

Projet : Gestionnaire d'une bibliothèque de film

Hantaou Karim

27 mars 2025



Movies Manager

Table des matières

1	Introduction	4
1.1	Contexte du projet	4
1.2	Objectifs	4
1.3	Accès à l'application web	5
2	Outils	6
3	Structure MVC	7
3.1	Présentation du modèle MVC	7
3.2	Extensions au modèle MVC	7
4	Description fonctionnelle	9
4.1	Fonctionnalités principales	9
5	Cas d'utilisation	11
6	Classes	12
7	Base de données	14
8	Diagramme de flux	15
9	Diagramme de séquence	18
10	Tests unitaires	20
10.1	Converture du code	20
10.2	Extrait des tests unitaires	21
11	Javadoc	23
11.1	Génération de la Javadoc	23
11.2	Extrait de la Javadoc	24
11.3	Accès à la Javadoc	24
12	Distribution et déploiement	25
12.1	Conversion du JAR en EXE	25
12.2	Méthodes de distribution	26
13		27
14	Annexes	28
14.1	Ressources	28

14.2	Images	29
14.2.1	Application	29
14.2.2	Plugins IntelliJ	30
14.2.3	Launch4j	33

Chapitre 1

Introduction

1.1 Contexte du projet

Dans le cadre du cours de développement en BTS SIO, ce projet consiste en la réalisation d'une application Java dédiée à la gestion d'une bibliothèque de films. Ce projet vise à appliquer les connaissances acquises en développement logiciel en utilisant les bonnes pratiques de programmation, tout en intégrant des outils et technologies modernes.

L'application proposée permettra de gérer une collection de films, incluant des fonctionnalités comme l'ajout, la modification, la suppression et la recherche de films. Elle s'appuiera sur une interface graphique conviviale développée en JavaFX ou Swing, et assurera la persistance des données grâce à une base de données MySQL.

Les livrables attendus comprennent :

- Une analyse fonctionnelle du besoin, illustrée par des diagrammes UML (diagrammes de cas d'utilisation, de séquence et de classe).
- Un code source documenté avec des commentaires conformes au format Javadoc.
- Une interface graphique opérationnelle pour une expérience utilisateur fluide.
- Des tests unitaires pour valider les principales fonctionnalités.
- Une documentation technique générée en HTML à partir de Javadoc.

Ce projet permet de mettre en œuvre une méthodologie complète de développement, de la conception à la réalisation, en passant par la validation et la documentation, tout en répondant à un besoin concret de gestion d'une bibliothèque de films.

1.2 Objectifs

L'application vise à répondre aux besoins d'une gestion efficace et sécurisée d'une bibliothèque de films. Les objectifs principaux sont les suivants :

- **Sécurité** : Mettre en place une authentification par une page de connexion, afin de restreindre l'accès au gestionnaire de la bibliothèque de films et de garantir la confidentialité des données.
- **Stockage des films** : Fournir une fonctionnalité permettant d'ajouter et de stocker des films dans une bibliothèque structurée, incluant des informations détaillées.
- **Gestion de la bibliothèque** : Offrir la possibilité de modifier les informations des films stockés, de les supprimer ou de les mettre à jour de manière intuitive et efficace.
- **Gestion des réservations** : Ajouter et consulter des réservations associées à des films, pour un suivi précis et une organisation optimale.

1.3 Accès à l'application web

Afin d'accéder et tester cette application, vous pouvez télécharger l'exécutable et utiliser les identifiants :

<https://github.com/karimhantaou/MoviesManager-Distrib>

Nom d'utilisateur : admin

Mot de passe : adminPass



Chapitre 2

Outils

Pour le développement et la gestion de l'application, plusieurs outils ont été utilisés afin d'assurer une efficacité et une qualité optimales. Les outils principaux sont décrits ci-dessous :

- **IntelliJ IDEA Ultimate** : Environnement de développement intégré (IDE) offrant des fonctionnalités avancées.
 1. **Swing** : Utilisé pour la création de l'interface graphique de l'application.
 2. **Database Tools and SQL** : Permet d'administrer et de gérer la base de données directement depuis IntelliJ.
 3. **JUnit 5** : Framework utilisé pour la réalisation des tests unitaires.
 4. **yFiles** : Utilisé pour la génération automatique et la visualisation de diagrammes UML tels que les diagrammes de classe.
- **MySQL / phpMyAdmin** : Base de données relationnelle pour stocker et gérer les données de l'application, administrée via phpMyAdmin pour une gestion simplifiée.
- **Git / GitHub** : Outil de contrôle de version et plateforme de collaboration pour le suivi des modifications du code et la gestion de projet en équipe.
- **Draw.io** : Application en ligne pour concevoir manuellement des diagrammes UML tels que les cas d'utilisation et les diagrammes de séquence.
- **Launch4J** : Outil utilisé pour convertir le fichier JAR en un exécutable (.exe) pour Windows.

Chapitre 3

Structure MVC

L'application a été développée en suivant le modèle architectural MVC (Modèle-Vue-Contrôleur), permettant de séparer distinctement les différentes logiques de l'application et ainsi garantir une meilleure organisation et maintenabilité du code.

3.1 Présentation du modèle MVC

- **Modèle** : Regroupe les fichiers contenant la logique de gestion des données. Cela inclut les interactions avec la base de données MySQL, comme les requêtes pour la récupération, l'ajout, la mise à jour ou la suppression des informations.
- **Vue** : Contient les fichiers responsables de l'affichage graphique. Dans ce projet, l'interface utilisateur a été réalisée en utilisant **Swing**.
- **Controlleur** : Centralise la logique de l'application en établissant un lien entre le Modèle et la Vue. Il gère les interactions utilisateur, les appels aux méthodes du Modèle et la mise à jour de la Vue.
- **main.java** : Point d'entrée principal de l'application, où sont initialisés les composants nécessaires et où débute l'exécution.

