المدرسة الوطنية للعلوم التطبيقية -أكادير
ⵜⵉⵎⵙⵙⵓⵜ ⵏ ⵜⵎⴰⵡⴰⵙⵜ ⵜⵉⵏⵖⵓⵜⵉⵏ - ⴰⵅⴰⴷⵉⵔ.
ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES - AGADIR

École Nationale des Sciences Appliquees

Département : Génie Informatique

Rapport de Projet JEE

Développement d'une application web de streaming de films - WOVIES

Réalisé par :
Malak Bourrich
Abdesamad Ait El Houari

Filière :
Génie Developpement Logiciel et Applicatif

Encadré par :
Mme. Zahra Bnider

Année universitaire : 2024–2025

Table des matières

Chapitre 1

Introduction

Le projet **Wovies** est une application web de streaming de films et séries développée dans le cadre d'un projet universitaire axé sur l'architecture JEE. Ce projet représente une mise en pratique concrète des concepts d'architecture logicielle moderne et des technologies Java Enterprise Edition.

1.1 Contexte du Projet

Dans un contexte où les plateformes de streaming dominent l'industrie du divertissement numérique, notre projet vise à concevoir une solution similaire tout en respectant les standards académiques et professionnels. L'application Wovies permet aux utilisateurs de découvrir, rechercher et organiser leur contenu vidéo préféré à travers une interface web intuitive et performante.

1.2 Objectifs

Objectifs Techniques : Concevoir un site web fonctionnel et structuré en respectant l'architecture MVC (Model-View-Controller), intégrer plusieurs technologies Java professionnelles (JSP, Hibernate, JSoup), et assurer une communication efficace avec une base de données MySQL.

Objectifs Fonctionnels : Offrir une expérience utilisateur fluide permettant la recherche de films et séries, la gestion d'une liste de favoris (watchlist), la consultation d'un historique de visionnage, et la gestion complète d'un compte utilisateur personnalisé.

Objectifs Pédagogiques : Développer nos compétences en architecture logicielle, en intégration d'API externes, en gestion de bases de données relationnelles, et en travail collaboratif sur un projet d'envergure.

1.3 Périmètre du Projet

Le projet couvre l'ensemble du cycle de développement d'une application web, depuis la conception de l'architecture jusqu'au déploiement, en passant par la modélisation UML, l'implémentation backend et frontend, et l'intégration de services externes pour l'enrichissement du contenu.

Chapitre 2

Architecture du Site Web

L'architecture du projet Wovies repose sur le modèle **MVC (Model-View-Controller)** en suivant les standards **Jakarta EE**. Cette architecture garantit une séparation claire des responsabilités et facilite la maintenance et l'évolutivité du code.

2.1 Architecture MVC Adoptée

Notre implémentation respecte strictement les principes de l'architecture MVC en organisant le code en packages distincts et cohérents :

Package Model : Ce package constitue la couche de base du projet. Il contient toutes les classes d'entités qui représentent les objets métier de l'application (Account, Video, Movie, Series, WatchList, History, Server, Season). Ces classes définissent la structure des données et établissent les relations entre les différentes entités du système.

Package DAO (Data Access Object) : Cette couche est responsable de toutes les interactions avec la base de données. Elle encapsule la logique d'accès aux données et fournit une abstraction pour les opérations CRUD. L'utilisation d'Hibernate ORM dans cette couche permet de mapper automatiquement les objets Java vers les tables de la base de données MySQL.

Package Service : Le package Service implémente toute la logique métier de l'application. Il agit comme un intermédiaire entre la couche Controller et la couche DAO. Cette couche orchestre les opérations complexes, applique les règles métier, et coordonne les appels à plusieurs DAO si nécessaire.

Package Controller : Ce package contient les servlets Java EE qui gèrent les requêtes HTTP entrantes. Les servlets agissent comme des contrôleurs en interceptant les requêtes des utilisateurs, en appelant les services appropriés pour traiter les données, et en redirigeant vers les vues JSP correspondantes.

2.2 Avantages de cette Architecture

Cette organisation modulaire offre plusieurs avantages : séparation claire des préoccupations, réutilisabilité des composants, facilitation des tests unitaires, et scalabilité optimale. Chaque couche peut être optimisée ou étendue sans impacter les autres.

2.3 Communication entre les Couches

La communication suit un flux unidirectionnel : Requête utilisateur → Controller (Servlet) → Service (Logique métier) → DAO → Hibernate → MySQL. Les résultats remontent dans le sens inverse jusqu'à la vue JSP qui génère la réponse HTML.

Chapitre 3

Conception UML

3.1 Diagramme de Classes

Le diagramme de classes illustre la structure statique du système et les relations entre les différentes entités.

Entité Account : Représente un utilisateur avec les attributs id_user, firstname, lastname, email, password, et isConnected. Liée à Home via une relation ”accède”.

Entité Home : Hub central agrégant les vidéos tendances et disponibles. Expose les méthodes getTrendings(), setTrendings(), getVideos(), setVideos(), getFeatures() et setFeatures().

Entité Video : Superclasse abstraite représentant tout contenu vidéo. Contient id, type, title, description, release_date, rating, image_url, et link. Héritée par Movie et Series.

Entités Movie et Series : Movie ajoute movieUrl, director, genre, duration, ageRating, et servers. Series se spécialise avec numberOfSeasons, seasons, et méthodes de gestion des saisons/épisodes.

Entité Season : Modélise une saison avec seasonNumber, numberOfEpisodes, et episodes.

Entité Server : Représente un serveur de streaming avec id_server, name, url. Relation ”héberge” avec Movie.

Entité WatchList : Gère les favoris avec id, userEmail, videoTitle, videoType, videoUrl, videoRating, image_url, addedDate, isWatched, addedYear, genre, et duration.

Entité History : Maintient l'historique avec id_History, watched_list, unwatched_list, et id_user.

