EMSI - École Marocaine des Sciences de l'Ingénieur

Casablanca

Rapport de Projet

Java Avancé

Application de Gestion des Ressources Pédagogiques LOM

Réalisé par :

ILYAS AIT MAINA
4IIR - Génie Informatique

Encadré par :

Pr. ABDERRAHIM LARHLIMI

Année Universitaire : 2025-2026

Table des matières

Chapitre 1

Introduction Générale

1.1 Contexte du Projet

Dans le domaine de l'éducation numérique, la gestion des ressources pédagogiques représente un enjeu majeur pour les institutions d'enseignement. Le standard **LOM (Learning Object Metadata)** 1.0, développé par l'IEEE, fournit un cadre normalisé pour décrire les objets d'apprentissage selon neuf catégories de métadonnées.

Ce projet s'inscrit dans le cadre du module de **Java Avancé** de la formation 4IIR à l'EMSI. Il vise à mettre en pratique les concepts avancés de programmation Java : la programmation orientée objet, les collections, l'API Streams, le multi-threading, la gestion des exceptions et l'ORM Hibernate.

1.2 Problématique

Les institutions éducatives font face à plusieurs défis dans la gestion de leurs ressources pédagogiques :

- **Absence de standardisation** : Les métadonnées sont souvent incomplètes ou non structurées.
- **Difficulté de recherche** : Sans indexation appropriée, retrouver une ressource spécifique devient laborieux.
- **Manque de collaboration** : Les systèmes existants ne permettent pas le partage et la notation des ressources.
- **Interopérabilité limitée** : L'export vers d'autres systèmes est complexe sans format standardisé.

1.3 Objectifs du Projet

L'application développée vise à répondre à ces problématiques en offrant les fonctionnalités suivantes :

1. **Gestion CRUD complète** des ressources pédagogiques selon le standard LOM.
2. **Système d'authentification** avec deux rôles : Administrateur et Utilisateur.
3. **Recherche avancée** multi-critères (langue, difficulté, type, mots-clés).

4. **Système de favoris** pour organiser les ressources préférées.
5. **Système de notation** avec commentaires (1 à 5 étoiles).
6. **Export XML** conforme au standard LOM pour l'interopérabilité.
7. **Statistiques et reporting** sur l'utilisation des ressources.

Chapitre 2

Analyse et Conception

2.1 Spécification des Besoins

2.1.1 Besoins Fonctionnels

ID	Description du Besoin
BF01	Le système doit permettre l'authentification des utilisateurs.
BF02	L'administrateur doit pouvoir créer, modifier et supprimer des ressources.
BF03	L'utilisateur doit pouvoir consulter et rechercher des ressources.
BF04	L'utilisateur doit pouvoir ajouter des ressources à ses favoris.
BF05	L'utilisateur doit pouvoir noter et commenter les ressources.
BF06	Le système doit permettre l'export XML des métadonnées LOM.
BF07	L'administrateur doit pouvoir gérer les tags et catégories.
BF08	Le système doit afficher des statistiques sur les ressources.

Table 2.1 – Besoins Fonctionnels

2.1.2 Besoins Non-Fonctionnels

ID	Description du Besoin
BNF01	Les mots de passe doivent être stockés de manière sécurisée.
BNF02	L'interface doit être intuitive et ergonomique.
BNF03	Le temps de réponse des requêtes doit être inférieur à 2 secondes.
BNF04	L'application doit supporter les accès concurrents (multi-threading).
BNF05	Le code doit respecter les principes SOLID et les design patterns.
BNF06	L'application doit être portable (Windows, Linux, macOS).

Table 2.2 – Besoins Non-Fonctionnels

2.2 Conception UML

2.2.1 Diagramme de Classes

Le diagramme de classes représente le modèle de données basé sur le standard LOM 1.0. L'entité centrale `LomSchema` agrège les neuf catégories de métadonnées :

- **General** : Titre, langue, description, mots-clés
- **Lifecycle** : Version, statut, contributeurs
- **MetaMetadata** : Informations sur les métadonnées
- **Technical** : Format, taille, exigences techniques
- **Educational** : Difficulté, public cible, temps d'apprentissage
- **Rights** : Droits d'auteur, conditions d'utilisation
- **Relation** : Relations avec d'autres ressources
- **Annotation** : Commentaires sur l'usage
- **Classification** : Taxonomies et catégories