3.2 Extensions au modèle MVC

En complément des trois composants traditionnels du modèle MVC, d'autres dossiers ont été ajoutés pour répondre à des besoins spécifiques du projet :

- **Config** : Contient le fichier dédié à l'initialisation de la connexion à la base de données MySQL, permettant une configuration centralisée et réutilisable.
- **Tests** : Regroupe les fichiers de tests unitaires utilisés pour valider les différentes méthodes de l'application. Ces tests garantissent la fiabilité et la robustesse des fonctionnalités

développées.

- **Class** : Inclut toutes les classes d'objet telles que **User**, **Movie** et **Reservation**. Ce composant permet de regrouper les entités et leurs attributs, favorisant une organisation claire et modulaire des objets manipulés par l'application.

Chapitre 4

Description fonctionnelle

4.1 Fonctionnalités principales

1. Fenêtre de connexion :

- **Page de connexion** : Une page d'accueil permettant à l'utilisateur de se connecter à l'application en saisissant un nom d'utilisateur et un mot de passe.
- **Gestion des erreurs** : Si les identifiants sont incorrects, un message d'erreur clair et explicite est affiché pour inviter l'utilisateur à réessayer.

2. Bibliothèque de films :

- **Section de gauche** : Liste des films disponibles dans la bibliothèque. Chaque élément affiche les informations de base du film.
- **Section de droite** : Espace dédié à l'ajout ou à la modification de films via un formulaire.
 - Si un film est sélectionné dans la section de gauche, le formulaire permet de modifier ses informations.
 - Sinon, le formulaire permet d'ajouter un nouveau film.

Informations manipulées :

- Nom du film.
- Auteur du film.
- Année de sortie du film.

Fonctionnalités supplémentaires accessibles pour un film sélectionné :

- Ajouter une réservation.
- Modifier les informations du film.
- Supprimer le film de la bibliothèque.
- Cacher les informations du film pour revenir au formulaire d'ajout.

3. Fenêtre d'ajout de réservation :

- **Formulaire** : Permet d'ajouter une réservation à un film sélectionné. Les informations suivantes doivent être renseignées :
 - Nom du client.
 - Date de début de la réservation.
 - Date de fin de la réservation.

- **Gestion des erreurs** : Si une date n'est pas valide ou si un champ est manquant, un message d'erreur s'affiche sous forme de popup.

4. **Fenêtre de la liste des réservations** :

- **Section de gauche** : Affiche la liste des réservations enregistrées.
- **Section de droite** : Permet de visualiser les détails d'une réservation sélectionnée, comme les informations du client et les dates associées.