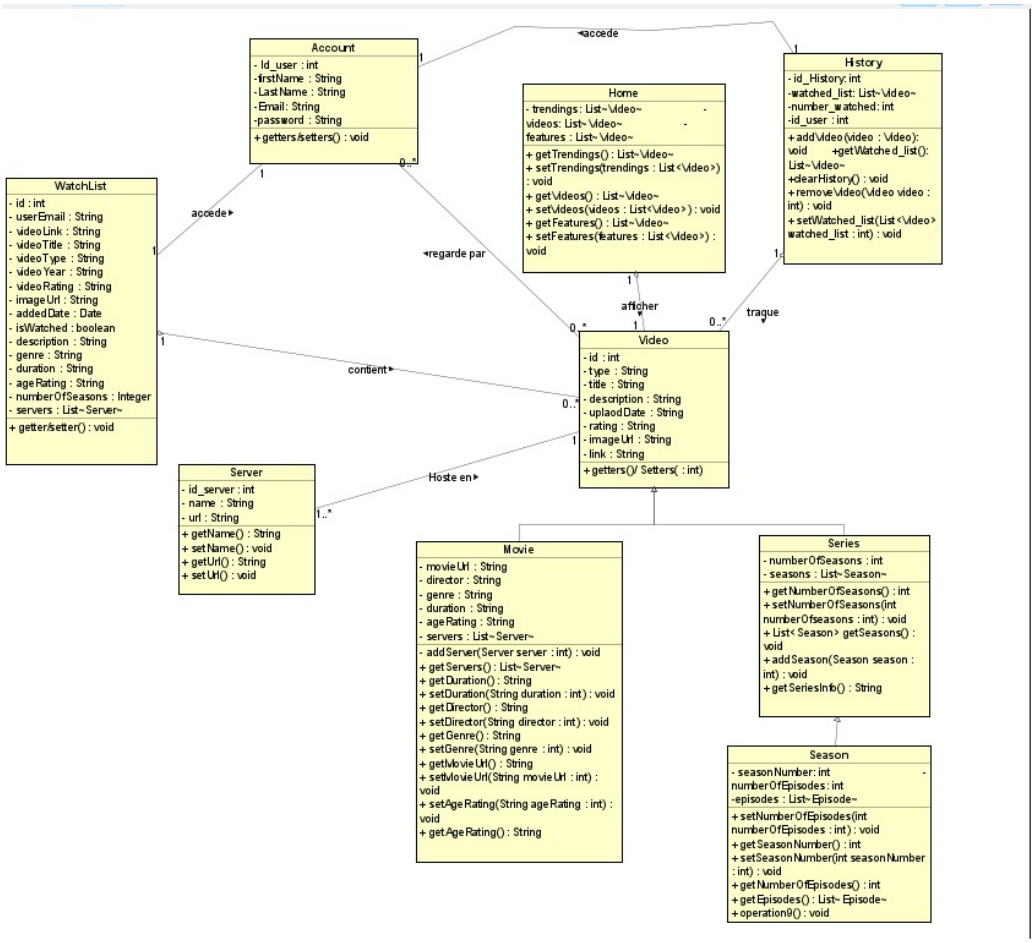

FIGURE 3.1 – Diagramme de classe de siteWeb Wovies

3.2 Diagramme de Cas d'Utilisation

Acteur Visiteur : Parcourir l'accueil, rechercher, filtrer par catégorie, se connecter, créer un compte.

Acteur Utilisateur Connecté : Toutes les fonctionnalités du visiteur plus se déconnecter, gérer son compte, consulter historique/watchlist, regarder une vidéo, ajouter à watchlist/historique, choisir serveur.

Acteur System : Services de scraping pour récupérer les informations des films et séries depuis des sources externes.

Les relations "include" indiquent des dépendances obligatoires tandis que "extend" représente des extensions optionnelles.

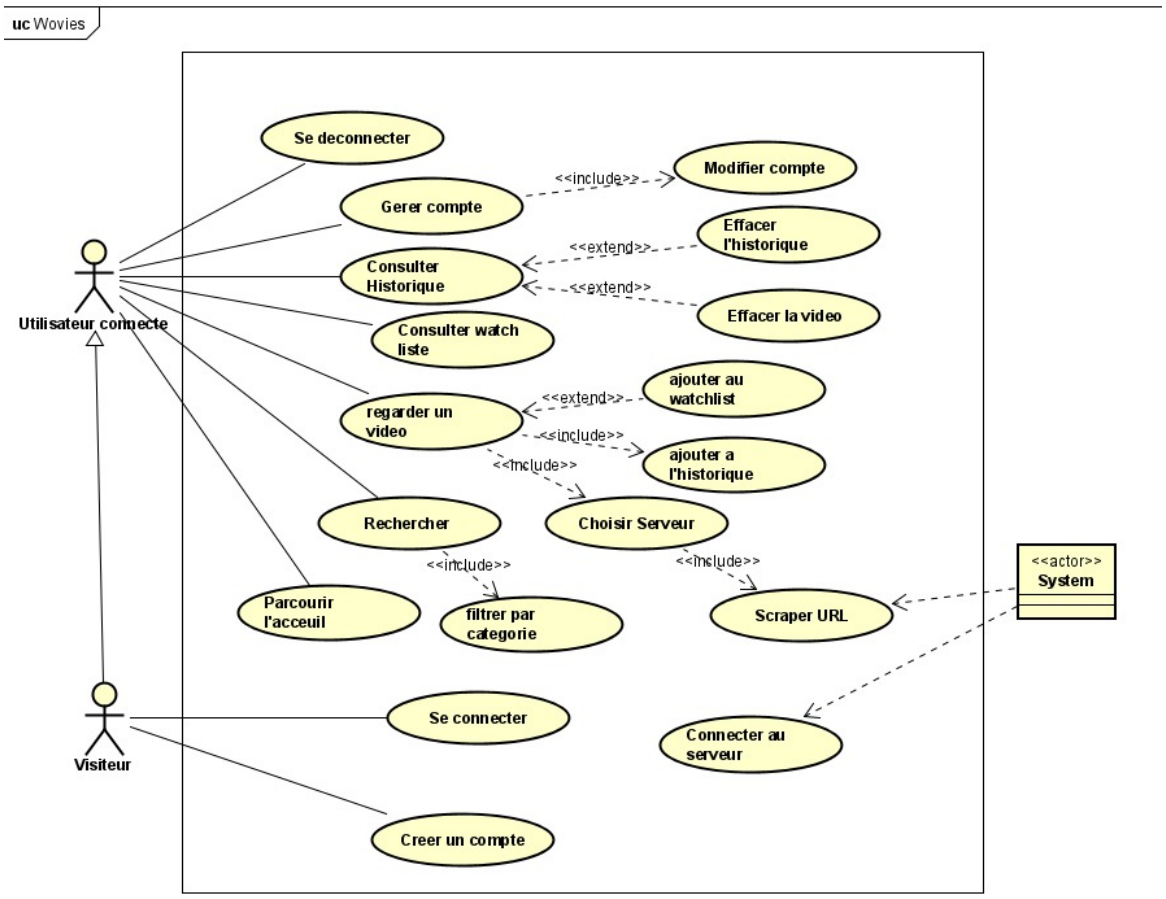

FIGURE 3.2 – Diagramme de cas d'utilisation de site web WOVIES

Chapitre 4

Technologies Utilisées

Vue d'ensemble

Ce projet utilise un ensemble de technologies modernes et éprouvées pour créer une application web robuste et évolutive de streaming vidéo. Cette section détaille les principales technologies, frameworks et outils intégrés dans l'architecture du système.

4.1 Architecture et Framework Web

4.1.1 Java Enterprise Edition (Jakarta EE)

Le projet est développé avec **Jakarta EE**, la plateforme d'entreprise pour Java qui fournit un ensemble standardisé d'API pour le développement d'applications web évolutives. Les composants Jakarta EE utilisés incluent :

- **Servlets** : Gestion des requêtes HTTP côté serveur via des classes comme `AccountServlet`, `SearchServlet`, `WatchServlet`
- **JSP (JavaServer Pages)** : Technologie de présentation pour générer dynamiquement du contenu HTML
- **Sessions HTTP** : Gestion de l'état utilisateur et du cache des résultats de recherche
- **Annotations** : Utilisation d'annotations comme `@WebServlet` pour la configuration déclarative

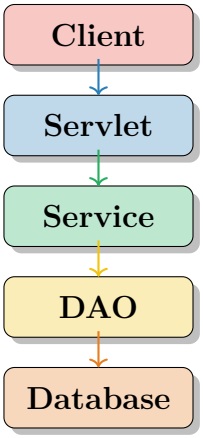

FIGURE 4.1 – Architecture

4.1.2 Architecture MVC (Modèle-Vue-Contrôleur)

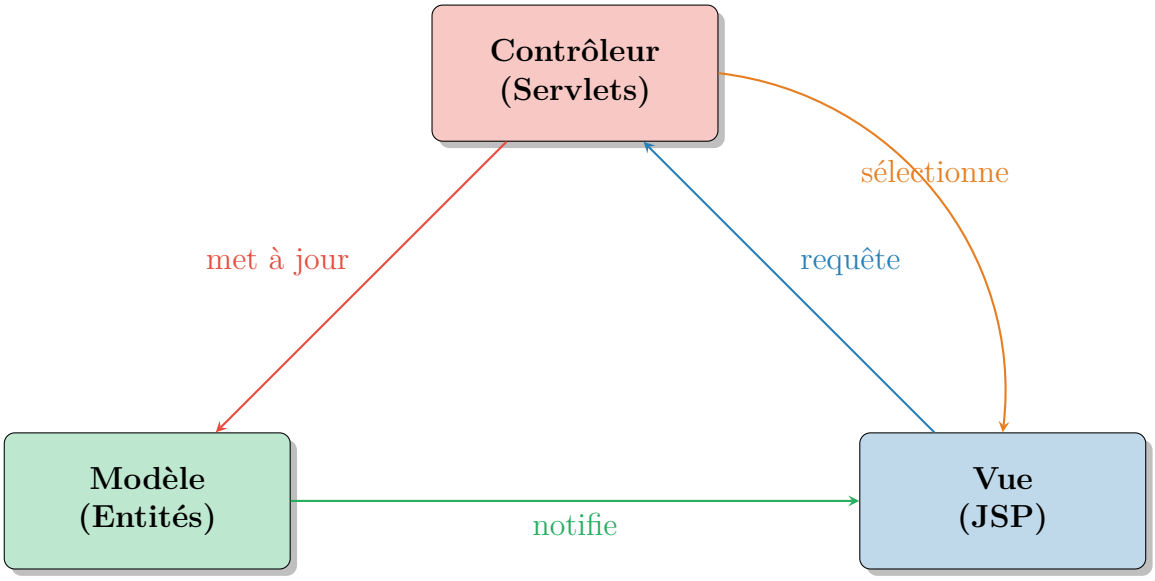

FIGURE 4.2 – Pattern MVC implémenté dans le projet

4.2 Persistance des Données

4.2.1 Hibernate ORM

Hibernate - Mapping Objet-Relationnel

Hibernate est utilisé comme framework de mapping objet-relationnel (ORM) pour gérer la persistance des données avec élégance et efficacité.

