnalisé.
- `AdminDashboard.fxml` : Interface admin avec gestion des livres, étudiants, et statistiques.
- `BorrowInterface.fxml` : Gestion des emprunts et retours.
- `StudentDashboard.fxml` : Tableau de bord des étudiants.

6.2 Captures d'Écran

Page de Connexion : Dégradé linéaire avec champs et bouton stylisés via `login.css`.
Admin Dashboard : Onglets pour la gestion et les statistiques avec `smartlib.css`.
Borrow Interface : Tableaux d'emprunts avec styles de `borrowconfirmation.css`.

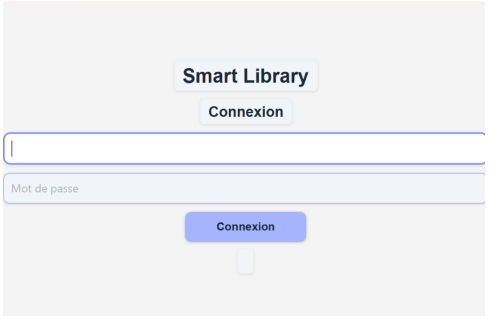


Figure 2: Login.fxml

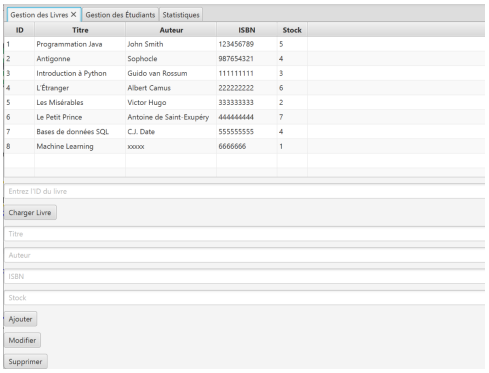


Figure 3: Gestion des livres

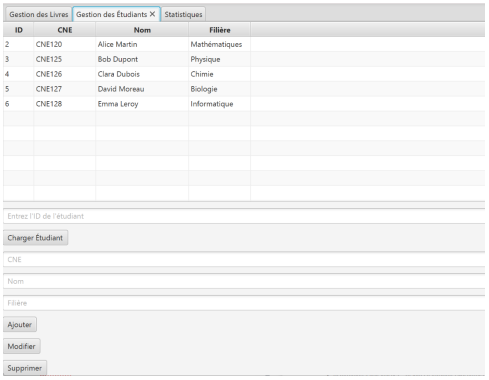


Figure 4: Gestion des étudiants

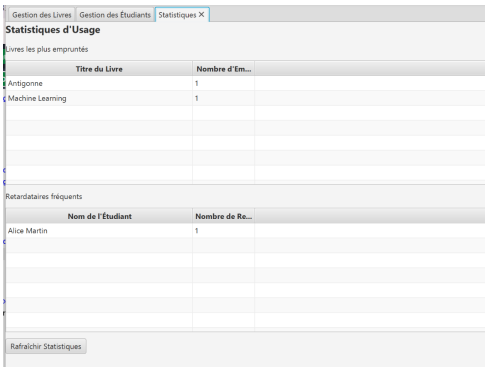


Figure 5: Statistiques

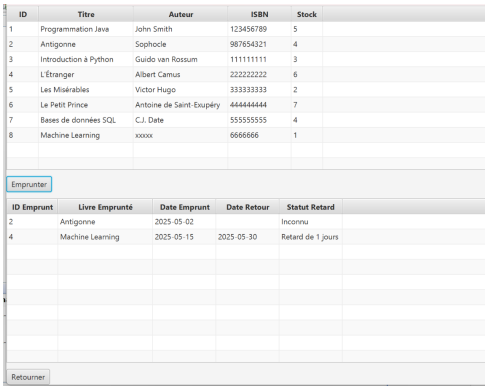


Figure 6: Interface étudiant

7 État des Tests

7.1 Tests Fonctionnels

- **CRUD Livres et Étudiants** : Fonctionne via `AdminController`.
- **Gestion des Emprunts** : Emprunt et retour via `BorrowInterfaceController`, avec correction des retards.
- `StudentDashboardController` : Affichage des données étudiant.
- **Statistiques** : Fonctionnelles dans `AdminDashboard.fxml`.

7.2 Problèmes Rencontrés

- Retardataires non affichés : Résolu en ajustant `EmpruntService`.
- CSS incohérent : Ajusté dans `login.css` et `smartlib.css`.