Chapitre 5

Cas d'utilisation

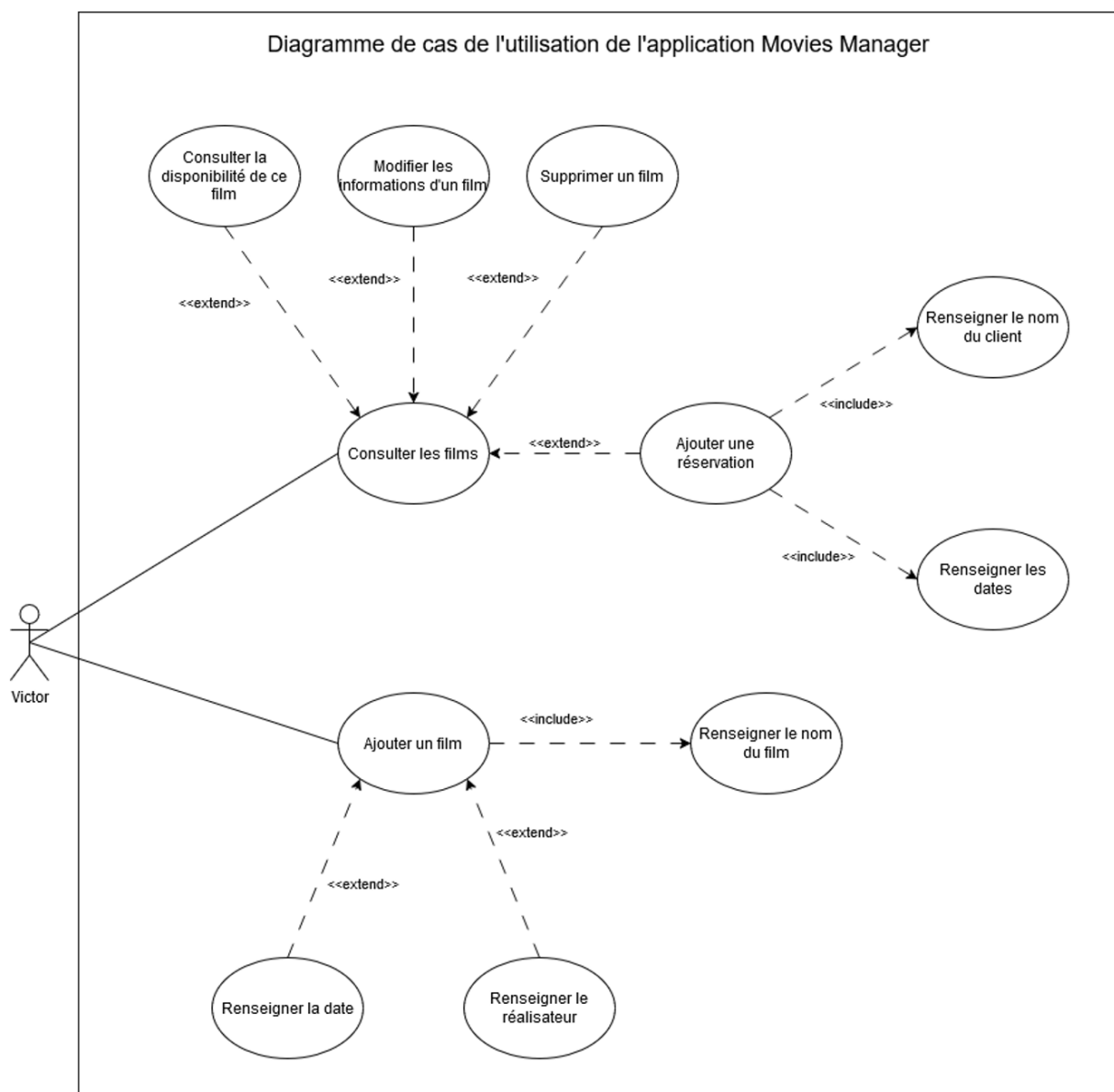


FIGURE 5.1 – Diagramme de cas d'utilisation

Chapitre 6

Classes

L'application Java repose sur trois classes principales, chacune correspondant à un élément clé du système. Ces classes structurent les données de l'application et facilitent leur manipulation.

- **User :**

- Contient les informations relatives aux utilisateurs de l'application, telles que :
 - Nom d'utilisateur (**username**).
 - Mot de passe (**password**).
 - Rôle de l'utilisateur (pas utilisé dans l'application) (**role**)
- Permet de gérer les authentifications et de sécuriser l'accès à l'application.

- **Movie :**

- Représente les films dans la bibliothèque et contient les informations suivantes :
 - Nom du film (**name**).
 - Auteur ou réalisateur (**author**).
 - Année de sortie (**year**).
- Inclut des méthodes pour manipuler ces informations, comme leur ajout, modification ou suppression.

- **Reservation :**

- Gère les informations des réservations associées à un film :
 - Nom du client (**name**).
 - Id du film réservé (**movieId**).
 - Date de début (**startDate**).
 - Date de fin (**endDate**).
- Permet de suivre les réservations pour chaque film de manière organisée.

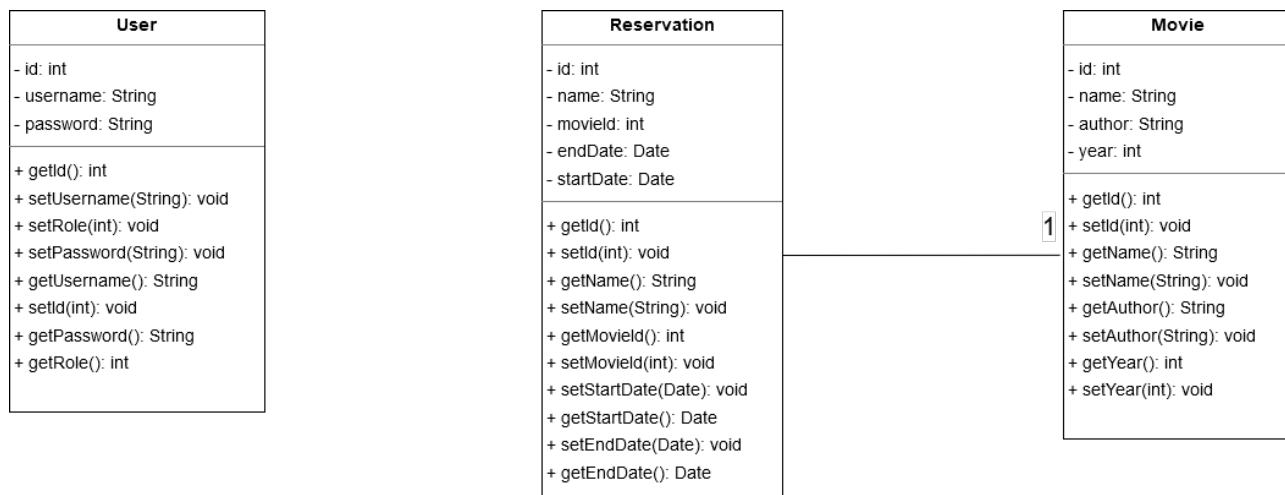


FIGURE 6.1 – Diagramme de classe



FIGURE 6.2 – Diagramme d'objet

Chapitre 7

Base de données

Une base de donnée MySQL à été utilisé afin de stocker les différentes informations.

- **users** : Les différents utilisateurs.
- **movies** : Les différents films enregistrés dans la bibliothèque.
- **reservations** : Les différentes réservations.

users
username: varchar(255) password: varchar(255) role: int
id: int

movies
name: varchar(255) author: varchar(255) year: int state: int
id: int

reservations
movieId: int name: varchar(255) dateStart: date dateEnd: date
id: int