Les fonctionnalités exploitées incluent :

- **Annotations JPA**
- **HQL (Hibernate-Query-Language)**
- **SessionFactory**
- **Gestion transactionnelle**
- **Lazy Loading**
- **Relations entre entités**

4.2.2 Pattern DAO (Data Access Object)

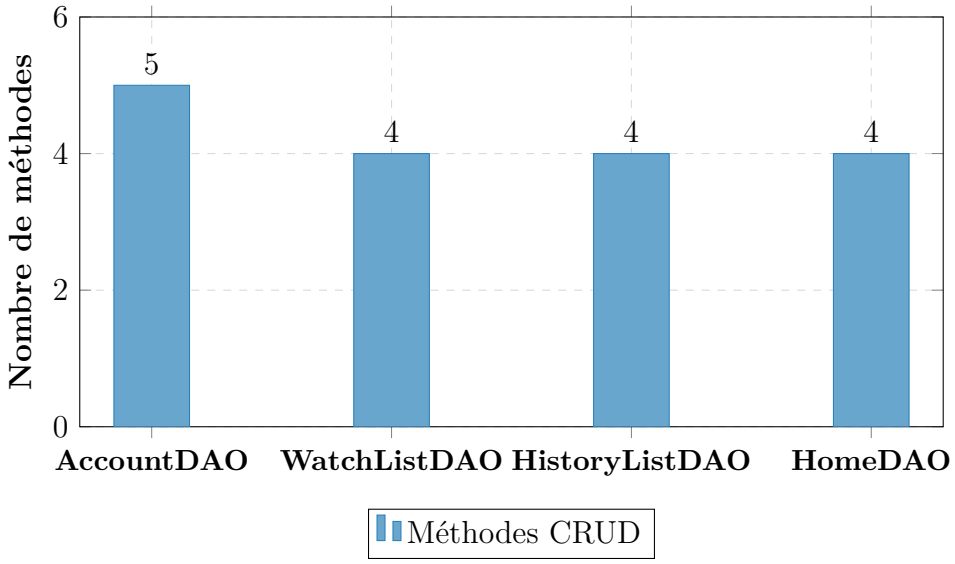

FIGURE 4.3 – Distribution des méthodes dans les classes DAO

4.3 Web Scraping et Intégration d'API

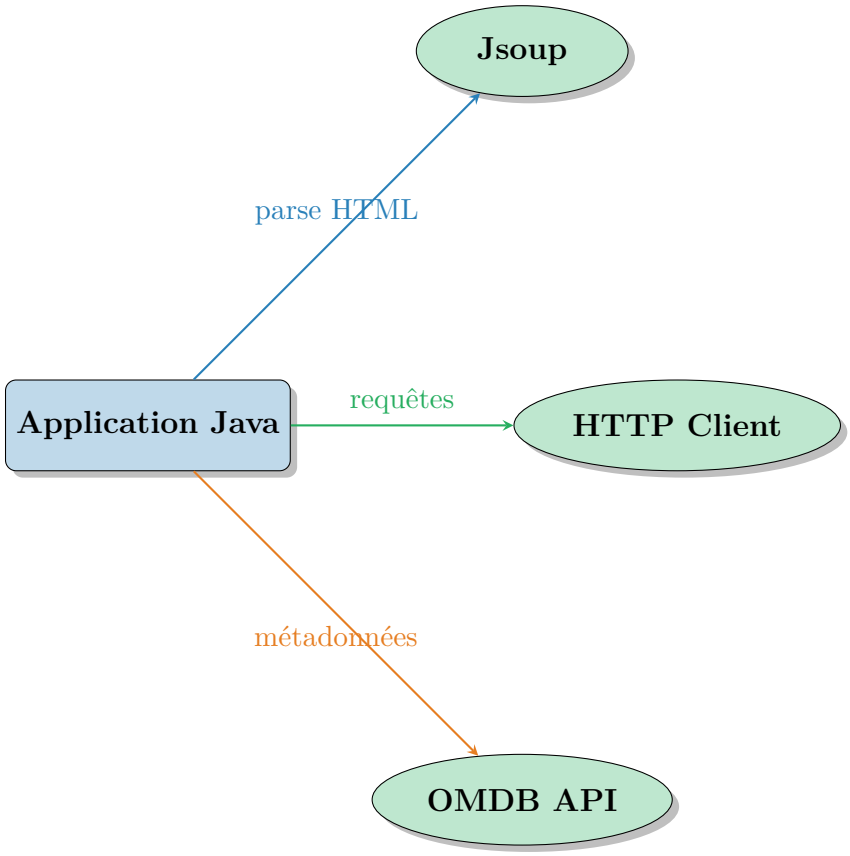

FIGURE 4.4 – Intégrations externes du système

4.3.1 Jsoup

</> Bibliothèque Java pour parser et extraire des données HTML :

- Parser les réponses HTML des sites de streaming
- Extraire les métadonnées vidéo (titres, images, liens)
- Décoder les URLs des serveurs de streaming

4.3.2 HTTP Client

Le projet utilise l'API `java.net.http.HttpClient` introduite dans Java 11 pour :

- Effectuer des requêtes HTTP/HTTPS vers des API externes
- Gérer les cookies et les en-têtes personnalisés
- Effectuer des requêtes asynchrones vers l'API OMDB

4.3.3 OMDB API

🎞 L'API Open Movie Database enrichit les métadonnées vidéo :

- Récupération des notes IMDb
- Extraction des informations détaillées (genre, durée, réalisateur, description)
- Classification automatique entre films et séries

4.4 Traitement et Manipulation de Données

Technologies de Traitement

Gson : Google Gson est utilisé pour le parsing et la sérialisation JSON

Java Streams & Lambda : Utilisation des fonctionnalités modernes de Java 8+ pour le traitement fonctionnel des collections

4.5 Couche Service

Service	Responsabilités
AccountService	Authentification, inscription, changement de mot de passe
SearchService	Recherche vidéo avec cache en session et filtrage
WatchService	Récupération des serveurs de streaming
WatchListService	Gestion de la liste de lecture utilisateur
HistoryListService	Suivi de l'historique de visionnage
HomeService	Agrégation des contenus pour la page d'accueil

TABLE 4.1 – Services métier de l'application

4.6 Fonctionnalités Avancées

4.6.1 Système de Cache

Optimisation des Performances

Un système de cache basé sur les sessions HTTP optimise considérablement les performances en évitant les requêtes répétitives vers les APIs externes.

