RAPPORT DE PROJET

Système de Gestion de Bibliothèque ISAE

Université : ISSAE-Cnam Liban

Cours : Programmation Avancée (NFP121)

Sujet : Implémentation d'un Système de Gestion de Bibliothèque utilisant les Patrons de Conception

Date : Janvier 2026

Préparé par : [Mahdi Ahmad Haidar]

NoDossier:[1334Baal]

Table des Matières

---

1. Introduction Générale

1.1 Contexte du Projet

Dans le paysage éducatif moderne, la gestion efficace des ressources numériques est primordiale pour les institutions académiques. L'ISSAE-Cnam Liban nécessite une solution logicielle robuste pour gérer sa bibliothèque multimédia croissante. Cette bibliothèque sert de dépôt central pour diverses ressources éducatives, allant des cours vidéo et documents PDF aux quiz interactifs.

Ce projet, intitulé "Système de Gestion de Bibliothèque Multimédia", a été développé dans le cadre du cours de Programmation Avancée (NFP121). Il vise à simuler un scénario réel où le logiciel doit être non seulement fonctionnel, mais aussi maintenable, modulaire et facilement extensible pour répondre aux besoins futurs sans nécessiter une refonte majeure du code.

1.2 Problématique

Le défi principal souligné dans le cahier des charges est l'Extensibilité. Un piège courant en génie logiciel est le "code rigide" — des systèmes qui nécessitent des modifications importantes des classes existantes chaque fois qu'une nouvelle fonctionnalité est ajoutée. Les questions architecturales clés abordées dans ce rapport incluent :

- **Comment ajouter un nouveau type de média (par exemple, "Jeu Éducatif") sans modifier la logique centrale du système ?**
- **Comment notifier immédiatement les étudiants connectés lorsqu'un contenu pertinent est ajouté, sans créer de dépendances fortes ?**

- **Comment assurer la cohérence des données entre plusieurs vues (Admin vs Étudiant) ?**

1.3 Objectifs Pédagogiques

Le processus de développement s'est concentré sur la maîtrise des Principes SOLID et l'application stricte des Patrons de Conception (Design Patterns) GoF. L'objectif était de passer du simple codage à une véritable Ingénierie Logicielle, garantissant un système à couplage faible et à forte cohésion.

---

2. Analyse et Besoins du Système

2.1 Besoins Fonctionnels

Le système est conçu pour deux acteurs principaux :

A. L'Administrateur (Admin)

- **Authentification : Mécanisme de connexion sécurisé.**
- **Gestion des Ressources : Capacité de créer, catégoriser et sauvegarder des ressources médias.**
- **Persistance des Données : Sauvegarde de l'état de la bibliothèque dans un fichier (catalogue.txt) et rechargement au démarrage.**
- **Reporting : Consultation des statistiques de base concernant le contenu de la bibliothèque.**

B. L'Étudiant

- **Authentification : Connexion via les identifiants stockés dans le système (chargés depuis XML).**
- **Vue Personnalisée : Visualisation des médias filtrés spécifiquement par leurs matières inscrites.**
- **Mises à jour en Temps Réel : Réception de notifications automatiques lorsqu'un nouveau contenu pertinent est ajouté.**

2.2 Besoins Non-Fonctionnels

- **Extensibilité : L'architecture doit supporter de manière transparente de nouveaux types de médias et formats d'exportation.**
- **Ergonomie (Usability) : Une interface graphique (GUI) Swing intuitive.**
- **Fiabilité : Gestion robuste des erreurs de fichiers (fichiers manquants, données corrompues).**

---

3. Architecture Logicielle et Conception

3.1 Architecture Globale

L'application adopte une variation du modèle MVC (Modèle-Vue-Contrôleur) pour séparer les préoccupations :

- **Modèle : Représente les données et la logique métier (Media, User, LibrarySystem). Ces classes sont indépendantes de l'interface utilisateur.**
- **Vue : Les composants Swing (AdminDashboard, StudentDashboard, LoginFrame) responsables de l'affichage des données.**
- **Contrôleur : La logique reliant le Modèle et la Vue, intégrée dans les écouteurs d'événements (Listeners) et les classes de service.**

3.2 Description du Diagramme de Classes UML

(Note : uml draw.io) **Le diagramme illustre la hiérarchie d'héritage de la classe abstraite Media et de ses sous-classes concrètes (Video, Document, Quiz). Il dépeint également le singleton LibrarySystem agissant comme le centre névralgique, et l'interface Observer implémentée par le StudentDashboard.**

---

4. Implémentation des Patrons de Conception (Analyse Cœur)

Cette section détaille les patrons de conception spécifiques sélectionnés pour résoudre les défis architecturaux posés dans les exigences du projet (Page 6 du PDF).