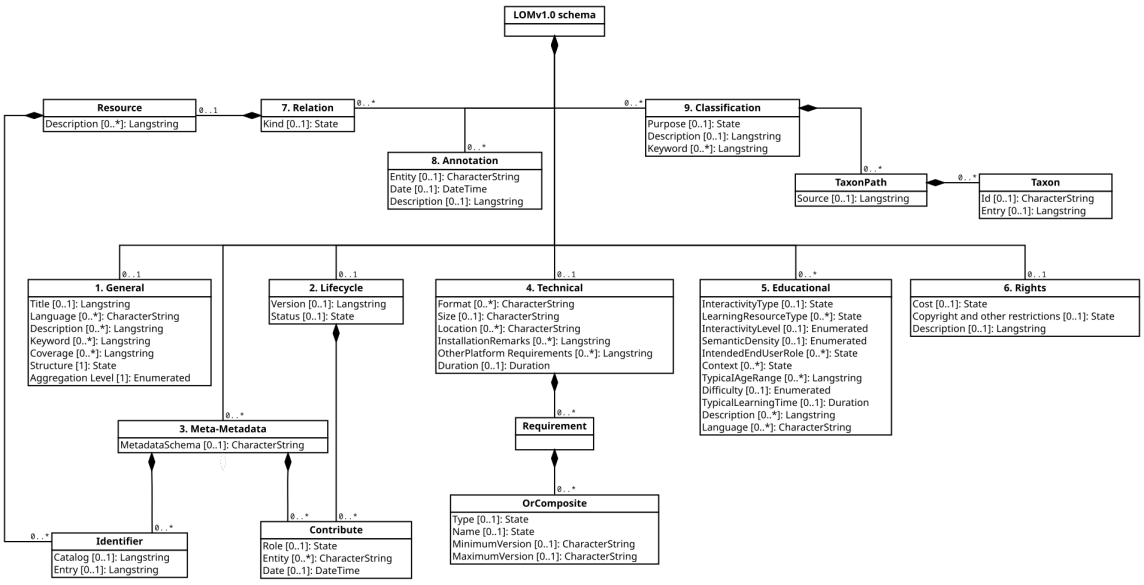

FIGURE 2.1 – Diagramme de Classes

2.3 Conception de la Base de Données

2.3.1 Modèle Logique de Données (MLD)

Le MLD comprend 18 tables organisées selon les catégories LOM et les fonctionnalités applicatives.

Table	Clé Primaire	Clés Étrangères
USER	id	-
LOM_SCHEMA	id	general_id, lifecycle_id, ...
GENERAL	id	lom_schema_id
EDUCATIONAL	id	lom_schema_id
TECHNICAL	id	lom_schema_id
LIFECYCLE	id	lom_schema_id
RIGHTS	id	lom_schema_id
TAG	id	-
LOM_SCHEMA_TAG	lom_schema_id, tag_id	FK vers les deux tables
FAVORITE	id	user_id, resource_id
RATING	id	user_id, resource_id

TABLE 2.3 – Extrait du Modèle Logique de Données

2.3.2 Dictionnaire de Données

Champ	Type	Taille	Contrainte
id	BIGINT	-	PK, AUTO_INCREMENT
username	VARCHAR	50	NOT NULL, UNIQUE
password	VARCHAR	255	NOT NULL
role	VARCHAR	20	DEFAULT 'USER'
resource_title	VARCHAR	255	NOT NULL
difficulty	INT	-	CHECK (1-5)
stars	INT	-	CHECK (1-5)
created_at	DATETIME	-	DEFAULT NOW()

TABLE 2.4 – Dictionnaire de Données (Extrait)

Chapitre 3

Environnement Technique

3.1 Langage de Programmation

Technologie	Justification
Java 17 (LTS)	Version stable avec support à long terme, fonctionnalités modernes (records, sealed classes, pattern matching).
JavaFX 21	Framework moderne pour les interfaces graphiques desktop, support FXML et CSS.

3.2 Environnement de Développement

- **IDE :** IntelliJ IDEA Ultimate - pour ses outils avancés de refactoring et de débogage.
- **Gestion de version :** Git + GitHub - pour le versionnement et la collaboration.
- **Conteneurisation :** Docker + Docker Compose - pour l'environnement de base de données reproductible.

3.3 Gestion de Projet avec Maven

Maven est utilisé pour automatiser la gestion des dépendances et le cycle de build. Voici les dépendances principales :

```
<dependencies>
    <!-- JavaFX -->
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-controls</artifactId>
        <version>21</version>
    </dependency>

    <!-- Hibernate ORM -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.6.15.Final</version>
```

```
    </dependency>

    <!-- MySQL Connector -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.33</version>
    </dependency>
</dependencies>
```

Listing 3.1 – Extrait du pom.xml

3.4 Base de Données

— **SGBD :** MySQL 8.0 - pour sa robustesse et sa compatibilité avec Hibernate.
— **Déploiement :** Docker Compose pour un environnement isolé et reproductible.
— **ORM :** Hibernate 5.6 avec mapping XML pour la persistance des entités.

Chapitre 4

Architecture et Implémentation

4.1 Architecture Logicielle

L'application suit une **architecture en couches** respectant le principe de séparation des responsabilités :

Package	Responsabilité
`org.emsi.entities`	Entités métier (POJO) mappées avec Hibernate
`org.emsi.dao`	Couche d'accès aux données (DAO Pattern)
`org.emsi.service`	Logique métier et services applicatifs
`org.emsi.exceptions`	Gestion centralisée des exceptions
`org.emsi.ui`	Interface utilisateur JavaFX avec FXML

TABLE 4.1 – Organisation des packages

4.2 Design Patterns Utilisés

4.2.1 Pattern Singleton

Le pattern Singleton est utilisé pour garantir une instance unique de la `SessionFactory` Hibernate :

```
1  public class HibernateUtil {
2      private static SessionFactory sessionFactory;
3
4      static {
5          try {
6              Configuration configuration = new Configuration();
7              configuration.configure("hibernate.cfg.xml");
8              sessionFactory = configuration.buildSessionFactory();
9          } catch (Exception e) {
10             throw new ExceptionInInitializerError(e);
11         }
12     }
13
14     public static SessionFactory getSessionFactory() {
15         return sessionFactory;
```

```
    }
}
```

Listing 4.1 – HibernateUtil.java - Pattern Singleton

Justification : Ce pattern garantit qu'une seule connexion à la base de données est maintenue, optimisant ainsi les ressources système.

