Soutenance de Projet

SAE-821 Développement d'un moteur de recommandations respectueux de la vie privée

Université de Toulon, La Garde

2 Juin 2025

Sommaire

Introduction

Introduction

- Les moteurs de recommandation sont omniprésents : streaming, e-commerce, réseaux sociaux.
- Leur rôle : proposer du contenu pertinent selon les préférences, l'historique ou les comportements d'utilisateurs similaires.
- Cette efficacité repose sur l'analyse de grandes quantités de données personnelles, souvent collectées de manière invisible.
- Cela soulève des enjeux majeurs de confidentialité et de respect de la vie privée.

Problématique

Problématique

★ **Problème central :** *Comment proposer des recommandations pertinentes sans compromettre la vie privée des utilisateurs ?*

✗ **Limites actuelles :**

- Solutions **centralisées** : profilage, tracking, fuites de données.
- Collecte massive de données personnelles.

▲ **Deux approches de protection :**

- **Differential Privacy** : bruit pour anonymiser.
- **Cryptographie homomorphe** : calcul sur données chiffrées.

✻ **Objectif :** Développer un moteur de recommandation basé sur `MovieLens`, tout en garantissant la confidentialité.

Conception UML

User Story

Le point de départ de notre projet repose sur une vision centrée utilisateur. La user story suivante résume notre objectif principal :

« En tant qu'utilisateur, je veux recevoir des recommandations personnalisées sans que mes données personnelles soient stockées en ligne. »
« En tant qu'utilisateur, je veux pouvoir consulter mes avis sur les films que j'ai vus, sans que mes données personnelles soient stockées en ligne. »
« En tant qu'utilisateur, je veux pouvoir rechercher des films par titre/genres, sans que mes données personnelles soient stockées en ligne. »

Cette phrase illustre une exigence fondamentale : proposer un service personnalisé tout en assurant une protection forte de la vie privée.

Diagramme de cas d'utilisation

Ce qui nous donne le diagramme de cas d'utilisation suivant pour un utilisateur :