4.1 Patron Singleton : Contrôle Centralisé

- **Contexte : L'application nécessite une "Source Unique de Vérité" pour le catalogue de la bibliothèque. De multiples instances de LibrarySystem entraîneraient une incohérence des données (ex: l'Admin ajoute une vidéo à une instance, mais l'Étudiant lit depuis une autre).**
- **Implémentation : Nous avons restreint l'instanciation de la classe LibrarySystem en utilisant un constructeur privé et une instance statique.**

Java

```
public static synchronized LibrarySystem getInstance() {
   if (instance == null) instance = new LibrarySystem();
   return instance;
}
```

- **Justification : Cela garantit que l'écran de connexion, le tableau de bord Admin et le tableau de bord Étudiant interagissent tous avec exactement le même ensemble de données en mémoire.**

4.2 Patron Factory Method : Assurer l'Extensibilité

- **Exigence n°1 : *"Comment concevoir un système de création qui permet d'ajouter de nouveaux types... sans modifier le code existant ?"***
- **Solution : Nous avons délégué la création d'objets à une classe MediaFactory. Le tableau de bord Admin n'instancie pas new Video() directement ; il demande un objet média à la Factory basé sur un identifiant de chaîne ("Video", "Quiz").**

- **Avantage : Si un nouveau type (par exemple, "Jeu") est requis à l'avenir, nous modifions uniquement la logique de la Factory. Le code client (Admin Dashboard) reste inchangé, satisfaisant ainsi le Principe Ouvert/Fermé (OCP).**

4.3 Patron Observer : Mises à jour Réactives

- **Exigence n°4 :** ***"Comment notifier automatiquement les différents utilisateurs... sans créer de dépendances rigides ?"***
- **Solution :**
  - **Sujet (Subject) : LibrarySystem (Détenteur des données).**
  - **Observateur (Observer) : StudentDashboard (Écouteur).**
  - **Action : Lorsque addMedia() est exécuté, notifyObservers() est déclenché, rafraîchissant automatiquement l'interface utilisateur de tous les étudiants actifs.**
- **Justification : Cela crée un modèle d'abonnement évolutif. Le système diffuse les mises à jour "à l'aveugle", découplant la source de données de la logique d'affichage.**

4.4 Stratégie de Filtrage (Approche Fonctionnelle)

- **Exigence n°2 : Critères de filtrage extensibles.**
- **Solution : Nous avons utilisé l'API Stream moderne de Java au sein du LibrarySystem.**

Java

```
public List<Media> getMediaByMatiere(String code) {
   return catalogue.stream()
         .filter(m -> m.getCodesMatieres().contains(code))
         .collect(Collectors.toList());
}
```

- **Justification : Cette approche fonctionnelle est plus propre et plus flexible que les boucles traditionnelles, permettant des critères de filtrage complexes et chaînés à l'avenir.**

---

5. Détails Techniques d'Implémentation

5.1 Persistance des Données

Les données sont stockées dans un fichier texte (catalogue.txt) en utilisant un format personnalisé basé sur des délimiteurs pour éviter la surcharge d'une base de données complexe.

- **Format : TYPE;ID;TITRE;AUTEUR;ANNEE;DESCRIPTION;ATTRIBUT_SPECIFIQUE;CODES**
- **Logique d'Analyse : Le système lit le fichier ligne par ligne. Le premier jeton (TYPE) indique à la Factory quel objet créer.**

5.2 Intégrité des Données

- **Génération Auto-ID : Pour éviter les conflits d'ID (comme l'erreur ID 0 exists), le système calcule le prochain ID disponible (max(currentIds) + 1) lors du chargement.**
- **Déduplication : Avant d'ajouter une ressource, le système vérifie si un média avec le même titre, auteur et type existe déjà.**