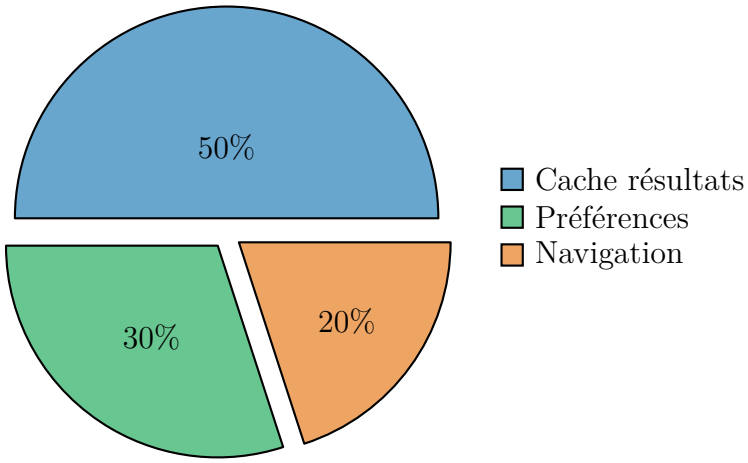

FIGURE 4.5 – Répartition des données en cache

4.6.2 Filtrage et Tri Dynamiques

Options de filtrage :
- Films uniquement
- Séries uniquement
- Contenus récents
- Non visionnés

Options de tri :
- Par titre (A-Z)
- Par note (IMDb)
- Par année
- Par date d'ajout

4.7 Sécurité

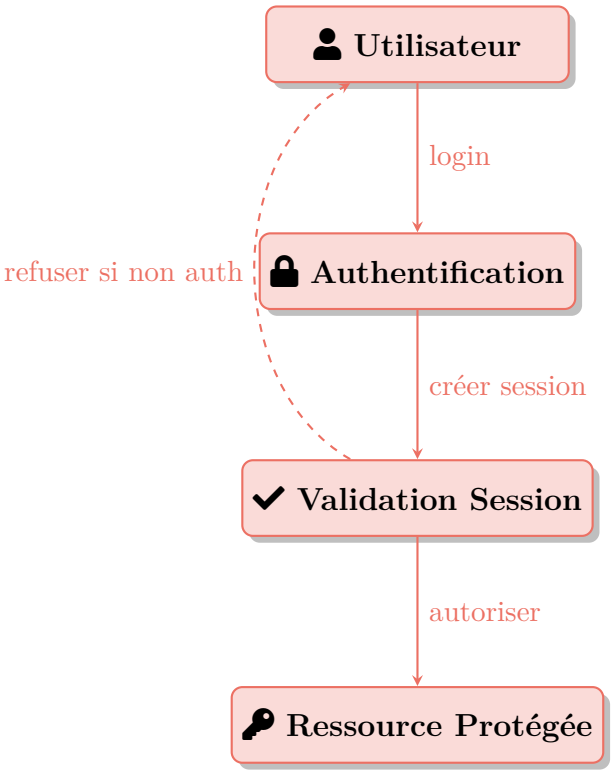

FIGURE 4.6 – Flux de sécurité de l'application

Mécanismes de Sécurité

Authentification et Autorisation :
- **Vérification de l'authentification utilisateur dans chaque servlet**
- **Redirection automatique vers la page de connexion pour les utilisateurs non authentifiés**
- **Validation des permissions avant l'accès aux ressources**

Gestion des Mots de Passe :
- **Validation des mots de passe actuels avant modification**
- **Gestion sécurisée des erreurs d'authentification**
- **Codes de statut HTTP appropriés pour les erreurs**

4.8 Modèle de Données

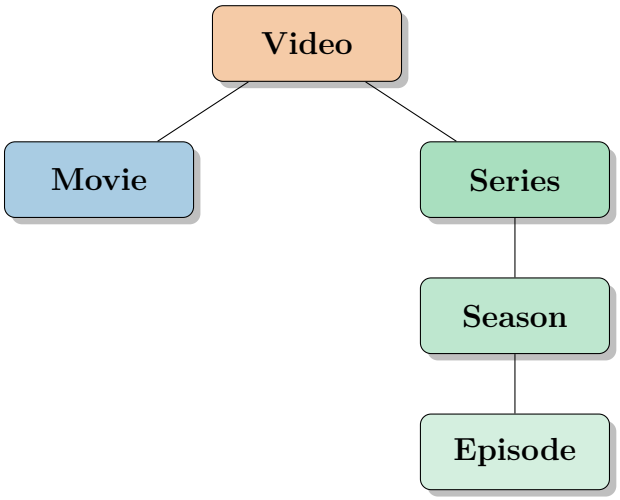

FIGURE 4.7 – Hiérarchie d'héritage des entités vidéo

4.9 Stack Technologique Complète

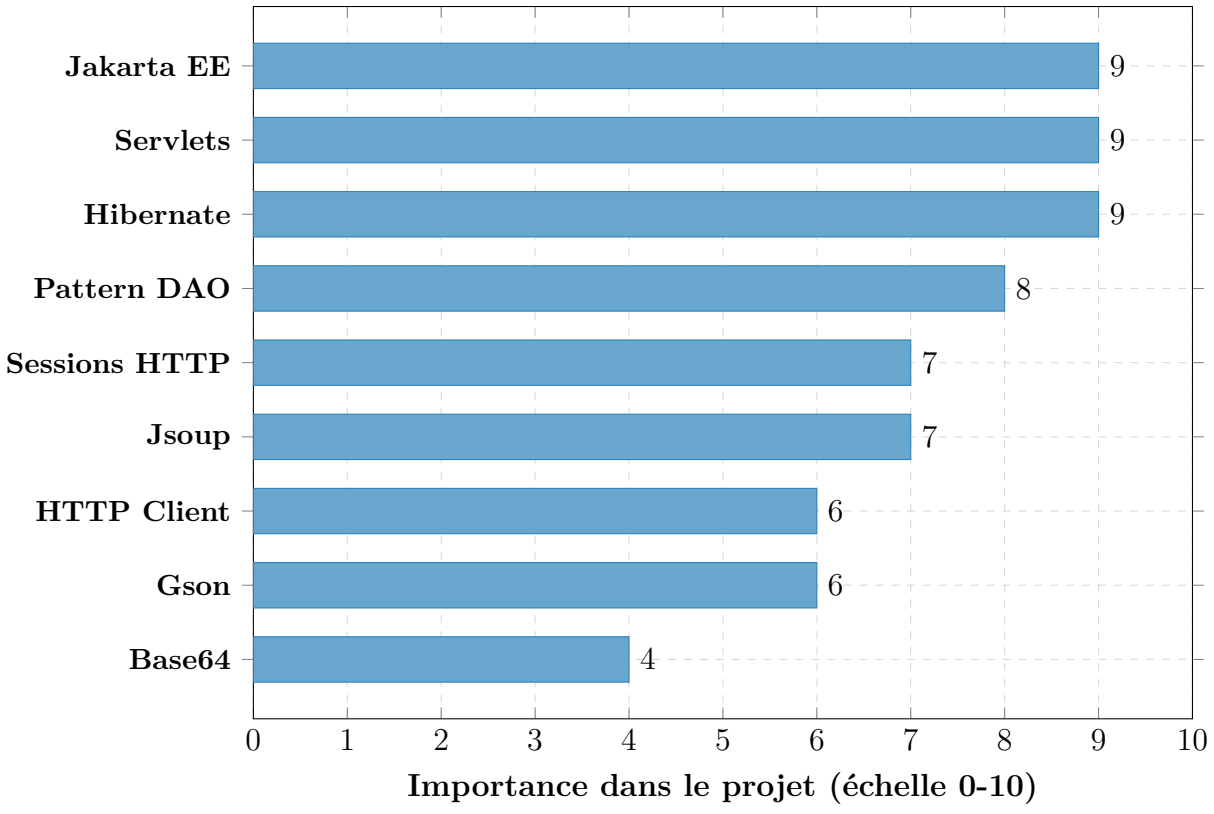

FIGURE 4.8 – Importance relative des technologies dans le projet

4.10 Points Forts de l'Architecture

Avantages Architecturaux

1. **Séparation des responsabilités : Architecture en couches claire (Présentation, Service, DAO, Modèle)**
2. **Réutilisabilité : Services et DAO réutilisables dans différents contextes**
3. **Maintenabilité : Code organisé en packages logiques avec nomenclature cohérente**
4. **Extensibilité : Facilité d'ajout de nouvelles fonctionnalités via l'architecture modulaire**
5. **Performance : Système de cache intelligent et chargement paresseux des données**
6. **Standards industriels : Utilisation de Jakarta EE, JPA, et patterns établis**