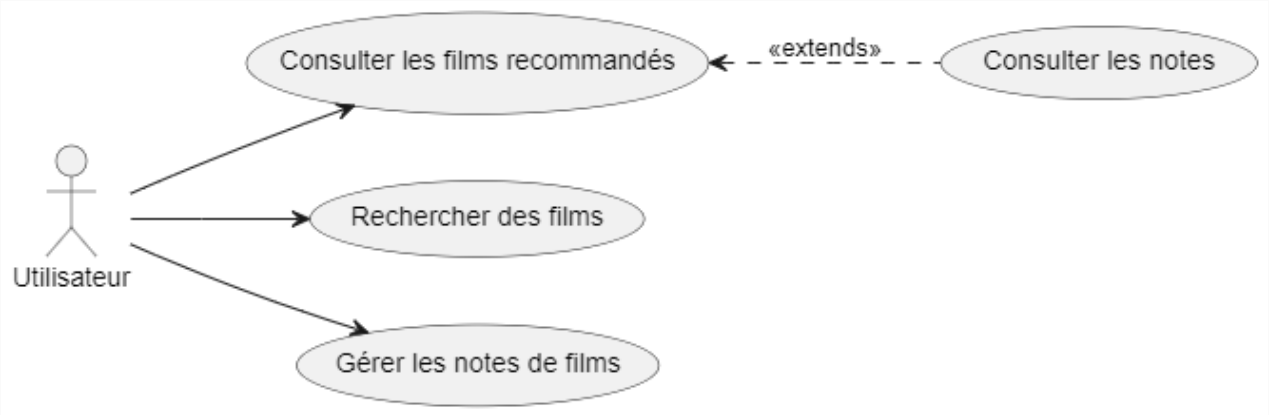


Diagramme de cas d'utilisation de l'utilisateur

Diagramme de cas d'utilisation

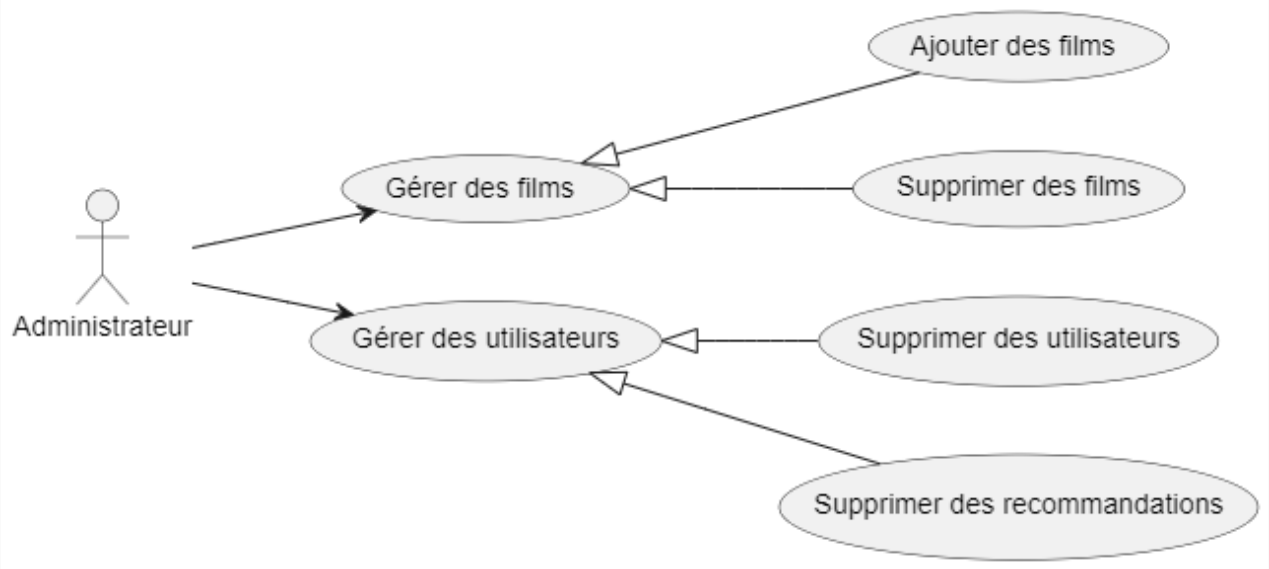


Diagramme de cas d'utilisation de l'administrateur

Fonctionnalités

Fonctionnalités – Techniques

D'un point de vue technique, notre solution repose sur une architecture modulaire et des technologies robustes :

- **Filtrage collaboratif par SVD** : permet de réduire la dimensionnalité et d'identifier les similarités entre utilisateurs/items.
- **Bases de données** : utilisation de deux bases de données : PostgreSQL pour la manipulation et noSql pour l'accès rapide à la donnée
- **Differential Privacy** : ajout de bruit contrôlé aux données lors des échanges avec le serveur de recommandation et sélection biaisée pour la recommendation à froid.
- **Communication sécurisée par endpoints** : les échanges entre le back-end (serveur Quarkus) et le front-end sont encapsulés dans des requêtes REST sécurisées.
- **Interface avec React** : permet d'afficher dynamiquement les recommandations dans un environnement web léger.