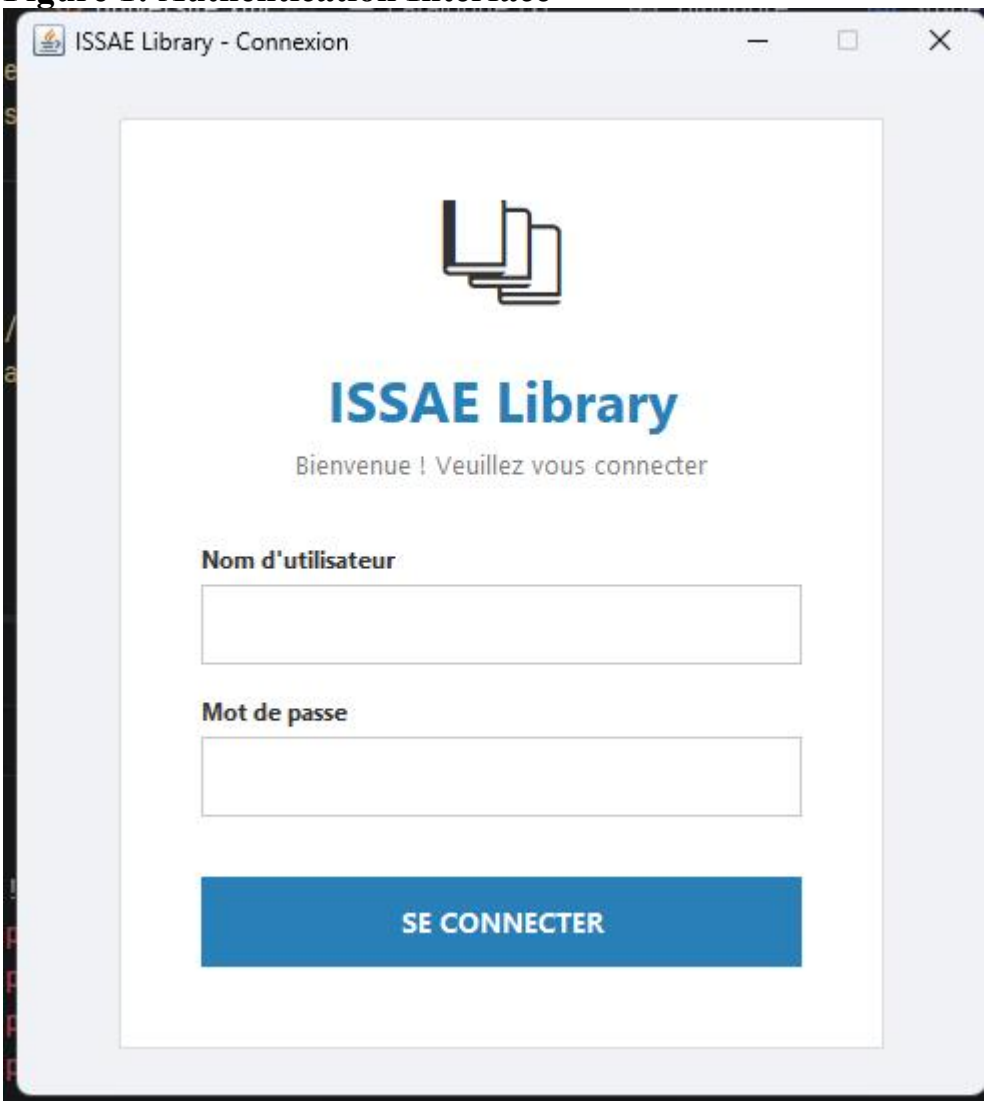
5.3 Interface Graphique (GUI):

L'interface utilisateur a été développée en utilisant la bibliothèque Java Swing, offrant une interaction fluide et ergonomique :

- **Structure Modulaire : Les fenêtres (AdminDashboard et StudentDashboard) sont indépendantes de la logique métier, respectant le principe de séparation des préoccupations.**