4.11 Statistiques du Projet

Métrique	Valeur
Nombre de Servlets	8
Nombre de Services	6
Nombre de DAO	4
Nombre d'Entités	11
APIs Externes	2
Packages	7

TABLE 4.2 – Métriques du projet

Architecture Robuste et Moderne pour le Streaming Vidéo

4.12 Diagramme de Déploiement

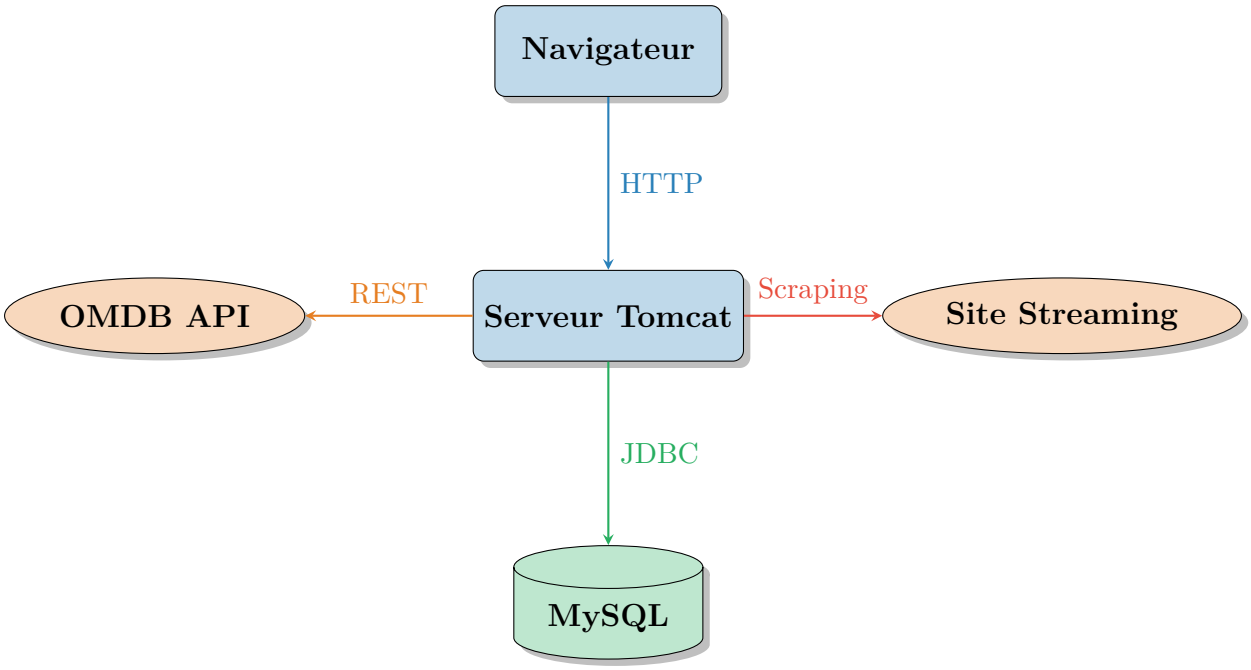

FIGURE 4.9 – Architecture de déploiement du système

Conclusion

Synthèse Technologique

Ce projet de streaming vidéo démontre l'utilisation efficace d'un ensemble cohérent de technologies modernes Java EE. L'architecture en couches, les patterns de conception éprouvés (MVC, DAO, Service), et l'intégration intelligente d'APIs externes créent une application robuste, maintenable et évolutive.

Points clés :

- **Architecture modulaire et séparation des préoccupations**
- **Utilisation de standards industriels (Jakarta EE, Hibernate)**
- **Optimisations de performance via cache et lazy loading**
- **Intégration fluide avec des services externes**
- **Sécurité renforcée et gestion des sessions**

Cette stack technologique offre une base solide pour le développement futur et la maintenance à long terme de l'application.

Chapitre 5

Difficultés Rencontrées

⚠ Vue d'ensemble des défis

Le développement de ce projet a présenté plusieurs défis techniques et architecturaux. Cette section analyse les principales difficultés rencontrées et les solutions adoptées pour les surmonter.

5.1 Web Scraping et Limitations Techniques

5.1.1 Limitation de Jsoup - Pages Statiques Uniquement

Problème identifié :

Jsoup, bien qu'excellent pour parser du HTML, ne peut pas exécuter du JavaScript. De nombreux sites de streaming modernes utilisent le rendu côté client avec JavaScript pour charger dynamiquement le contenu, ce qui rend le scraping impossible avec Jsoup seul.

- ✖ Impossible d'extraire les données chargées via AJAX
- ✖ Les lecteurs vidéo dynamiques ne sont pas accessibles
- ✖ Le contenu généré par JavaScript reste invisible

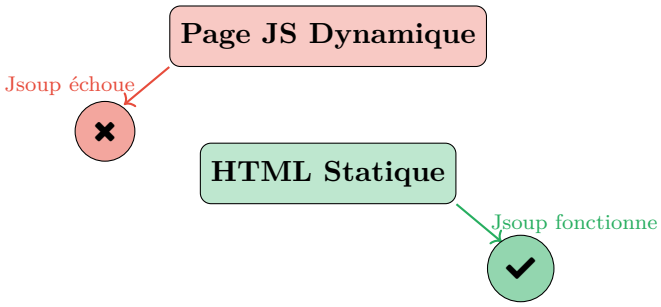

FIGURE 5.1 – Limitation de Jsoup

Solution adoptée :

Ciblage de sites utilisant du HTML statique ou semi-statique. Le site sélectionné (Wecima) génère son contenu côté serveur, permettant à Jsoup de parser efficacement la structure HTML.

5.1.2 Alternatives Envisagées

Solution	Complexité	Performance	Adopté
Jsoup (HTML statique)	✔ Faible	✔ Excellent	✔ Oui
Selenium WebDriver	⚠ Élevée	✖ Lent	✖ Non
Playwright/Puppeteer	⚠ Élevée	⚠ Moyen	✖ Non
API officielle	✔ Faible	✔ Optimal	✖ Indisponible

TABLE 5.1 – Comparaison des solutions de scraping

5.2 Performance et Optimisation

5.2.1 Problème de Latence des Requêtes Externes

Défi Performance

Chaque recherche nécessite plusieurs requêtes HTTP externes :

- 1 requête vers le site de streaming (scraping)
- N requêtes vers l'API OMDB (une par résultat)
- Temps de réponse total : 3-8 secondes par recherche

Impact sur l'expérience utilisateur :

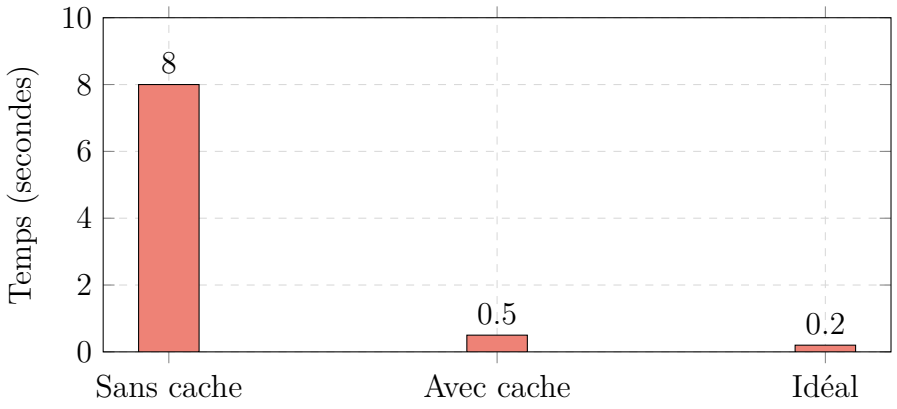

FIGURE 5.2 – Impact du cache sur les temps de réponse

Solutions implémentées :

1. **Cache en session HTTP :** Stockage des résultats de recherche pour éviter les requêtes répétitives
2. **Limitation des requêtes OMDB :** Filtrage préalable des résultats avant enrichissement
3. **Timeout optimisé :** Configuration de timeouts pour éviter les blocages prolongés

5.3 Gestion de l'Encodage et des Caractères Spéciaux

5.3.1 Contenu Multilingue (Arabe/Anglais)

Problème : Le site source contient du contenu en arabe et en anglais, avec des encodages variés.