FIGURE 7.1 – Diagramme de classe de la base de données

Chapitre 8

Diagramme de flux

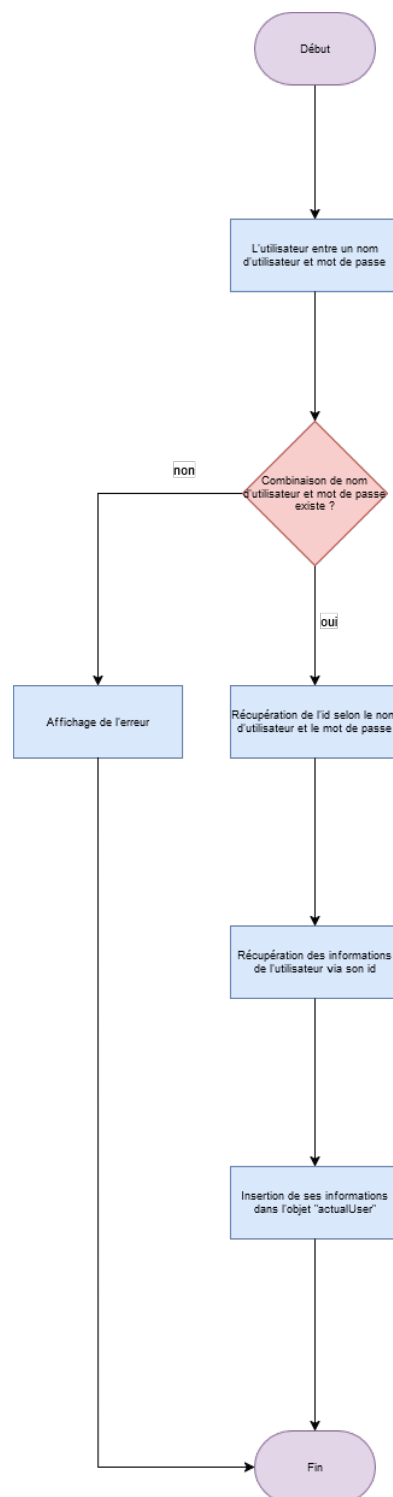


FIGURE 8.1 – Diagramme de flux de la connexion à l'application

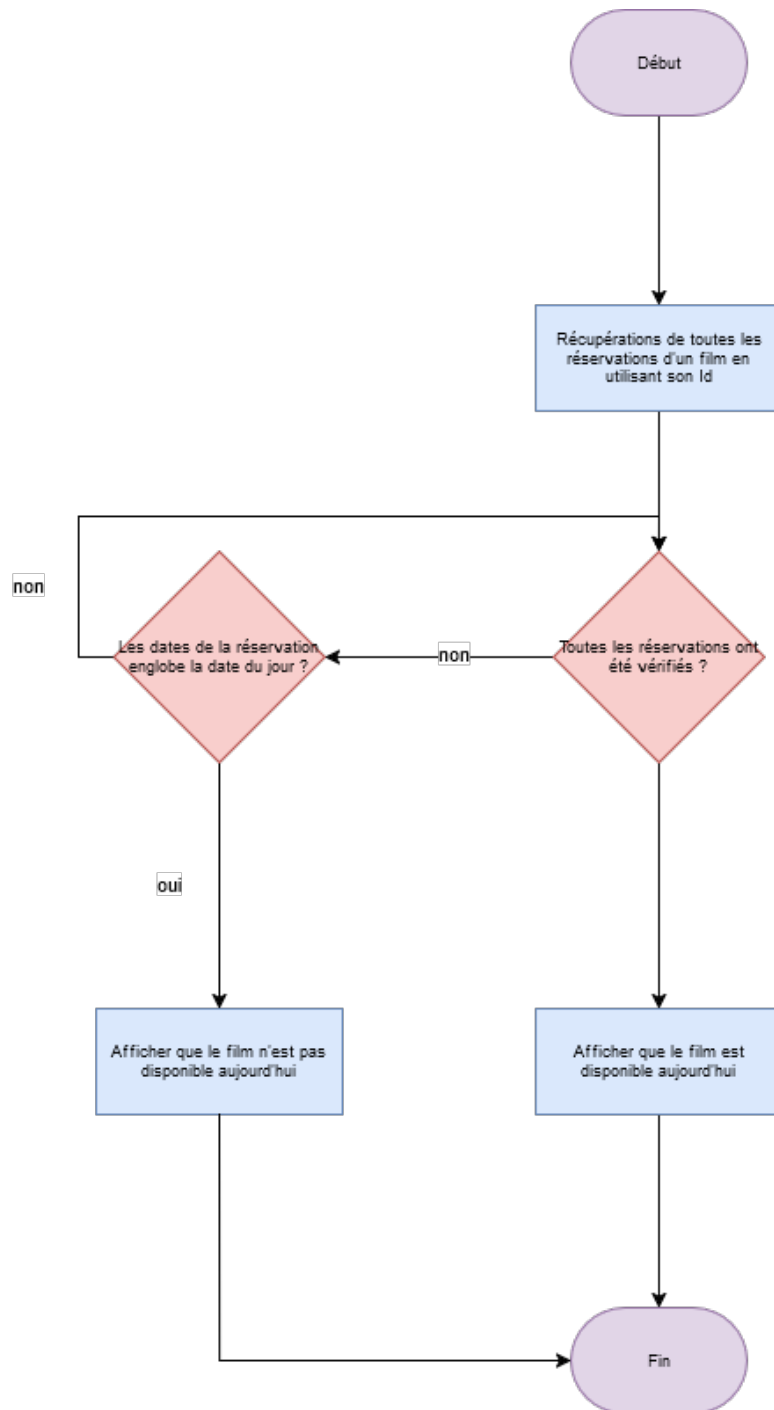


FIGURE 8.2 – Diagramme de flux de la vérification de la disponibilité d'un film

Chapitre 9

Diagramme de séquence

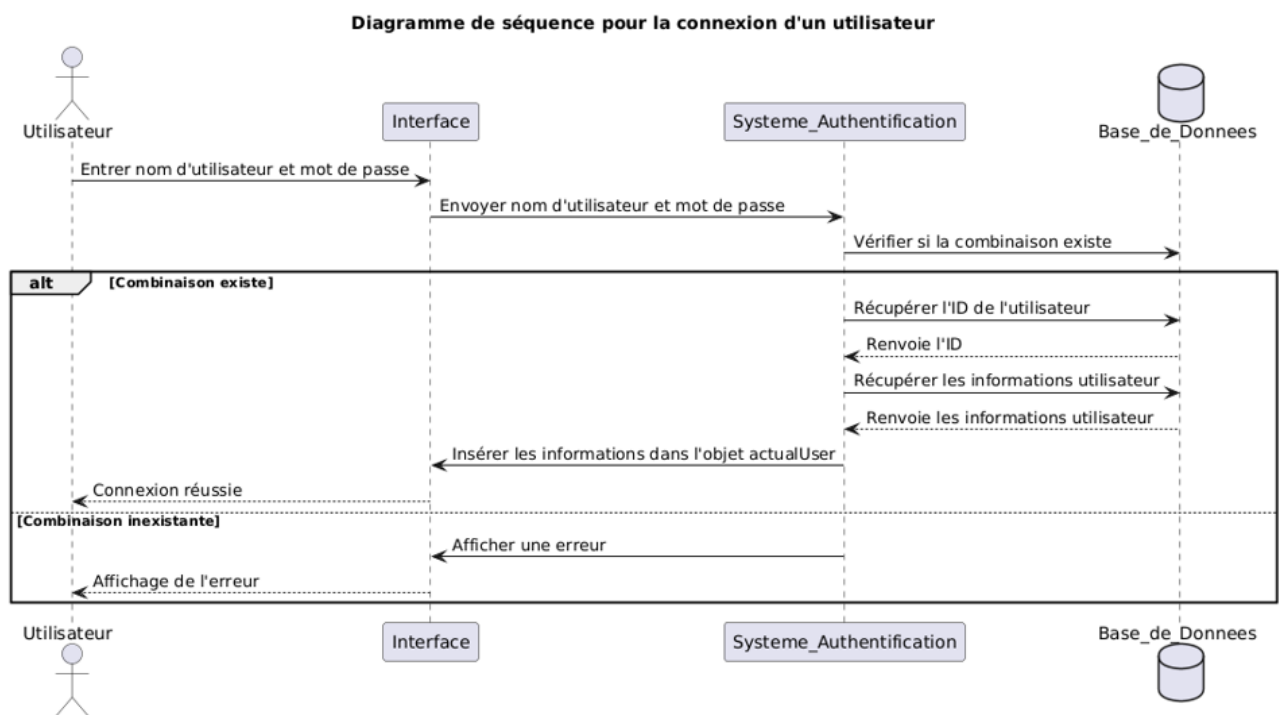


FIGURE 9.1 – Diagramme de séquence de la connexion à l'application



FIGURE 9.2 – Diagramme de séquence de la vérification de la disponibilité d'un film

Chapitre 10

Tests unitaires

Des tests unitaires ont été réalisés pour vérifier le bon fonctionnement des différentes méthodes. Pour ce faire j'ai utilisé JUnit 5 et les outils de test intégrés à IntelliJ IDEA Ultimate.

10.1 Converture du code

Les tests unitaires ont été exécutés avec une analyse de couverture de code pour mesurer les parties du code source testées. Ce rapport identifie les zones non couvertes afin de les améliorer.

Ci-dessous, une capture d'écran illustre les résultats obtenus.

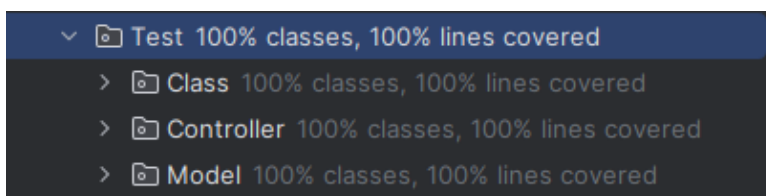