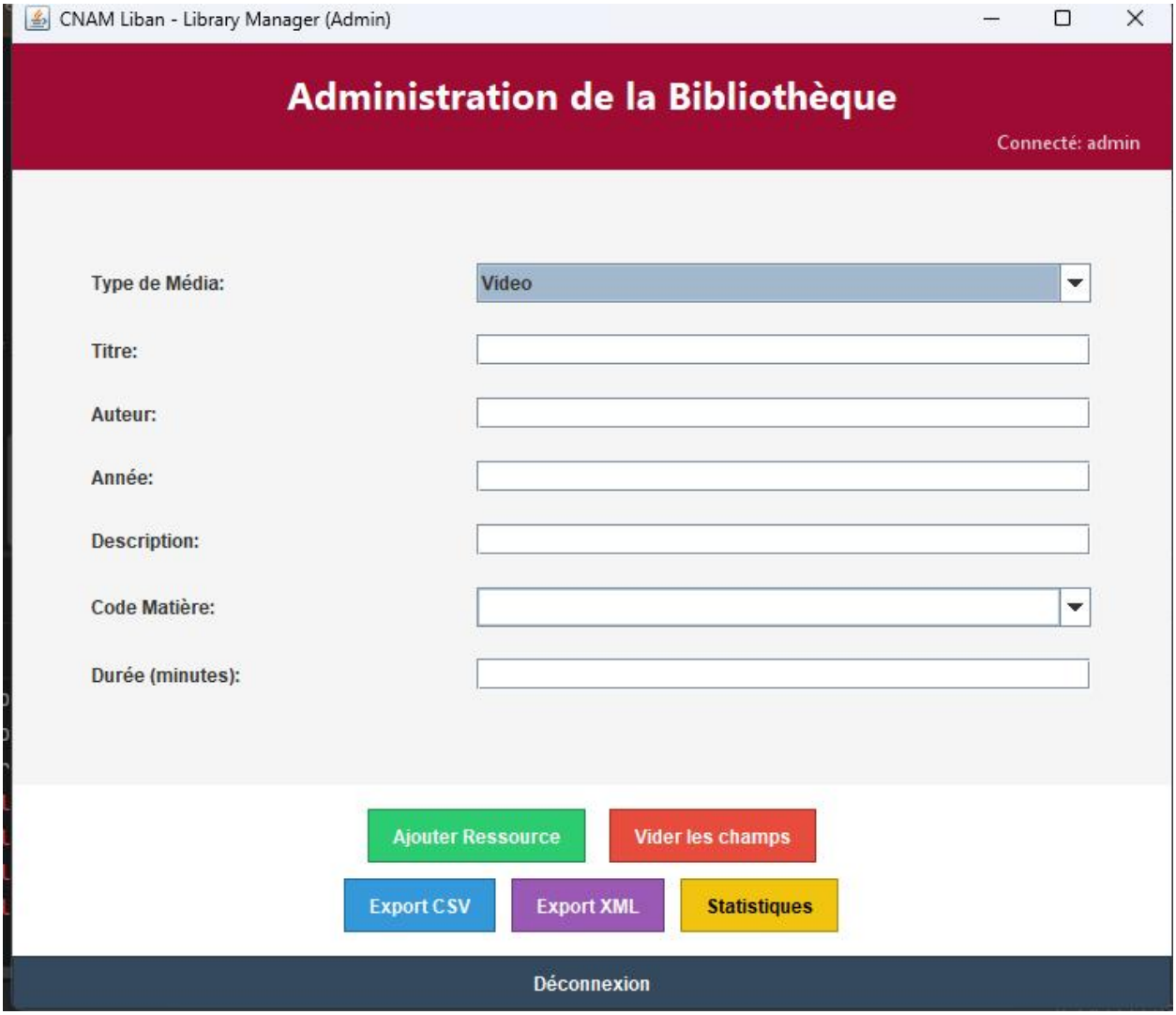
- **Tableaux Dynamiques : Le JTable du tableau de bord étudiant n'est pas statique ; il est connecté directement aux données filtrées du LibrarySystem et se met à jour automatiquement grâce au patron Observer.**
- **Disposition (Layout) : Nous avons utilisé GridBagLayout pour aligner précisément les formulaires d'administration, et BorderLayout pour structurer la vue étudiante (Liste latérale et Tableau central).**
- **Interactivité : Des ActionListeners sont utilisés pour gérer les clics sur les boutons et la sélection des matières, déclenchant le filtrage instantané des ressources.**

5.4 Application Screenshots

- **Figure 1: Authentication Interface**
- 

- *Description : Point d'entrée sécurisé redirigeant vers le tableau de bord approprié selon les identifiants (Admin ou Étudiant).*