4.2.2 Pattern DAO (Data Access Object)

Le pattern DAO isole le code d'accès aux données du reste de l'application :

```
public class GenericDao<T, ID extends Serializable> {
    private final Class<T> entityClass;

    public GenericDao(Class<T> entityClass) {
        this.entityClass = entityClass;
    }

    public ID save(T entity) {
        try (Session session = HibernateUtil
                .getSessionFactory().openSession()) {
            Transaction tx = session.beginTransaction();
            ID id = (ID) session.save(entity);
            tx.commit();
            return id;
        }
    }

    public List<T> findAll() {
        try (Session session = HibernateUtil
                .getSessionFactory().openSession()) {
            return session.createQuery(
                "FROM " + entityClass.getSimpleName(),
                entityClass
            ).list();
        }
    }
}
```

Listing 4.2 – GenericDao.java - DAO Générique

Justification : La généricité permet de réutiliser ce code pour toutes les entités, évitant la duplication.

4.3 Extraits de Code Clés

4.3.1 API Streams - Statistiques

L'API Streams est utilisée intensivement pour les calculs statistiques :

```
public Map<String, Long> countResourcesByLanguage() {
    List<LomSchema> resources = lomSchemaDao.findAll();
```

```
3
4       return resources.stream()
5           .filter(r -> r.getGeneral() != null)
6           .filter(r -> r.getGeneral().getLanguage() != null)
7           .collect(Collectors.groupingBy(
8               r -> r.getGeneral().getLanguage(),
9               Collectors.counting()
10          ));
11  }
12
13  public double getAverageDifficulty() {
14      return resources.stream()
15          .filter(r -> r.getEducational() != null)
16          .mapToInt(r -> r.getEducational().getDifficulty())
17          .average()
18          .orElse(0.0);
19  }
```

Listing 4.3 – StatisticsService.java - Utilisation des Streams

4.3.2 Multi-threading avec ThreadPoolExecutor

Le service d'import par lot utilise un pool de threads personnalisé :

```
1   private final ExecutorService executorService =
2       new ThreadPoolExecutor(
3           4,                                  // corePoolSize
4           10,                                 // maximumPoolSize
5           60L, TimeUnit.SECONDS,              // keepAliveTime
6           new LinkedBlockingQueue<>(100), // workQueue
7           r -> {
8               Thread t = new Thread(r, "Import-Thread");
9               t.setDaemon(true);
10              return t;
11          }
12      );
13
14  public Future<LomSchema> importResourceAsync(ResourceData data) {
15      return executorService.submit(() ->
16          lomService.createResource(data.title, data.url)
17      );
18  }
```

Listing 4.4 – BatchImportService.java - ThreadPoolExecutor

4.3.3 Synchronisation avec ReadWriteLock

Le service de synchronisation utilise des verrous pour gérer les accès concurrents :

```
1   private final ReadWriteLock rwLock = new ReentrantReadWriteLock();
2
3   public LomSchema readResourceSafe(Long id) {
```

```
    rwLock.readLock().lock();
    try {
        return resourceCache.computeIfAbsent(id,
            k -> lomSchemaDao.findById(k));
    } finally {
        rwLock.readLock().unlock();
    }
}

public void updateResourceSafe(LomSchema resource) {
    rwLock.writeLock().lock();
    try {
        lomSchemaDao.update(resource);
        resourceCache.remove(resource.getId());
    } finally {
        rwLock.writeLock().unlock();
    }
}
```

Listing 4.5 – DatabaseSyncService.java - ReadWriteLock

Chapitre 5

Interface Utilisateur et Tests

5.1 Présentation des Interfaces

5.1.1 Page de Connexion

L'écran de connexion permet l'authentification des utilisateurs avec validation des champs et messages d'erreur explicites.

Fonctionnalités :