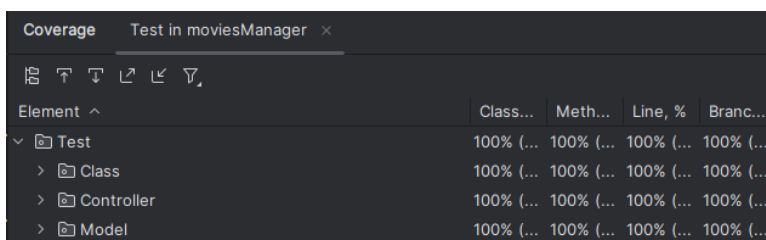


FIGURE 10.1 – Enter Caption



Element ^	Class...	Meth...	Line, %	Branch...
Test	100% (...)	100% (...)	100% (...)	100% (...)
> Class	100% (...)	100% (...)	100% (...)	100% (...)
> Controller	100% (...)	100% (...)	100% (...)	100% (...)
> Model	100% (...)	100% (...)	100% (...)	100% (...)

FIGURE 10.2 – Enter Caption

10.2 Extrait des tests unitaires

```
1
2  /**
3   * Methode ex cut e avant tous les tests.
4   * Initialise les donn es n cessaires pour les tests.
5   * Ces donn es simulent les valeurs des films et de leurs attributs
6   * dans la base de donn es.
7   *
8   * @throws Exception Si une exception se produit lors de l'
9   initialisation.
10  */
11  @BeforeAll
12  public static void setUpBeforeClass() throws Exception {
13      // Initialisation des variables de test
14      name1 = "Film1";
15      name2 = "Film2";
16
17      autor1 = "Auteur1";
18      autor2 = "Auteur2";
19
20      year1 = 1965;
21      year2 = 2005;
22
23      state1 = 0; // tat du film 1 (ex. : non disponible)
24      state2 = 1; // tat du film 2 (ex. : disponible)
25  }
```