7.3 Tests Unitaires

Non implémentés ; recommandation d'utiliser JUnit pour tester les services.

8 Code Source

8.1 Entités

Livre.java :

```
package model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
```

```java
@Table(name = "livre")
public class Livre {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String titre;
    private String auteur;
    private String isbn;
    private int stock;

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getTitre() { return titre; }
    public void setTitre(String titre) { this.titre = titre; }
    public String getAuteur() { return auteur; }
    public void setAuteur(String auteur) { this.auteur = auteur; }
    public String getIsbn() { return isbn; }
    public void setIsbn(String isbn) { this.isbn = isbn; }
    public int getStock() { return stock; }
    public void setStock(int stock) { this.stock = stock; }
}
```

Etudiant.java :

```java
package model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "etudiant")
public class Etudiant {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String cne;
    private String nom;
    private String filiere;
    private String email;
    private double moyenne;

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getCne() { return cne; }
    public void setCne(String cne) { this.cne = cne; }
    public String getNom() { return nom; }
```

```
    public void setNom(String nom) { this.nom = nom; }
    public String getFiliere() { return filiere; }
    public void setFiliere(String filiere) { this.filiere =
        filiere; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public double getMoyenne() { return moyenne; }
    public void setMoyenne(double moyenne) { this.moyenne =
        moyenne; }
}
```

Emprunt.java :

```
package model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import java.time.LocalDate;

@Entity
@Table(name = "emprunt")
public class Emprunt {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private LocalDate dateEmprunt;
    private LocalDate dateRetour;
    private boolean rendu;
    @ManyToOne
    private Livre livre;
    @ManyToOne
    private Etudiant etudiant;

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public LocalDate getDateEmprunt() { return dateEmprunt; }
    public void setDateEmprunt(LocalDate dateEmprunt) { this.
        dateEmprunt = dateEmprunt; }
    public LocalDate getDateRetour() { return dateRetour; }
    public void setDateRetour(LocalDate dateRetour) { this.
        dateRetour = dateRetour; }
    public boolean isRendu() { return rendu; }
    public void setRendu(boolean rendu) { this.rendu = rendu; }
    public Livre getLivre() { return livre; }
    public void setLivre(Livre livre) { this.livre = livre; }
    public Etudiant getEtudiant() { return etudiant; }
    public void setEtudiant(Etudiant etudiant) { this.etudiant =
```

```
        etudiant; }
}
```

8.2 Services

EmpruntService.java (extrait) :

```
package service;

import model.Emprunt;
import model.Etudiant;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Query;
import config.HibernateUtil;
import java.util.Date;
import java.util.List;

public class EmpruntService {
    @SuppressWarnings("unchecked")
    public List<Object[]> getFrequentDelayers() {
        SessionFactory sessionFactory = HibernateUtil.
           getSessionFactory();
        Session session = null;
        try {
            session = sessionFactory.openSession();
            String hql = "SELECT e.etudiant, COUNT(e) FROM
               Emprunt e WHERE e.dateRetour < :currentDate GROUP
               BY e.etudiant ORDER BY COUNT(e) DESC";
            Query query = session.createQuery(hql);
            query.setParameter("currentDate", new Date());
            return query.list();
        } catch (Exception e) {
            System.err.println("Erreur dans getFrequentDelayers :
                " + e.getMessage());
            return null;
        } finally {
            if (session != null && session.isOpen()) {
                session.close();
            }
        }
    }
}
```

8.3 Contrôleurs

LoginController.java (extrait) :

```
package controller;

import javafx.fxml.FXML;
```

```
import javafx.scene.control.TextField;
import javafx.scene.control.Button;
import javafx.scene.control.Label;

public class LoginController {
    @FXML private TextField usernameField;
    @FXML private TextField passwordField;
    @FXML private Button loginButton;
    @FXML private Label errorLabel;

    @FXML
    private void handleLogin() {
        String username = usernameField.getText();
        String password = passwordField.getText();
        if ("admin".equals(username) && "password".equals(
           password)) {
            errorLabel.setText("");
        } else {
            errorLabel.setText("Nom d'utilisateur ou mot de passe
                incorrect.");
        }
    }
}
```

8.4 Interfaces FXML

Login.fxml (extrait) :