- ⚠ Problèmes d'encodage UTF-8 lors du parsing
- ⚠ Titres mixtes (arabe + anglais) difficiles à traiter
- ⚠ Normalisation nécessaire pour les recherches OMDB

Solution :
Nettoyage systématique des titres pour extraire uniquement les noms en anglais avant la requête OMDB.

5.4 Décodage des URLs de Streaming

5.4.1 Obfuscation des Liens Serveurs

Sécurité Anti-Scraping

Les sites de streaming protègent leurs liens serveurs par encodage Base64 et obfuscation JavaScript pour empêcher l'extraction automatique.

Méthode de protection rencontrée :

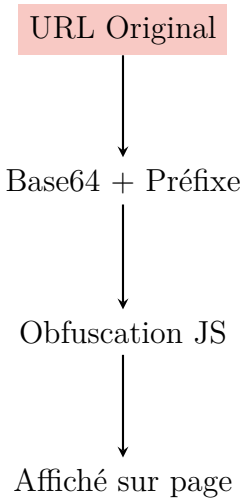

FIGURE 5.3 – Processus d'obfuscation des URLs

Solution de décodage :

```
String encodedUrl = element.attr("data-url");
String cleaned = encodedUrl.replaceAll("\\+", "");
String base64String = "aHR0c" + cleaned;
byte[] decodedBytes = Base64.getDecoder().decode(base64String);
String decodedUrl = new String(decodedBytes, StandardCharsets.UTF_8);
```

Ingénierie inverse du processus d'encodage pour extraire les URLs réelles.

5.5 Architecture et Conception

5.5.1 Gestion de la Hiérarchie Video/Movie/Series

Défi : Modéliser correctement l'héritage entre les différents types de vidéos tout en maintenant la persistance Hibernate.

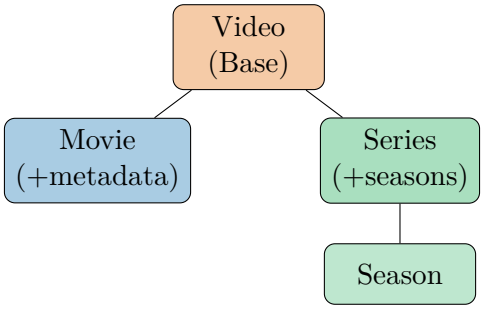

FIGURE 5.4 – Hiérarchie d'héritage complexe

Problèmes rencontrés :

— Mapping Hibernate avec héritage (Table per Class vs. Single Table)

— Sérialisation/désérialisation lors du stockage en session

— Cast d'objets lors de la récupération depuis le cache

5.5.2 Synchronisation Cache et Base de Données

Cohérence des Données

Challenge : Maintenir la cohérence entre les résultats cachés en session et les données persistées en base de données (watchlist, history).

Scénario problématique :

1. Utilisateur recherche "Inception" → résultats cachés
2. Utilisateur ajoute à la watchlist → sauvegardé en DB
3. Résultats de recherche expirés → nouveau cache
4. IDs des vidéos changent → incohérence potentielle

Solution : Utilisation du `videoLink` comme identifiant unique au lieu de l'ID généré dynamiquement.

5.6 Gestion des Sessions et Cookies

5.6.1 Maintien de la Session de Scraping

Les sites web modernes utilisent des cookies et des tokens anti-bot pour détecter le scraping automatisé.

```
.header("cookie", Constants.COOKIE)
.header("X-Requested-With", "XMLHttpRequest")
.header("Accept-Language", "en-US,en;q=0.8")
```

Défis :

— Cookies expirés nécessitant mise à jour manuelle

— Détection de bot basée sur les patterns de requêtes

— Cloudflare protection nécessitant des headers spécifiques

5.7 Limitations de l'API OMDB

Limitation	Impact
1000 requêtes/jour	Nécessité de gérer un quota d'utilisation
Recherche par titre uniquement	Difficulté avec les titres en arabe
Pas de support multilingue	Nettoyage manuel des titres requis
Données parfois incomplètes	Gestion des valeurs "N/A"

TABLE 5.2 – Contraintes de l'API OMDB

5.8 Résumé des Solutions

Approches Réussies

1. **Cache intelligent : Réduction drastique des temps de réponse (8s → 0.5s)**
2. **Sélection de sources compatibles : Ciblage de sites avec HTML statique**
3. **Décodage personnalisé : Reverse engineering des mécanismes de protection**
4. **Nettoyage des données : Normalisation des titres multilingues**
5. **Identifiants stables : Utilisation d'URLs comme clés uniques**
6. **Gestion des erreurs : Fallbacks et valeurs par défaut pour données manquantes**

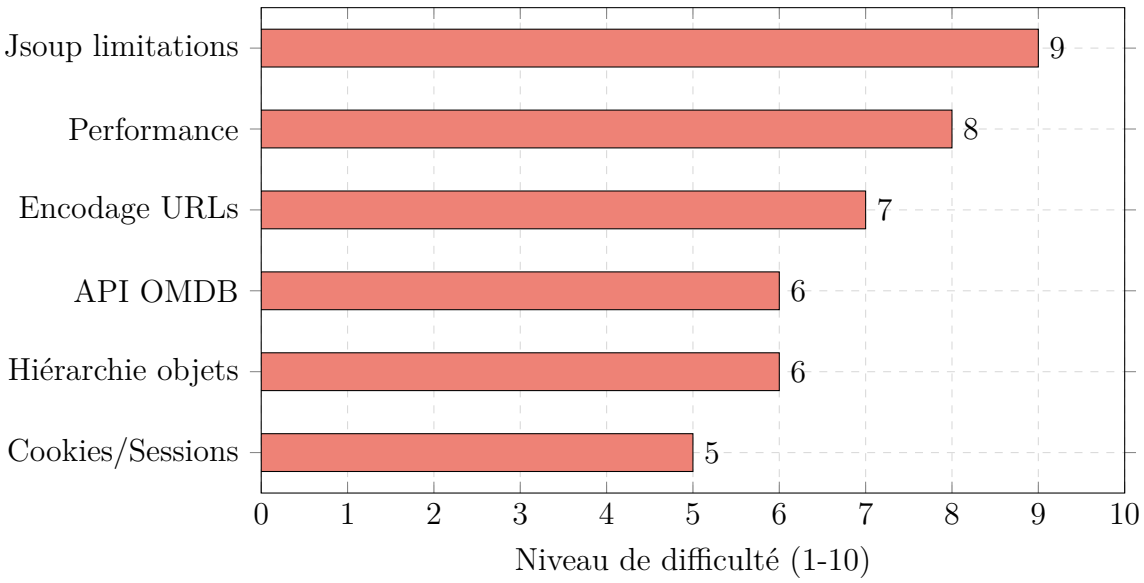

FIGURE 5.5 – Niveau de difficulté des principaux défis

5.9 Leçons Apprises

Points positifs :

- ✔ Expérience en web scraping
- ✔ Optimisation performance
- ✔ Résolution de problèmes
- ✔ Architecture résiliente

Améliorations futures :

- Utiliser Selenium pour JS
- Implémenter un cache Redis
- Rate limiting intelligent
- Monitoring des erreurs

Les défis techniques ont renforcé la robustesse du système

Chapitre 6

Conclusion

Bilan du Projet

Ce projet de plateforme de streaming vidéo représente l'aboutissement d'un travail approfondi combinant technologies modernes, architectures logicielles éprouvées et résolution de défis techniques complexes. Cette conclusion récapitule les accomplissements, les apprentissages et les perspectives d'évolution du système.