Listing 10.1 – Extrait des tests unitaires.

```

1  /**
2   * Test de la m thode getMovieById().
3   * V rifie que la r cup ration des films par ID fonctionne
4   correctement pour des films existants dans la base de donn es.
5   */
6   @Test
7   void testGetMovieById() {
8
9       // V rification pour le film avec l'ID 1
10      Movie movie1 = movieModel.getMovieById(1);
11
12      // V rifie que le film n'est pas null (le film doit exister dans la
13      base)
14      assertNotNull(movie1);
15
16      // V rifie que l'ID du film r cup r correspond l'ID attendu
17      assertEquals(1, movie1.getId());
18
19      // V rifie que les autres attributs du film sont corrects
20      assertEquals(name1, movie1.getName()); // V rifie le nom du film
21      assertEquals(autor1, movie1.getAuthor()); // V rifie l'auteur du
22      film
23      assertEquals(year1, movie1.getYear()); // V rifie l'ann e du film
24      assertEquals(state1, movie1.getState()); // V rifie l' tat du film
25
26      // V rification pour le film avec l'ID 2
27      Movie movie2 = movieModel.getMovieById(2);
28
29      // V rifie que le film n'est pas null
30      assertNotNull(movie2);
31
32      // V rifie que l'ID du film r cup r correspond l'ID attendu
33      assertEquals(2, movie2.getId());
34
35      // V rifie que les autres attributs du film sont corrects
36      assertEquals(name2, movie2.getName()); // V rifie le nom du film
37      assertEquals(autor2, movie2.getAuthor()); // V rifie l'auteur du
38      film
39      assertEquals(year2, movie2.getYear()); // V rifie l'ann e du film
40      assertEquals(state2, movie2.getState()); // V rifie l' tat du film
41  }

```

Listing 10.2 – Extrait des tests unitaires.

Chapitre 11

Javadoc

La Javadoc a été générée pour documenter le code de manière détaillée. Elle permet de décrire le fonctionnement des classes et des méthodes, facilitant ainsi la compréhension et l'utilisation du code par d'autres développeurs.

11.1 Génération de la Javadoc

La Javadoc a été générée à l'aide des outils intégrés dans IntelliJ IDEA. Grâce à ces outils, il est possible de créer automatiquement la documentation à partir des commentaires JavaDoc présents dans le code source. Cela permet de garantir que la documentation est toujours à jour avec le code.

11.2 Extrait de la Javadoc

Voici un extrait de la Javadoc générée pour la classe Main :

```
1 /**
2  * Classe principale de l'application. Cette classe contient le point d'
   entr e du programme
3  * et initialise les composants n cessaires      l'ex cution de l'
   application.
4  */
5 public class Main {
6     /**
7      * Point d'entr e de l'application.
8      *
9      * @param args les arguments de ligne de commande
10     */
11     public static void main(String[] args) {
12         // Code de l'application
13     }
14 }
```

Listing 11.1 – Extrait de la class Main (les accents s'affichent pas car latex c'est guez et oui j'ai essayé de les faire marcher et non j'ai pas réussi.).

11.3 Accès à la Javadoc

La Javadoc complète de l'application est disponible en ligne. Vous pouvez y accéder à l'adresse suivante :

<https://karimhantaou.github.io/MoviesManager-Distrib/>



Chapitre 12

Distribution et déploiement

Ce chapitre présente le processus de distribution de l'application, en particulier la conversion du fichier `.jar` généré par **IntelliJ IDEA** en fichier exécutable `.exe` à l'aide de l'outil **Launch4J**, ainsi que les différentes méthodes utilisées pour déployer et distribuer l'application.

12.1 Conversion du JAR en EXE

Le fichier `.jar` de l'application est généré directement depuis **IntelliJ IDEA**, en utilisant les fonctionnalités de build intégrées de l'IDE. Une fois le fichier `.jar` généré, il est converti en fichier exécutable `.exe` à l'aide de l'outil **Launch4J**. Cette conversion permet de lancer l'application directement sur un système Windows sans avoir à installer Java au préalable.

- **Génération du fichier JAR avec IntelliJ IDEA** : IntelliJ IDEA facilite la création du fichier `.jar` via l'option de construction du projet. Cette fonctionnalité permet de compiler l'application et de créer un fichier exécutable autonome contenant tous les éléments nécessaires au bon fonctionnement de l'application.
- **Launch4J** : Cet outil génère un fichier `.exe` qui encapsule le fichier `.jar` et assure son lancement dans un environnement Windows. Il permet également de configurer des paramètres comme l'icône du programme ou des options de gestion des erreurs.
- **Configuration de Launch4J** : Une fois le fichier `.jar` généré, les paramètres de configuration de **Launch4J** sont définis, tels que le chemin vers le fichier `.jar`, la mémoire allouée et d'autres options liées à l'exécution de l'application.

La conversion en `.exe` simplifie grandement le déploiement, car elle évite aux utilisateurs de devoir installer Java, rendant ainsi l'application plus accessible.

12.2 Méthodes de distribution

Le fichier exécutable `.exe`, ainsi que la Javadoc complète du projet, sont disponibles en téléchargement sur le dépôt GitHub du projet.

<https://github.com/karimhantaou/MoviesManager-Distrib>

Nom d'utilisateur : admin

Mot de passe : adminPass

Chapitre 13

Chapitre 14

Annexes

14.1 Ressources

- Ressources pour le plugin Swing d’IntelliJ : <https://www.jetbrains.com/help/idea/design-gui-using-swing.html>
- Ressources pour l’utilisation de MySQL avec IntelliJ : <https://www.jetbrains.com/help/idea/mysql.html>
- Ressources pour les bases de données relationnelles avec IntelliJ : <https://www.jetbrains.com/help/idea/relational-databases.html>
- Ressources pour l’utilisation du plugin IntelliJ afin de générer des diagrammes de classes : https://www.jetbrains.com/help/idea/class-diagram.html#analyze_class
- Ressources pour la compilation des applications avec IntelliJ : https://www.jetbrains.com/help/idea/compiling-applications.html#rebuild_project
- Ressources de Launch4j (wrap de .jar) : <https://launch4j.sourceforge.net/>
- Ressources pour JUnit dans IntelliJ : <https://www.jetbrains.com/help/idea/junit.html>
- Application pour créer des diagrammes en ligne (Draw.io) : <https://app.diagrams.net/>
- Ressources pour MySQL Connector : <https://www.mysql.com/products/connector/>
- Ressources pour OpenJDK 23 : <https://openjdk.org/projects/jdk/23/>