Architecture du système

Diagramme de classes

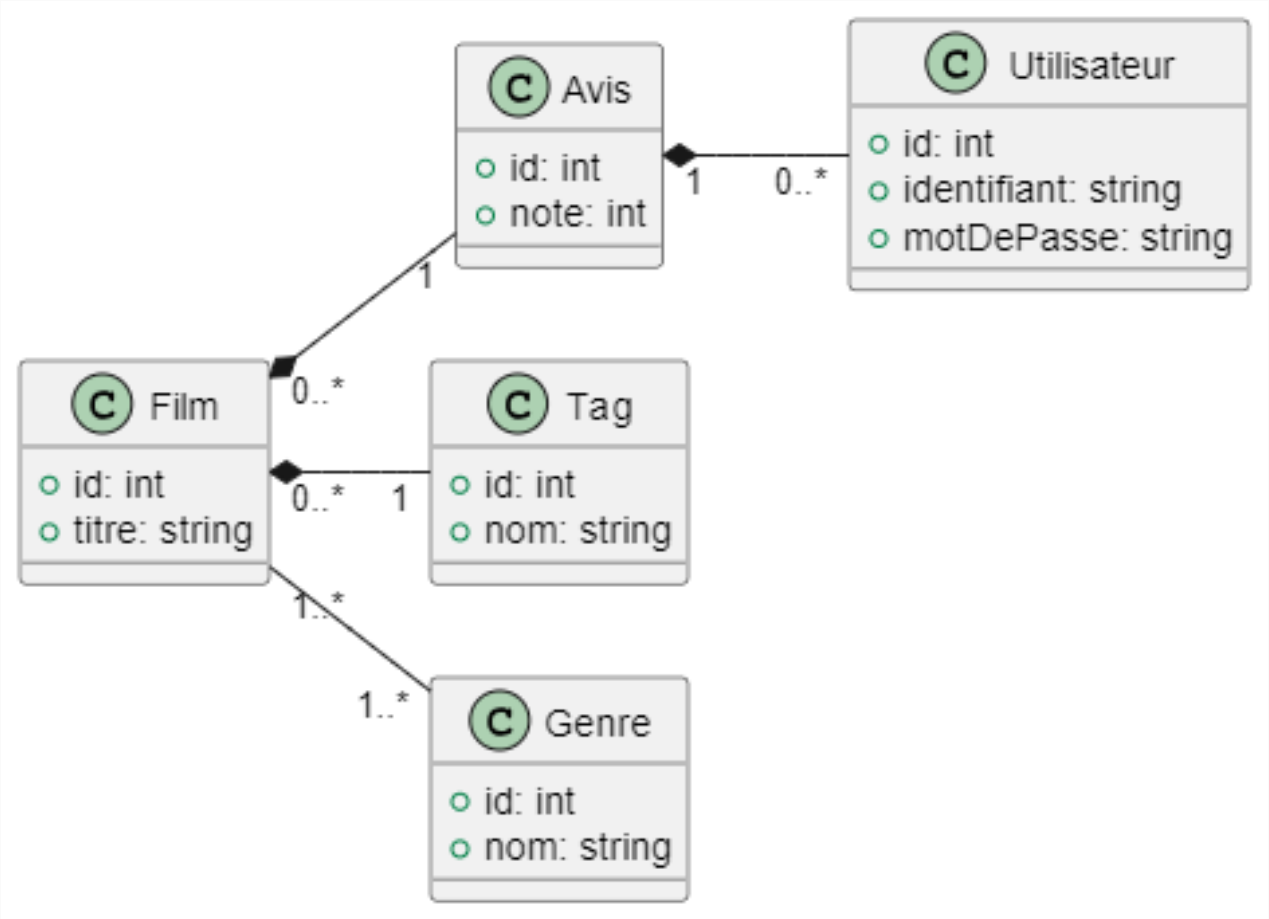


Figure 1: Diagramme de classes du système

Diagramme de deploiement

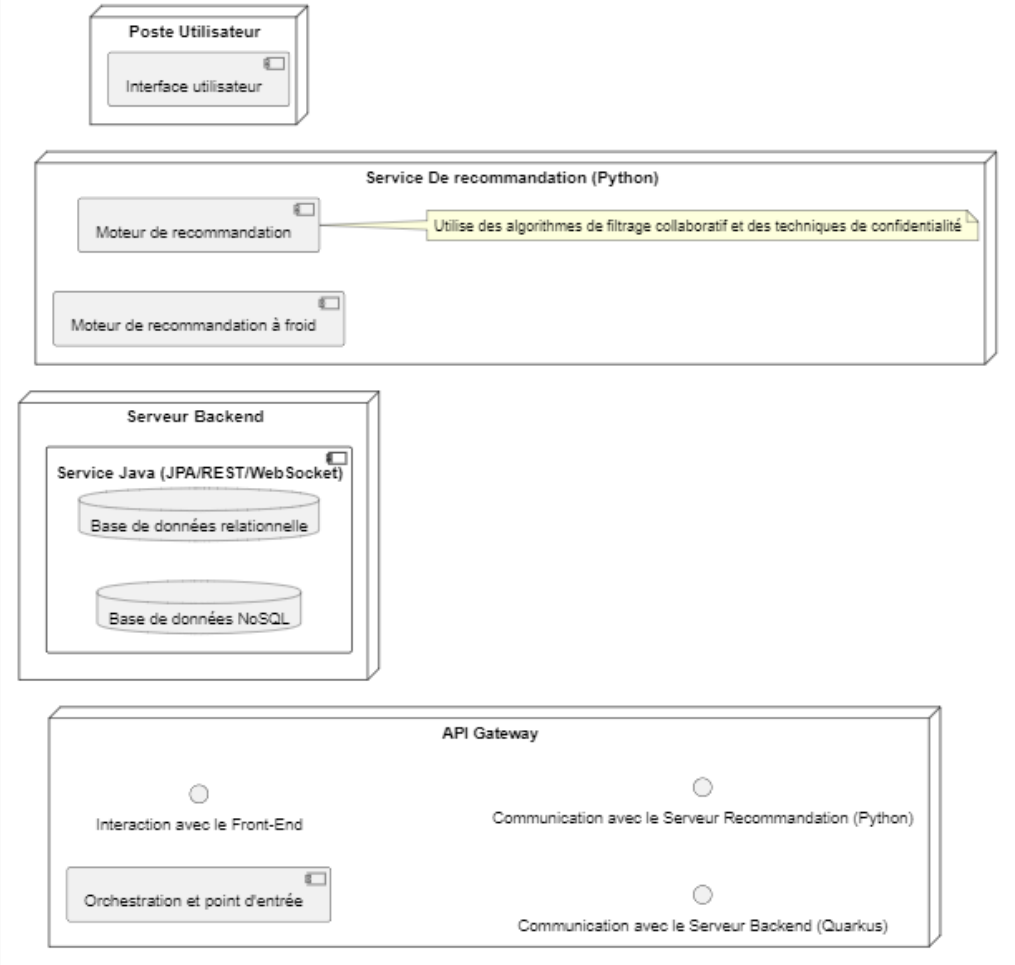


Figure 2: Diagramme de déploiement du système

Choix et démarche

Application Quarkus

Organisation des classes : conforme au modèle UML défini en amont, garantissant la cohérence entre conception et implémentation

Organisation des endpoints Quarkus :

- **Controller :** gère les requêtes HTTP et expose les endpoints REST.
- **Service :** contient la logique métier.
- **Repository :** s'occupe de l'accès aux données.

Service d'importation de CSV : automatisation de l'importation dans la base de données relationnelle et noSql

- Traitement par batchs (paquets) pour éviter les surcharges mémoire.
- Multi-threading en utilisant CompletableFuture pour accélérer l'import parallèle.
- Flush/Clear périodique de l'EntityManager pour libérer les ressources.
- Retry logique sur l'indexation Elasticsearch pour plus de robustesse.

Application Quarkus(2)