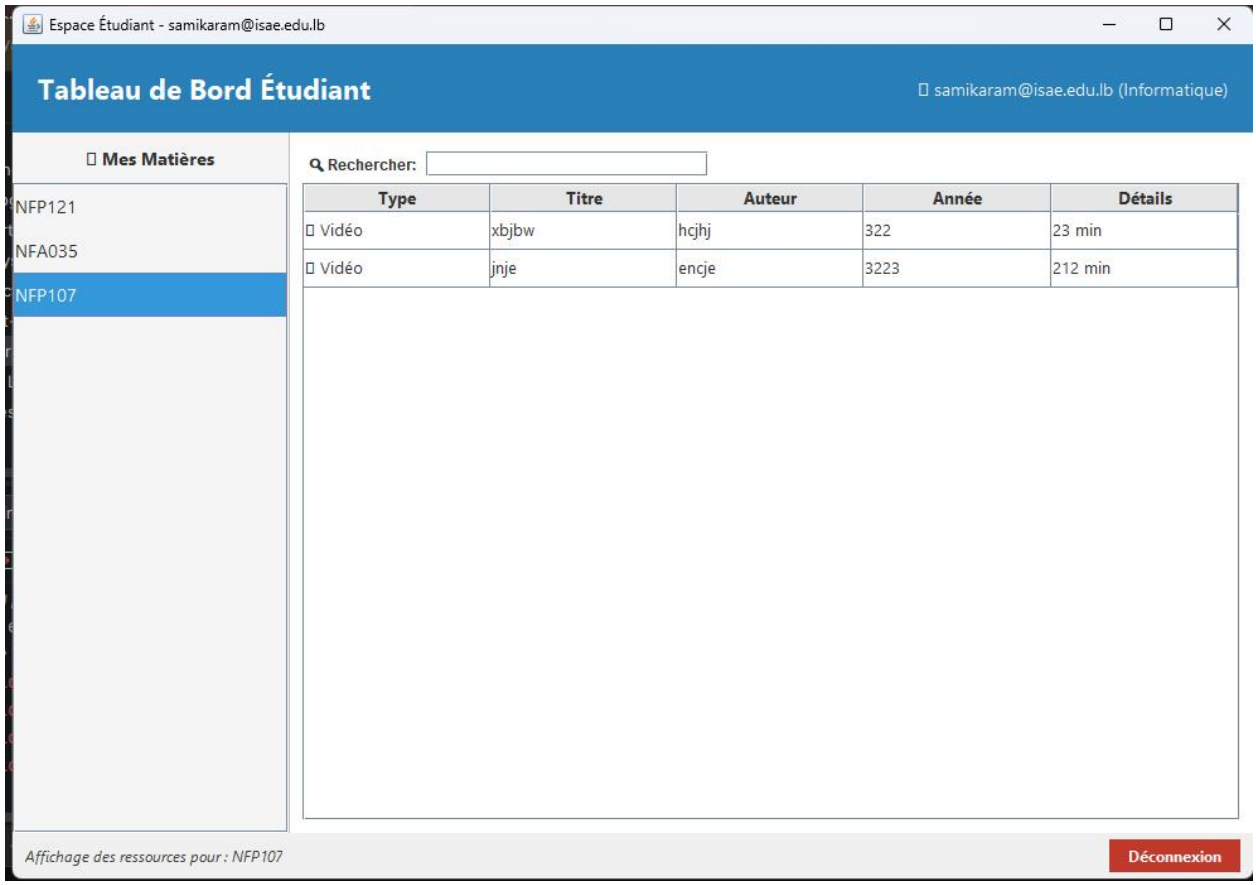
- **Figure 2: Administrator Dashboard**
  (



Description : Le panneau d'administration permet d'ajouter de nouvelles ressources via le patron Factory et d'exporter les données.

-

- **Figure 3: Student Dashboard**



Description : Vue étudiante affichant les médias filtrés par matière. Le tableau se met à jour en temps réel grâce au patron Observer.

---

6. Défis et Améliorations Futures

6.1 Défis Rencontrés

- **Analyse des Données (Parsing) : La gestion de différents types de données (Entiers pour la durée des vidéos vs Chaînes pour les niveaux de Quiz) a nécessité une logique d'analyse minutieuse dans la méthode loadData.**

- **Thread Swing : Assurer que les mises à jour de l'interface utilisateur (via Observer) se produisent sur le thread de répartition des événements (EDT) pour éviter le gel de l'application.**

6.2 Extensions Proposées

Les exigences du projet mentionnent l'exportation vers différents formats (XML, CSV). Bien qu'une logique d'exportation de base existe, satisfaire pleinement l'exigence d'exportation extensible impliquerait la mise en œuvre du Patron Strategy :

- **Interface : ExportStrategy avec une méthode void export(List<Media> data).**
- **Classes Concrètes : XmlExportStrategy, CsvExportStrategy.**
- **Contexte : L'Admin sélectionnerait une stratégie au moment de l'exécution, permettant d'ajouter de nouveaux formats (par exemple, PDF) en tant que nouvelles classes sans modifier le code de l'Admin.**

---

7. Conclusion

Le projet "Système de Gestion de Bibliothèque Multimédia" a servi d'exercice complet en architecture logicielle. En appliquant rigoureusement les patrons Singleton, Factory et Observer, nous avons obtenu un système robuste face au changement.

La séparation des préoccupations garantit que l'ajout d'un nouveau type de média le semestre prochain prendrait quelques minutes, et non des heures.

L'utilisation du patron Observer offre une expérience utilisateur moderne et réactive. Ce projet démontre avec succès la puissance des Patrons de Conception pour résoudre des problèmes courants d'ingénierie logicielle, aboutissant à une application modulaire, maintenable et évolutive.