— Validation du nom d'utilisateur et mot de passe

— Lien vers la création de compte

— Messages d'erreur personnalisés

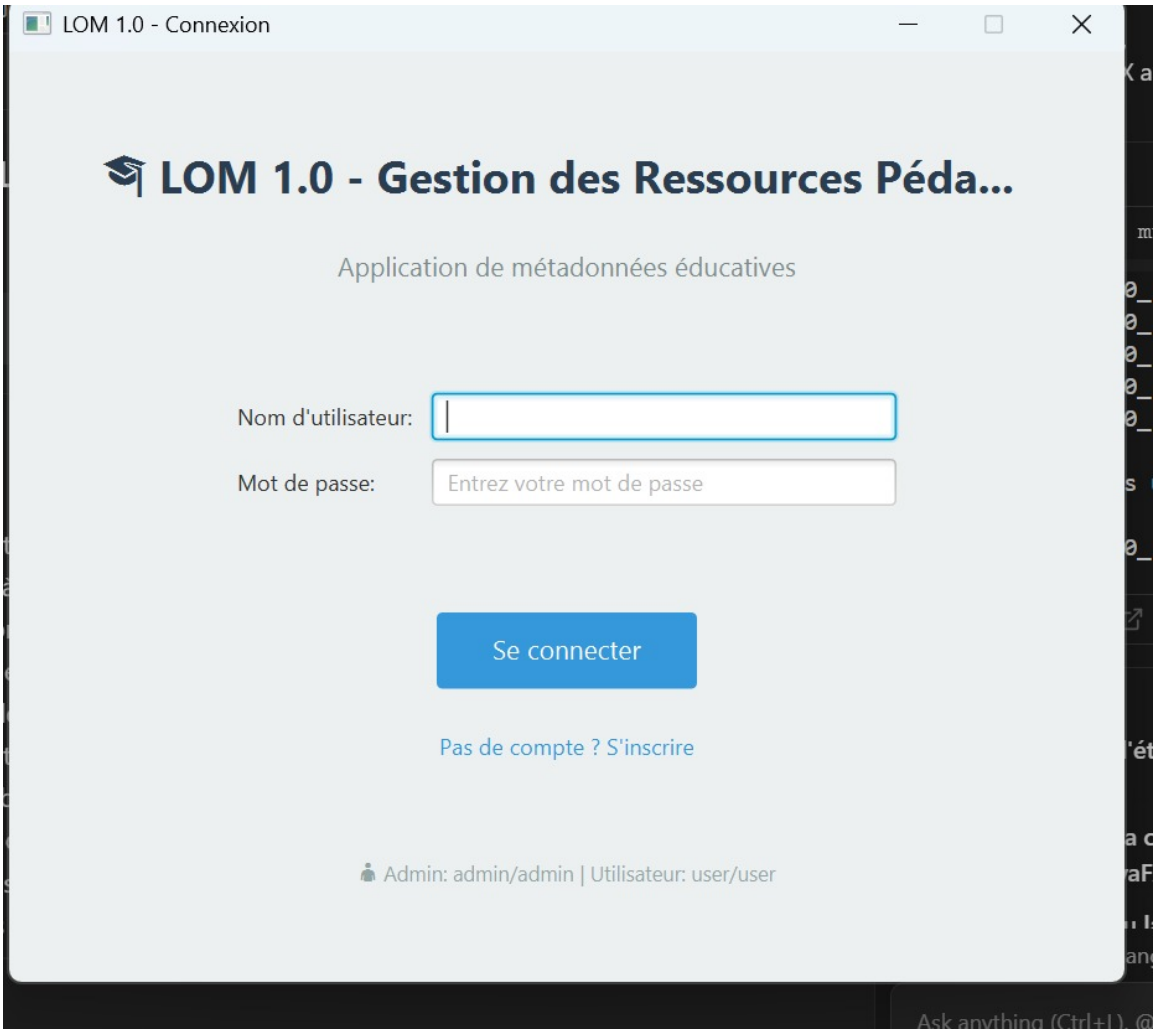

FIGURE 5.1 – Page de Connexion

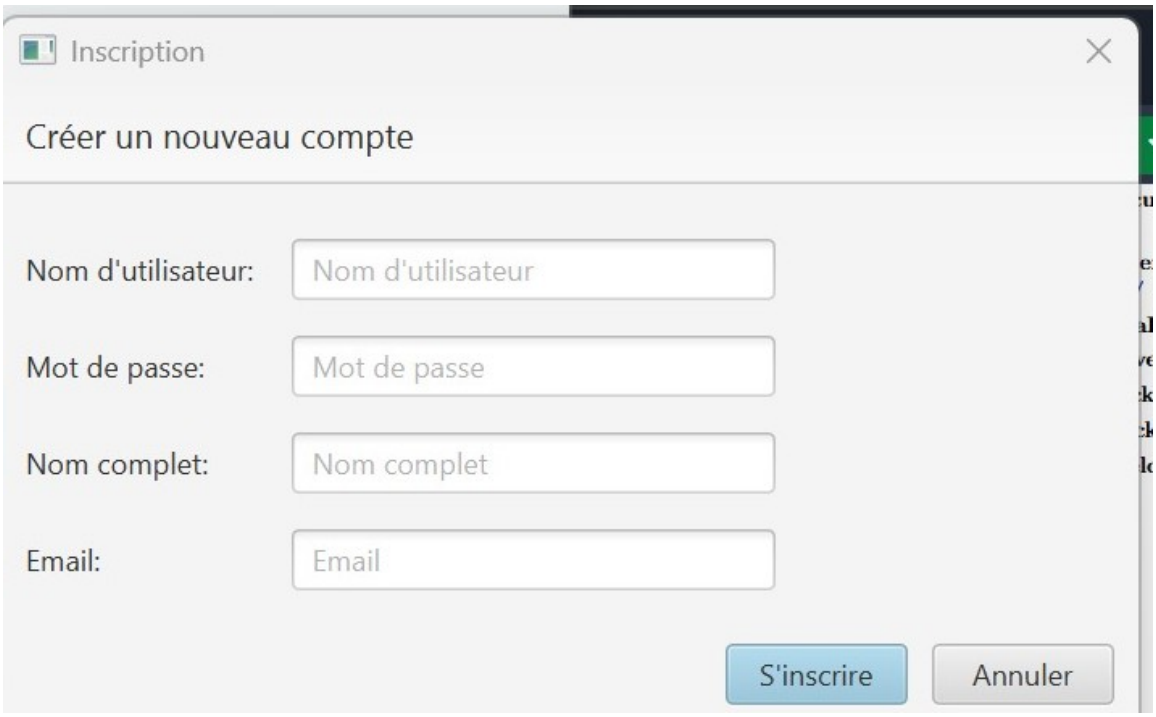

FIGURE 5.2 – Page de Création de Compte

5.1.2 Tableau de Bord Administrateur

Le dashboard administrateur offre une vue complète sur les ressources :
Fonctionnalités :