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.VBox?>

<VBox fx:id="mainFxmlClass" alignment="CENTER" spacing="10" xmlns
   ="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml"
   fx:controller="controller.LoginController">
    <stylesheets>
        <String fx:value="/smartlib/login.css" />
    </stylesheets>
    <Label text="Smart Library" style="-fx-font-size: 24px; -fx-
       font-weight: bold;" />
    <Label text="Connexion" style="-fx-font-size: 18px;" />
    <TextField fx:id="usernameField" promptText="Nom d'
       utilisateur" />
    <TextField fx:id="passwordField" promptText="Mot de passe" />
    <Button fx:id="loginButton" text="Connexion" onAction="#
       handleLogin" />
    <Label fx:id="errorLabel" text="" style="-fx-text-fill: red;"
        />
```

```
</VBox>
```

9 Conclusion

SmartLib offre une gestion efficace des bibliothèques universitaires. Les modules principaux sont opérationnels, avec une interface moderne. Les évolutions futures incluent l'authentification et les notifications.

10 Annexes

10.1 Script SQL

```sql


-- Cr er la base de donn es
CREATE DATABASE SmartLib;
USE SmartLib;

-- Cr er la table etudiant (sans d pendance    utilisateur)
CREATE TABLE etudiant (
    id INT AUTO_INCREMENT PRIMARY KEY,
    cne VARCHAR(20) NOT NULL UNIQUE,
    nom VARCHAR(100) NOT NULL,
    filiere VARCHAR(50),
    email VARCHAR(100),
    moyenne DOUBLE
);

-- Cr er la table utilisateur (avec r f rence    etudiant)
CREATE TABLE utilisateur (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    role ENUM('ADMIN', 'ETUDIANT') NOT NULL,
    etudiant_id INT,
    CONSTRAINT fk_utilisateur_etudiant FOREIGN KEY (etudiant_id)
       REFERENCES etudiant(id) ON DELETE SET NULL
);

-- Cr er la table livre
CREATE TABLE livre (
    id INT AUTO_INCREMENT PRIMARY KEY,
    titre VARCHAR(100) NOT NULL,
    auteur VARCHAR(100),
    genre VARCHAR(50),
    isbn VARCHAR(20) NOT NULL UNIQUE,
    stock INT NOT NULL
);
```

```sql

-- Cr er la table emprunt (avec cl s    trangres    et ON DELETE CASCADE)
CREATE TABLE emprunt (
    id INT AUTO_INCREMENT PRIMARY KEY,
    date_emprunt DATE NOT NULL,
    date_retour DATE,
    rendu BOOLEAN DEFAULT FALSE,
    livre_id INT,
    etudiant_id INT,
    CONSTRAINT fk_emprunt_livre FOREIGN KEY (livre_id) REFERENCES livre(id) ON DELETE CASCADE,
    CONSTRAINT fk_emprunt_etudiant FOREIGN KEY (etudiant_id) REFERENCES etudiant(id) ON DELETE CASCADE
);
```

10.2 Fichier .jar

`SmartLib.jar` est fourni avec le projet.

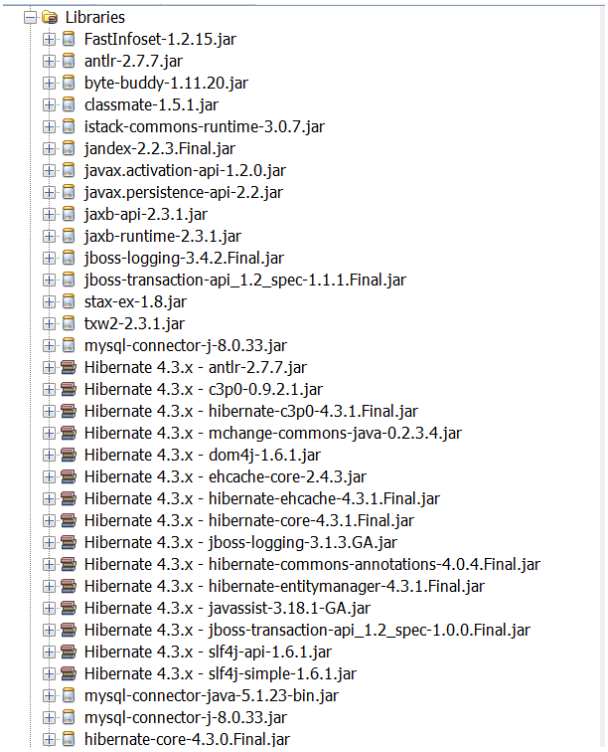


Figure 7: Librairie du projet