14.2 Images

14.2.1 Application

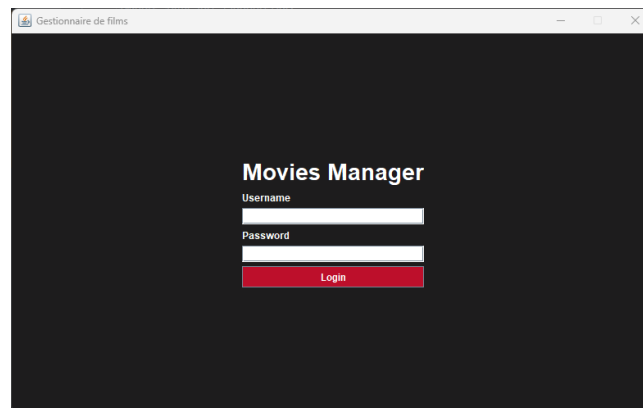


FIGURE 14.1 – Fenêtre de connexion

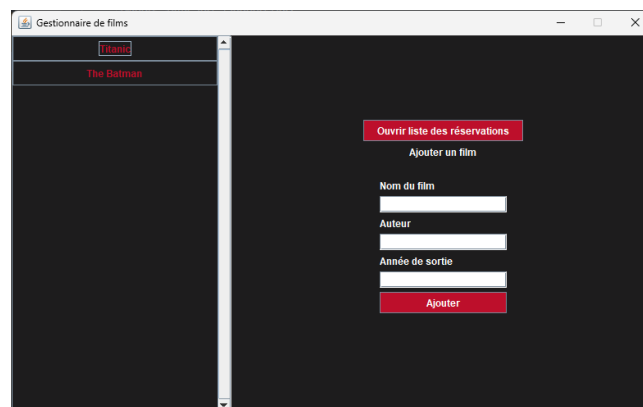


FIGURE 14.2 – Fenêtre d'ajout d'un film

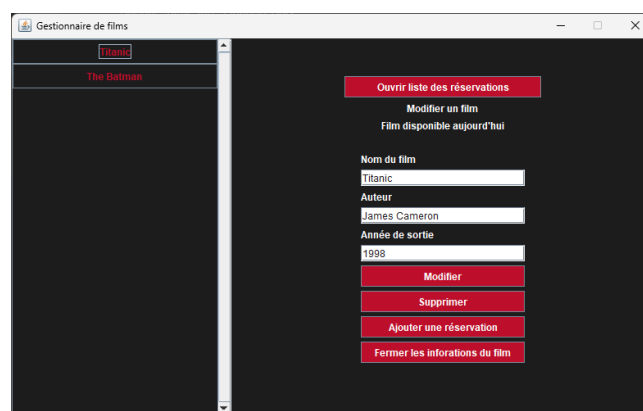


FIGURE 14.3 – Fenêtre avec les informations d'un film enregistré

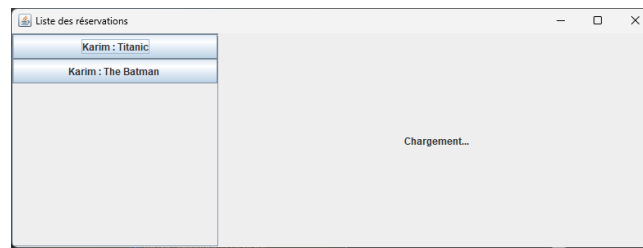


FIGURE 14.4 – Fenêtre de la liste des réservations

FIGURE 14.5 – Fenêtre d'ajout d'une réservation

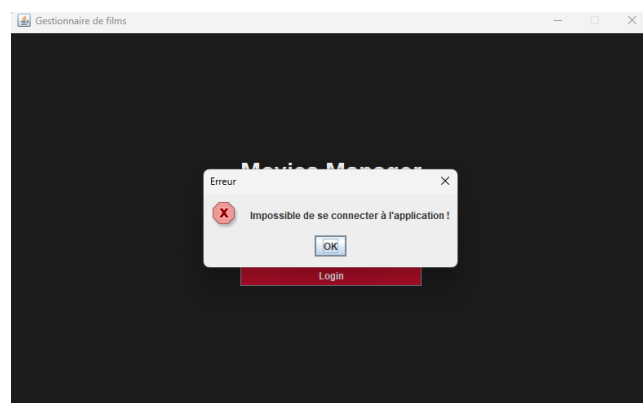


FIGURE 14.6 – Fenêtre d'erreur quand il est impossible de se connecter à la base de données.