Authentification sécurisée : vérification du mot de passe haché avec un salt : Le mot de passe est combiné à un salt unique, puis haché avec SHA-256 pour sécuriser les données stockées.

Intégration d'une base NoSQL avec Elasticsearch : utilisé pour accélérer la recherche grâce à l'indexation des données (films, genres...), uniquement pour la consultation rapide, pas pour le traitement métier.

Quarkus : Serveur principal de l'application est orchestré par un environnement docker compose

- **PostgreSQL :** base de données relationnelle principale.
- **Elasticsearch :** base NoSQL pour les recherches rapides via indexation.
- **API Python :** microservice dédié aux recommandations intelligentes.

Architecture générale

Attributs nécessaire :

- **Dataset**
- **Production (boolean)**

Méthodes nécessaire :

- **Initialisation des données**
- **Entrainement du modèle**
- **Sauvegarde / Chargement du modèle**
- **Prédiction**
- **Précision**

Création des datasets

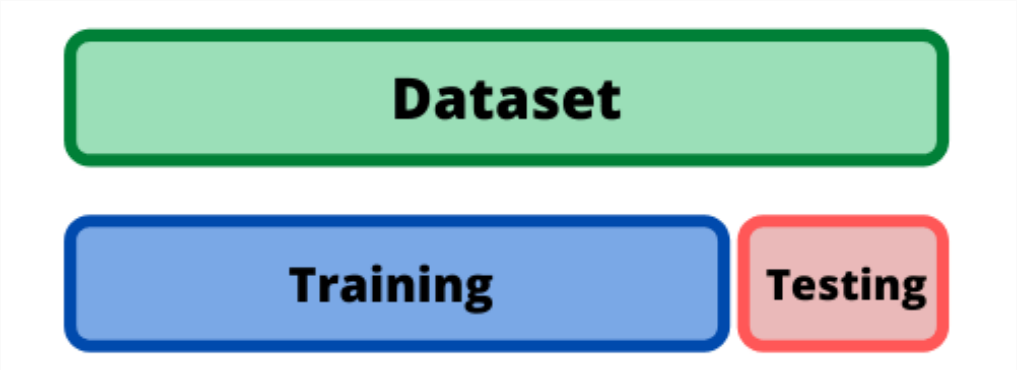
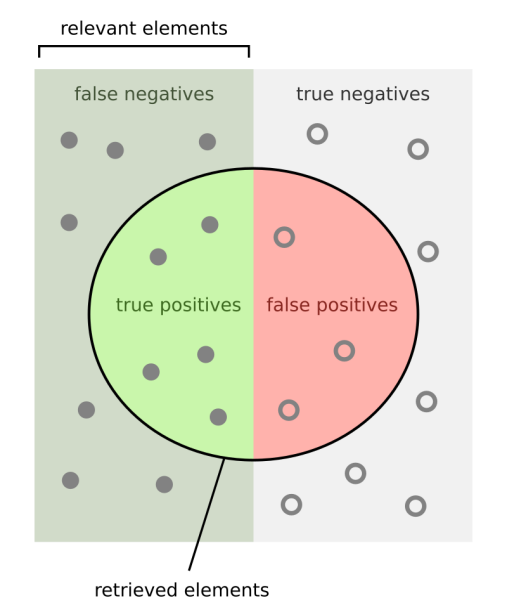