— Liste des ressources avec recherche
— Boutons CRUD (Créer, Modifier, Supprimer)
— Export XML des métadonnées
— Statistiques en temps réel

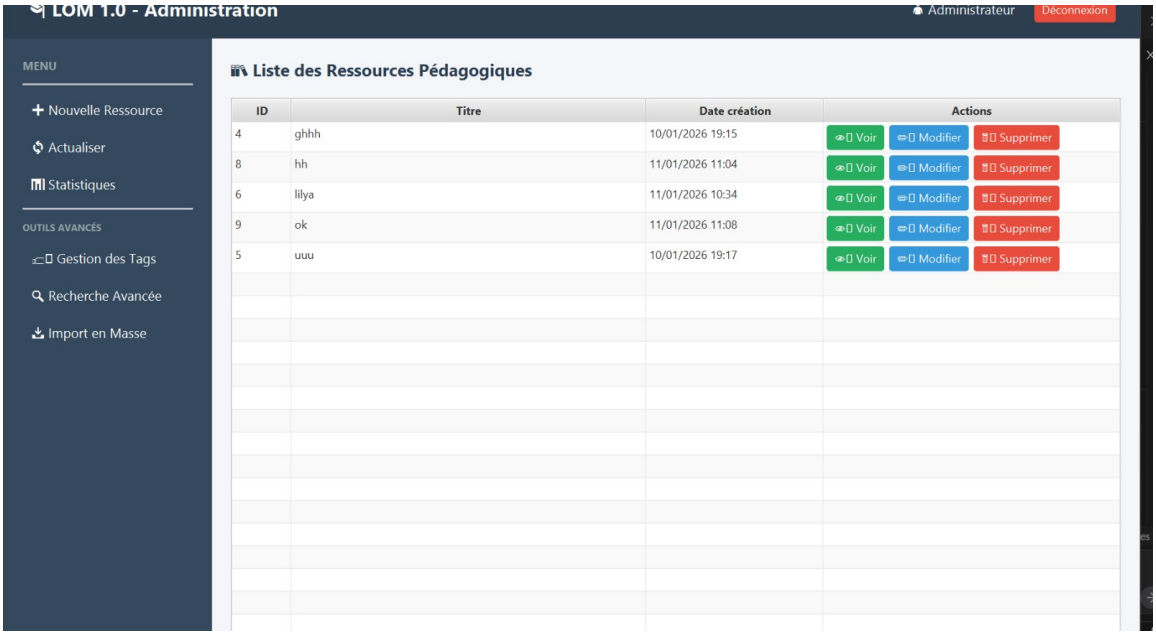

FIGURE 5.3 – Tableau de Bord Administrateur

5.1.3 Tableau de Bord Utilisateur

L'interface utilisateur permet la consultation et l'organisation des ressources :
Fonctionnalités :

- Recherche avancée multi-critères
- Gestion des favoris
- Notation avec commentaires
- Historique de consultation