14.2.2 Plugins IntelliJ

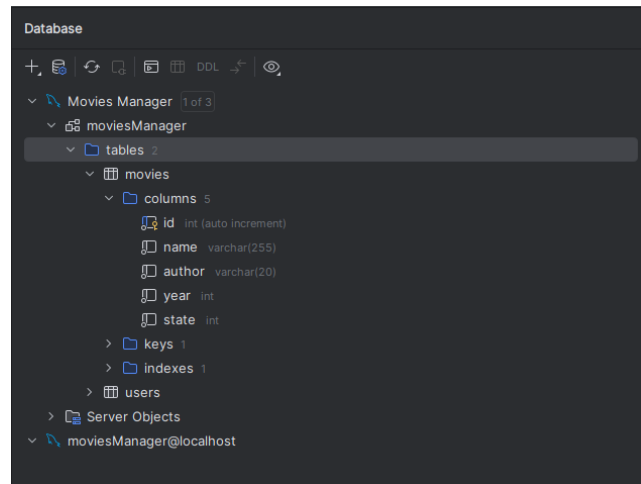


FIGURE 14.7 – Fenêtre IntelliJ du plugin des bases de données

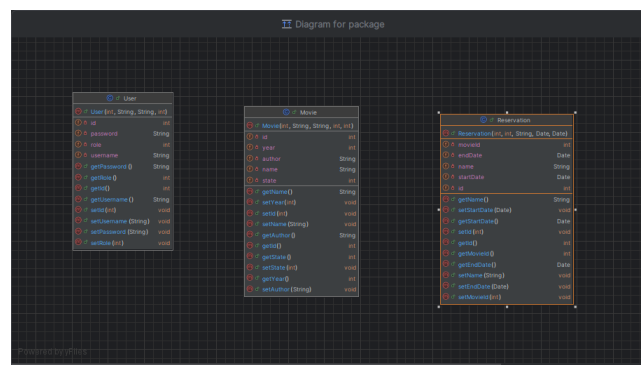


FIGURE 14.8 – Fenêtre IntelliJ génération des diagrammes de classes

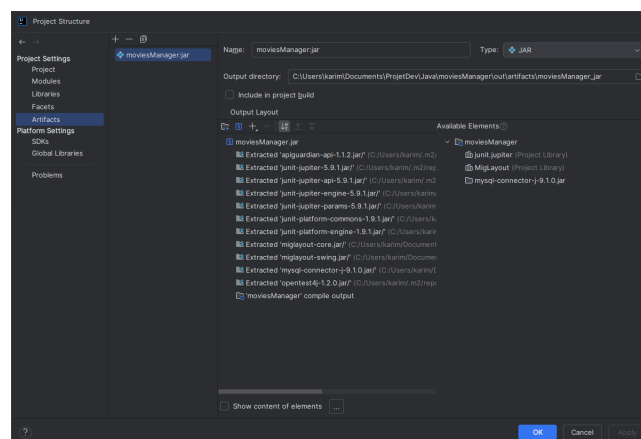


FIGURE 14.9 – Fenêtre IntelliJ création du .jar

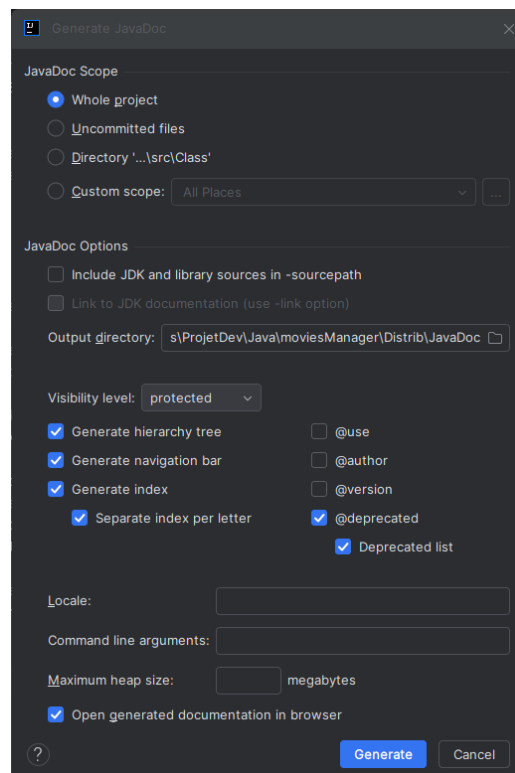


FIGURE 14.10 – Fenêtre IntelliJ Javadoc

14.2.3 Launch4j

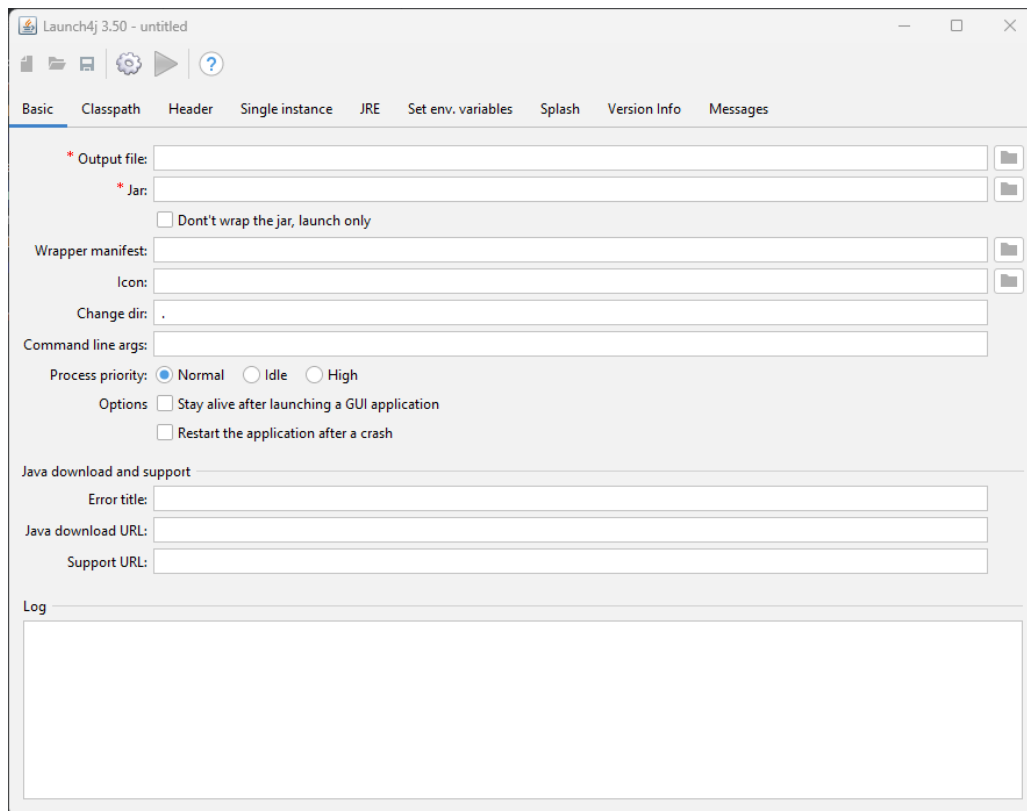


FIGURE 14.11 – Application Launch4j