6.1 Objectifs Atteints

6.1.1 Réalisations Techniques

Le projet a réussi à mettre en place une application web complète et fonctionnelle offrant les fonctionnalités suivantes :

- ✔ Système d'authentification sécurisé
- ✔ Moteur de recherche avec filtres avancés
- ✔ Lecture de vidéos multi-serveurs
- ✔ Gestion de watchlist personnalisée
- ✔ Historique de visionnage
- ✔ Interface responsive et intuitive
- ✔ Cache de performance optimisé
- ✔ Intégration APIs externes

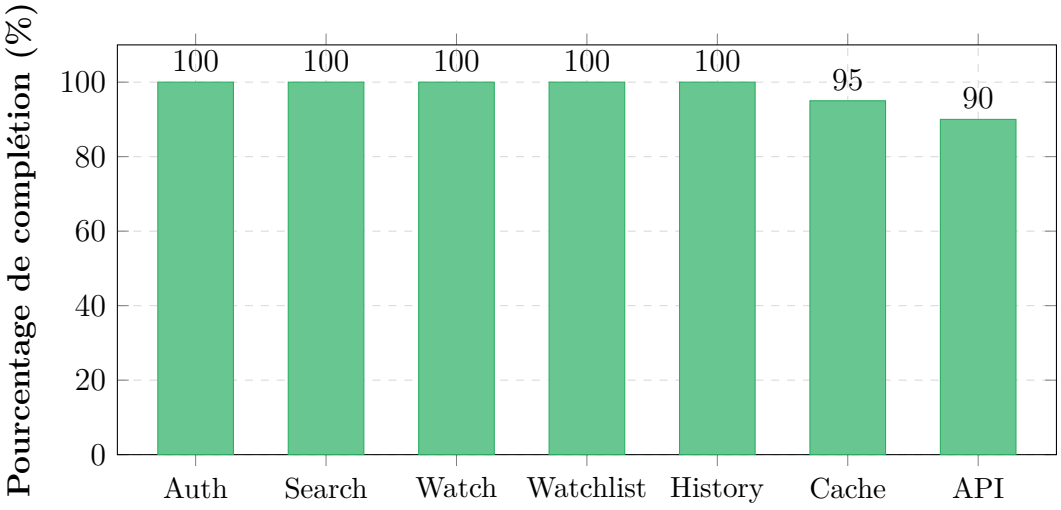

FIGURE 6.1 – Taux de complétion des fonctionnalités principales

6.1.2 Architecture Logicielle Robuste

L'architecture mise en place répond aux standards industriels et garantit :

Séparation des préoccupations : L'architecture MVC avec couches Service et DAO assure une organisation claire du code et facilite la maintenance.

Évolutivité : La structure modulaire permet l'ajout facile de nouvelles fonctionnalités sans refonte majeure.

Maintenabilité : Le code bien organisé en packages et l'utilisation de patterns éprouvés facilitent les interventions futures.

Performance : Le système de cache et les optimisations implémentées garantissent des temps de réponse acceptables.

6.2 Compétences Développées

6.2.1 Compétences Techniques

Ce projet a permis de développer et consolider un large éventail de compétences techniques :

Domaine	Compétences Acquises
</> Backend Java	Jakarta EE, Servlets, Sessions HTTP, Hibernate ORM, Pattern DAO
Persistance	Hibernate, JPA, Mapping objet-relationnel, Transactions, HQL
Intégration	Web scraping (Jsoup), APIs REST (OMDB), HTTP Client Java
Architecture	Pattern MVC, Architecture en couches, Séparation des préoccupations
Performance	Mise en cache, Optimisation requêtes, Lazy loading
Sécurité	Authentification, Gestion sessions, Protection ressources
Debug	Résolution problèmes encodage, Reverse engineering, Gestion erreurs

Table 6.1 – Compétences techniques développées

6.2.2 Compétences Transversales

Au-delà des aspects techniques, le projet a développé des compétences essentielles :

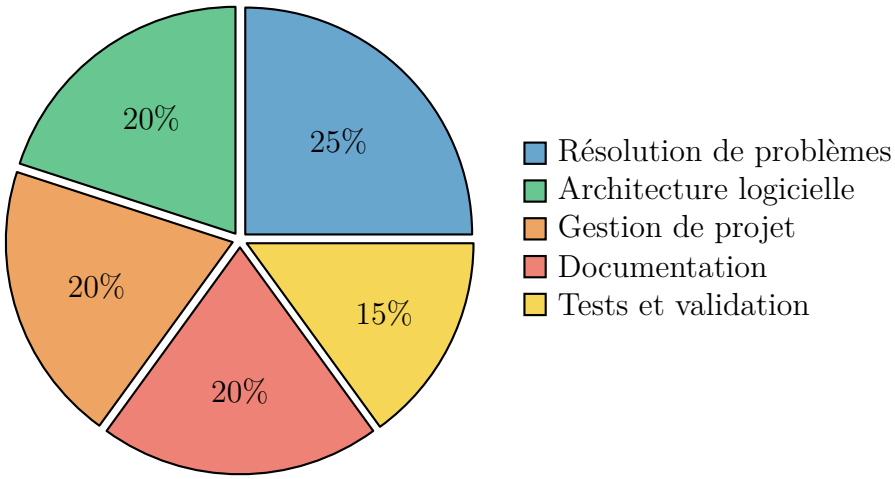

FIGURE 6.2 – Répartition des compétences transversales

6.3 Défis Surmontés

Le développement a été marqué par plusieurs défis techniques significatifs qui ont été relevés avec succès :

1. **Limitation du scraping statique** : Adaptation de la stratégie en ciblant des sources compatibles avec Jsoup
2. **Performance des requêtes externes** : Implémentation d'un système de cache réduisant les temps de réponse de 8s à 0.5s
3. **Obfuscation des URLs** : Décodage réussi des mécanismes de protection par reverse engineering
4. **Gestion multilingue** : Normalisation efficace des contenus arabe/anglais
5. **Cohérence des données** : Synchronisation cache/base de données via identifiants stables

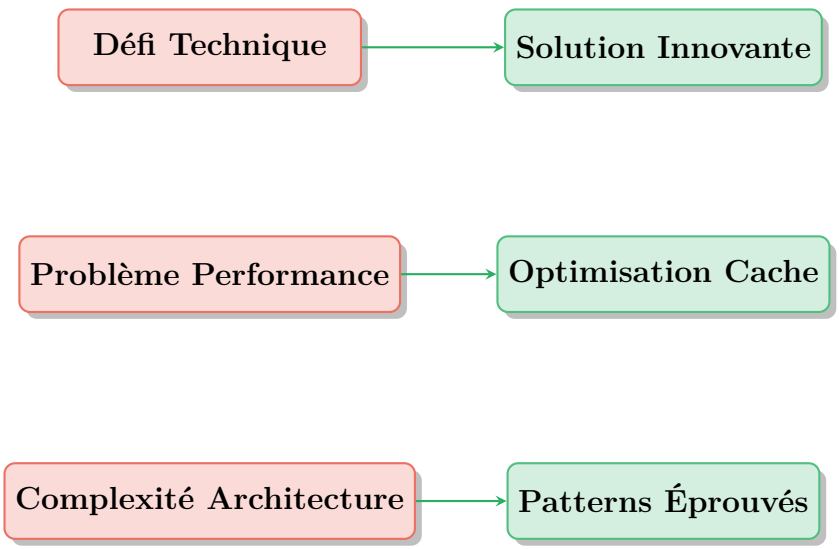

FIGURE 6.3 – Approche de résolution des défis

6.4 Limitations Actuelles

Malgré les réussites, certaines limitations subsistent et ouvrent des pistes d'amélioration :