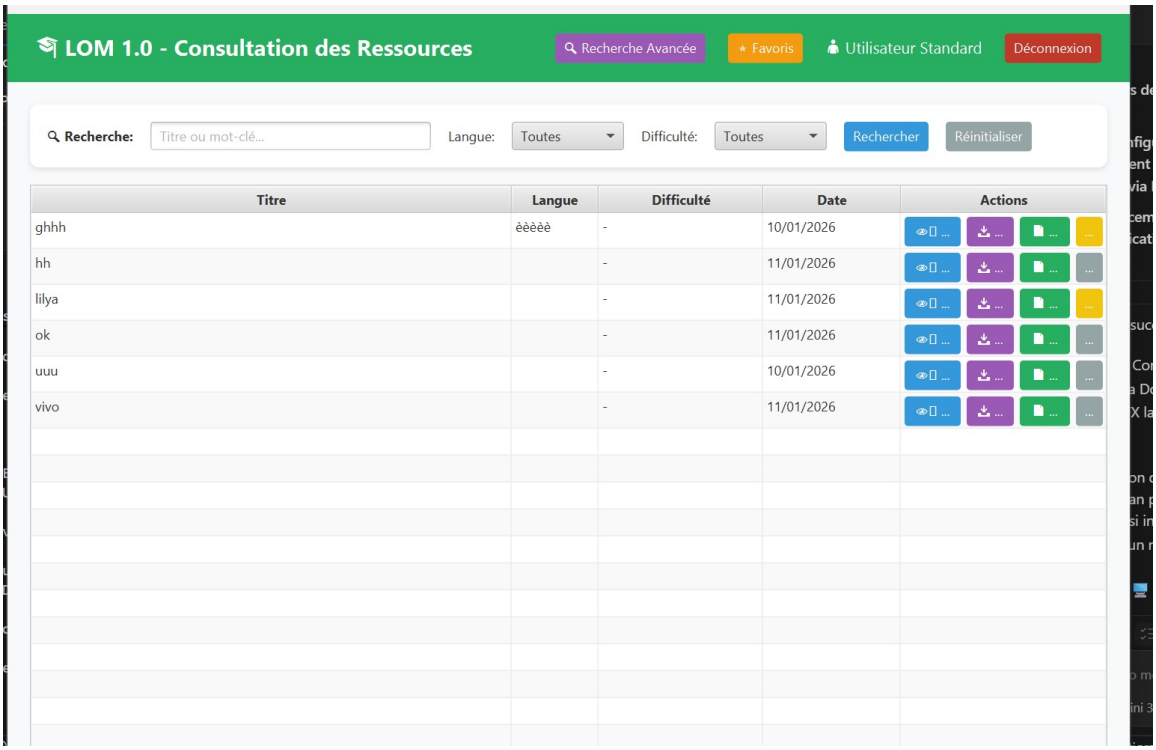

FIGURE 5.4 – Tableau de Bord Utilisateur

5.2 Scénarios de Test

5.2.1 Tests Nominaux

ID	Scénario	Résultat Attendu
T01	Connexion avec identifiants valides	Redirection vers le dashboard
T02	Création d'une nouvelle ressource	Ressource ajoutée et visible dans la liste
T03	Ajout d'une ressource aux favoris	Étoile colorée, ressource dans l'onglet favoris
T04	Notation d'une ressource (4 étoiles)	Note enregistrée, moyenne mise à jour
T05	Export XML d'une ressource	Fichier XML conforme au standard LOM

TABLE 5.1 – Scénarios de Test Nominaux

5.2.2 Tests d'Erreurs

ID	Scénario	Résultat Attendu
E01	Connexion avec mot de passe incorrect	Message "Mot de passe incorrect"
E02	Création de compte avec username existant	Message "Utilisateur déjà existant"
E03	Création de ressource sans titre	Message "Champ obligatoire manquant"
E04	Suppression d'une ressource en cours d'utilisation	Confirmation puis suppression

TABLE 5.2 – Scénarios de Test d'Erreurs

5.2.3 Captures d'écran des Tests

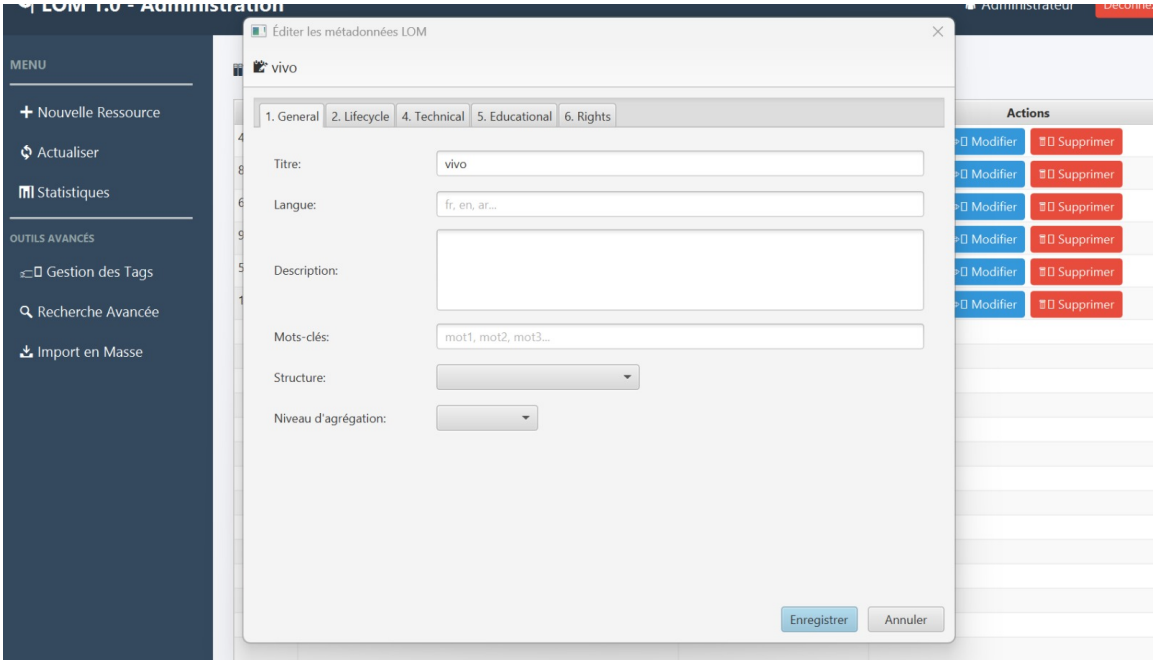

FIGURE 5.5 – Test - Création de Ressource

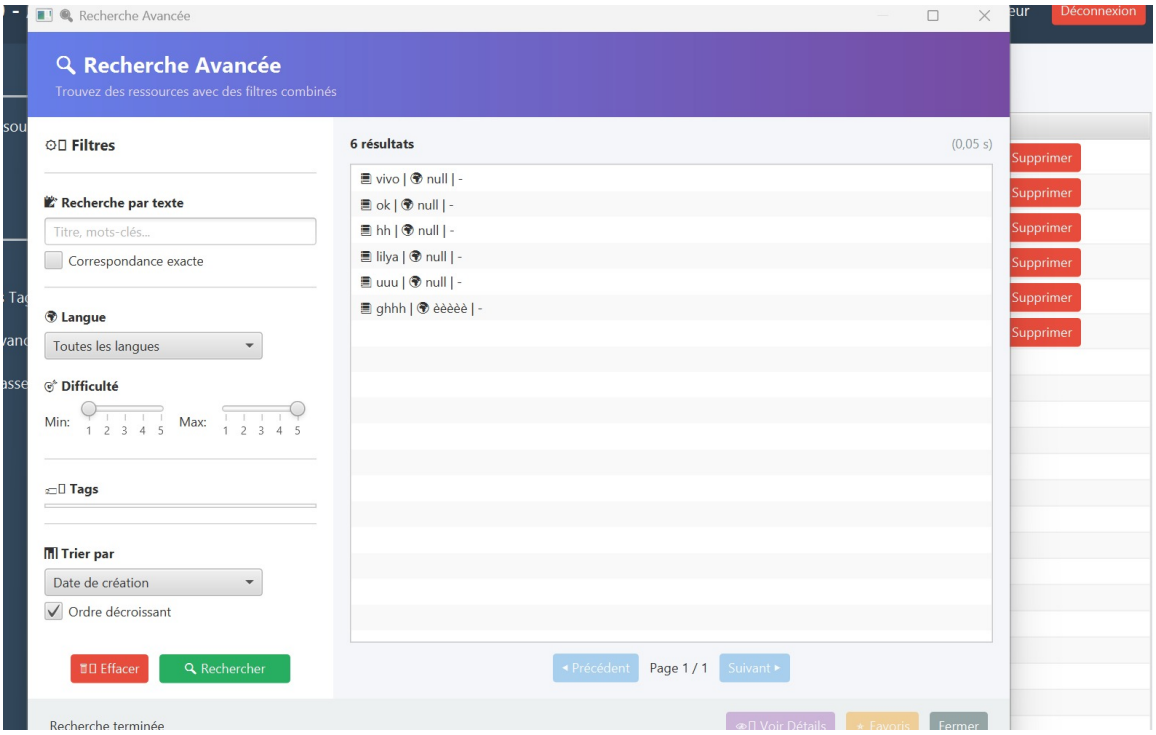

FIGURE 5.6 – Test - Recherche Avancée

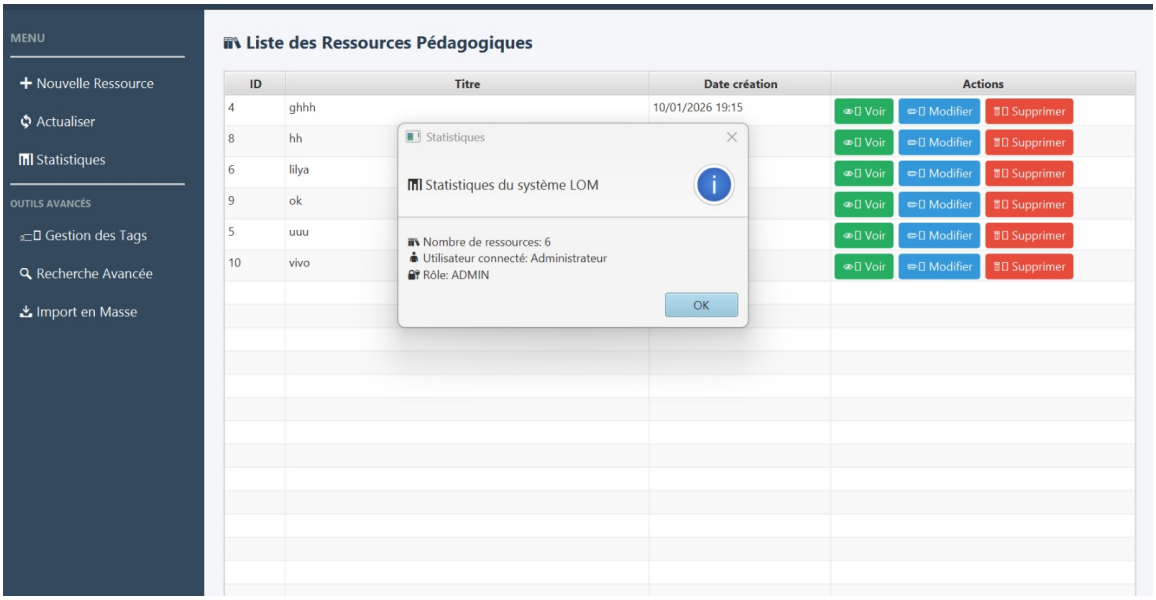

FIGURE 5.7 – Statistiques

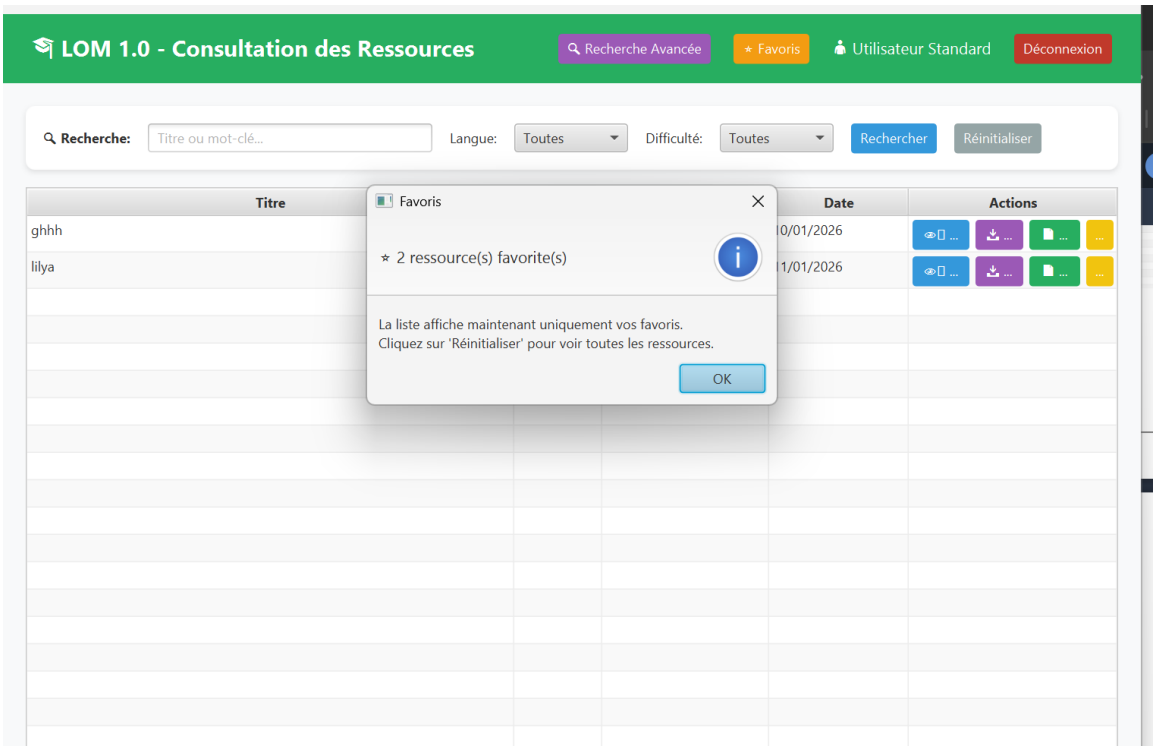

FIGURE 5.8 – Test - Gestion des Favoris

Chapitre 6

Conclusion et Perspectives

6.1 Bilan Technique

Ce projet a permis de développer une application complète de gestion des ressources pédagogiques respectant le standard LOM 1.0. Les objectifs fixés ont été atteints :

Gestion CRUD complète des ressources

Système d'authentification avec rôles

Recherche avancée multi-critères

Système de favoris et de notation

Export XML conforme au standard LOM

Interface utilisateur intuitive avec JavaFX

6.2 Bilan Personnel

Ce projet m'a permis d'acquérir et de consolider plusieurs compétences :

— **Java Avancé :** Maîtrise des Streams, du multi-threading, et des design patterns.

— **Hibernate ORM :** Configuration, mapping XML, gestion des transactions.

— **JavaFX :** Création d'interfaces modernes avec FXML et CSS.

— **Docker :** Conteneurisation de l'environnement de développement.

— **Architecture logicielle :** Application des principes SOLID et des patterns.

6.3 Difficultés Rencontrées