Points d'Amélioration

- **Scraping limité au HTML statique : Impossibilité de traiter les sites avec JavaScript dynamique**
- **Dépendance aux sources externes : Vulnérabilité aux changements de structure des sites scrapés**
- **Quota API OMDB : Limitation de 1000 requêtes/jour**
- **Absence de cache persistant : Cache limité à la session utilisateur**
- **Gestion manuelle des cookies : Nécessité de mise à jour périodique**

6.5 Perspectives d'Évolution

6.5.1 Améliorations Court Terme

Fonctionnalités :

- Système de recommandations
- Notation et commentaires
- Partage social
- Mode hors-ligne

Techniques :

- Cache Redis distribué
- API REST documentée
- Tests automatisés
- Monitoring des performances

6.5.2 Évolutions Long Terme

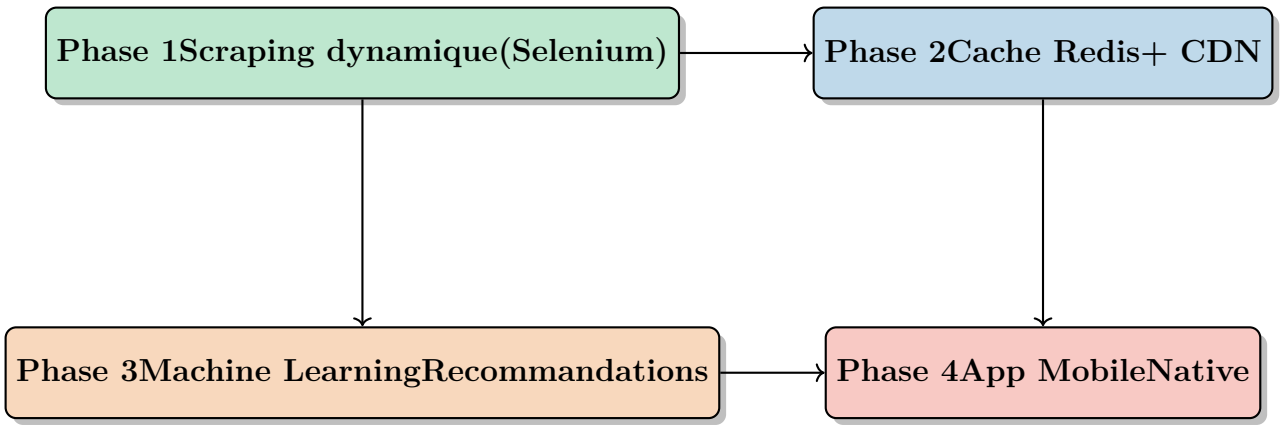

FIGURE 6.4 – Feuille de route des évolutions futures

6.5.3 Technologies Futures à Intégrer

Technologie	Priorité	Objectif
Selenium WebDriver	Haute	Scraping de sites JavaScript dynamiques
Redis Cache	Haute	Cache distribué et persistant
Spring Boot	Moyenne	Modernisation du framework
Docker	Moyenne	Containerisation et déploiement
Elasticsearch	Basse	Recherche full-text avancée
React/Angular	Basse	Interface utilisateur moderne

TABLE 6.2 – Technologies envisagées pour les évolutions

6.6 Apport Pédagogique

6.6.1 Mise en Pratique des Connaissances

Ce projet a permis de mettre en application concrète les concepts théoriques étudiés :

- **Programmation orientée objet** : Héritage, polymorphisme, encapsulation
- **Bases de données** : Modélisation, normalisation, requêtes SQL/HQL
- **Architecture logicielle** : Patterns MVC, DAO, Service Layer
- **Développement web** : Servlets, JSP, Sessions HTTP
- **Intégration continue** : Gestion des dépendances, build automation

6.6.2 Expérience Professionnelle

Le projet simule un environnement de développement professionnel :

— **Travail sur un projet de taille significative avec architecture complexe**
— **Gestion des contraintes techniques et des limitations externes**
— **Documentation technique et choix architecturaux justifiés**
— **Résolution de problèmes réels sans solution toute faite**
— **Respect des standards et bonnes pratiques de l'industrie**

6.7 Synthèse Finale

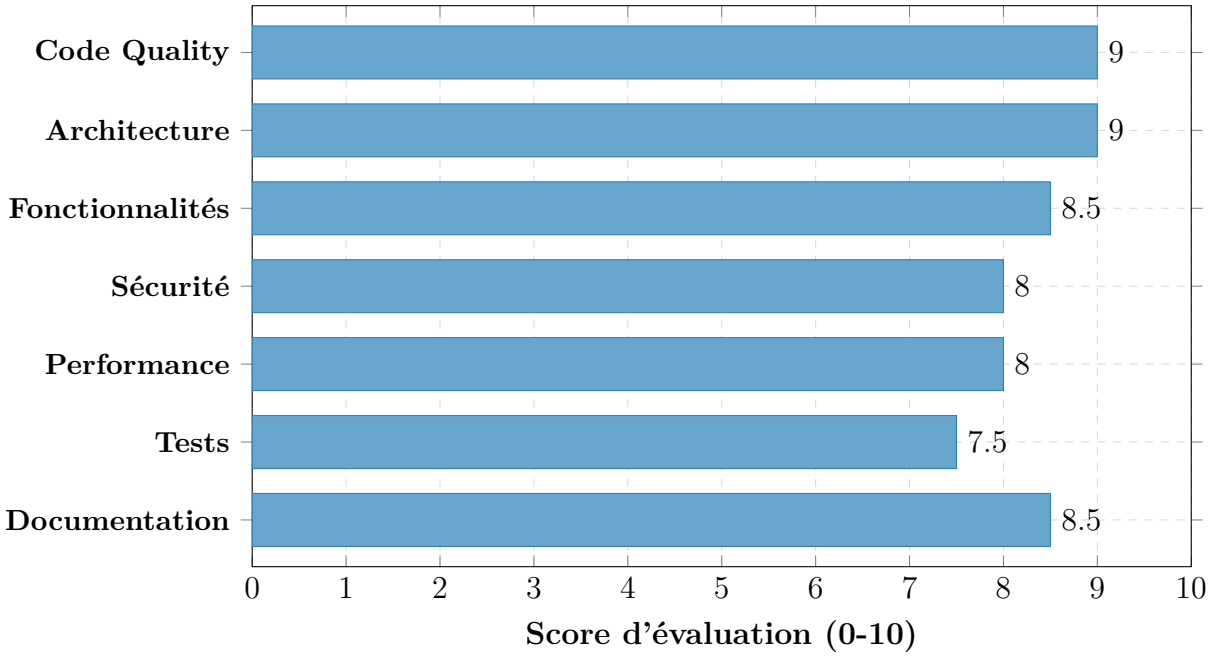

FIGURE 6.5 – Évaluation globale du projet

Conclusion Générale

Ce projet de plateforme de streaming vidéo a atteint ses objectifs principaux en livrant une application fonctionnelle, performante et bien architecturée. Les défis techniques rencontrés ont été autant d'opportunités d'apprentissage et d'innovation.

L'utilisation de technologies modernes (Jakarta EE, Hibernate, Jsoup) combinée à des patterns architecturaux éprouvés (MVC, DAO) a permis de construire une base solide pour de futures évolutions. Le système de cache intelligent, le scraping efficace et l'intégration fluide d'APIs externes démontrent la maîtrise des concepts avancés du développement web.

Au-delà de la réalisation technique, ce projet a permis de développer des compétences essentielles en résolution de problèmes, en architecture logicielle et en gestion de projet. Les limitations identifiées ouvrent des perspectives d'amélioration claires, notamment l'adoption de Selenium pour le scraping dynamique et l'implémentation d'un cache Redis distribué.

En définitive, ce projet constitue une expérience complète et enrichissante qui prépare efficacement aux défis du développement d'applications web professionnelles.

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué, de près ou de loin, à la réussite de ce projet. Je remercie particulièrement mon encadrant, Mme. Zahra Bnider, pour ses conseils et son accompagnement, ainsi que mes collègues pour leurs retours et l'ensemble de la communauté open-source pour les outils et bibliothèques utilisés.