1. **Lazy Loading Hibernate :** Résolu par l'utilisation du fetch eager pour les relations critiques.
2. **Thread-safety JavaFX :** Utilisation de `Platform.runLater()` pour les modifications UI.
3. **Gestion des transactions :** Implémentation du pattern try-with-resources pour garantir la fermeture des sessions.

6.4 Perspectives d'Amélioration

Avec plus de temps, les améliorations suivantes pourraient être envisagées :

— **Version Web :** Migration vers Spring Boot avec API REST.

— **Application Mobile :** Développement d'une application Android/iOS.

— **Intelligence Artificielle :** Recommandations personnalisées basées sur l'historique.

— **Cloud :** Déploiement sur AWS ou Azure avec base de données managée.

— **Tests automatisés :** Ajout de tests unitaires avec JUnit et Mockito.

Webographie

1. **IEEE LOM Standard** - https://standards.ieee.org/standard/1484_12_1-2002.html
2. **Documentation Oracle Java 17** - https://docs.oracle.com/en/java/javase/17/
3. **Hibernate ORM Documentation** - https://hibernate.org/orm/documentation/5.6/
4. **JavaFX Documentation** - https://openjfx.io/javadoc/21/
5. **Maven Central Repository** - https://mvnrepository.com/
6. **Docker Documentation** - https://docs.docker.com/
7. **Stack Overflow** - https://stackoverflow.com/
8. **Baeldung Java Tutorials** - https://www.baeldung.com/