Figure 3: Répartition des données en ensembles d'entraînement et de test

Métriques d'évaluation : précision et rappel



How many retrieved items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{retrieved elements}}$$

How many relevant items are retrieved?

$$\text{Recall} = \frac{\text{true positives}}{\text{relevant elements}}$$

Système de recommandation à chaud

Contexte : Utilisé lorsque l'utilisateur a un historique de notes ou d'interactions avec le système.

Méthodes utilisées :

- **Filtrage collaboratif** : basé sur les préférences d'utilisateurs similaires.
- **Matrix Factorization (SVD)** : extraction de facteurs latents pour modéliser les préférences.

Choix justifiés :

- Fortes performances avec un dataset riche (MovieLens).
- Algorithmes bien adaptés à la personnalisation dynamique.
- Intégration fluide avec la Differential Privacy pour les échanges sécurisés.

Differential Privacy

Mise en place du bruit de Laplace : Lors de l'envoie des données des utilisateurs au serveur de recommendation, nous ajoutons un bruit de Laplace aux notes données par l'utilisateurs.

Objectifs :

- Brouiller les données des utilisateurs pour limiter l'extraction des informations personnelles depuis le systeme de recommendation.
- Conserver les données statistiques globales pour l'entraînement des modèles.

Système de recommandation à froid

Contexte : Utilisé lorsque l'utilisateur n'a pas d'historique de notes ou pour les nouveaux utilisateurs.

Méthodes utilisées :

- **Les méthodes basées sur le contenu** : utilisent les similarités entre les items pour faire des recommendations.
- **Les approches hybrides** : combinent le filtrage collaboratif et le filtrage basé sur le contenu.
- **Les méthodes basées sur le clustering** : regroupent les utilisateurs ou les items selon leurs caractéristiques pour recommander des films. communes

Choix justifiés :

- Complémentarité avec le système à chaud pour une couverture complète.
- Utilisation de DBSCAN pour gérer l'évolution de la base de données.
- Représenter la diversité des films présent dans la base de données.

Differential Privacy (2)

Mise en place du mécanisme exponentiel :
Ce mécanisme choisit des films en fonction de leur score de pertinence, tout en introduisant une part aléatoire contrôlée.

Objectifs :

- Protéger la vie privée des utilisateurs en rendant difficile l'inférence des notes globales attribuées aux films.
- Proposer des recommandations pertinentes selon certains critères, comme l'écart maximal de la note moyenne par rapport à 2.5.

Déroulement du projet

Déroulement du projet

Le projet a été mené selon une méthodologie **agile** structurée en **sprints hebdomadaires**. Chaque lundi, une réunion de planification permettait de définir les tâches à réaliser, accompagnées d'un **chiffrage en points de charge**. Un **rapport journalier textuel** était rédigé tout les jours pour assurer un suivi continu de l'avancement.

Outils :

- **Python** : langage principal pour le développement du moteur de recommandation
- **GitHub** : collaboration, versionnement, documentation
- **Quarkus** : architecture backend
- **React** : front-end pour la visualisation locale

Tous les éléments (code, docs, rapports) sont sur GitHub pour une architecture micro-service et une collaboration efficace.

Résultats

Résultats obtenus : SVD

Notre système a permis d'atteindre plusieurs objectifs clés, tant sur le plan fonctionnel que technique :

- **Prototype opérationnel** : système complet de bout en bout avec SVD.
- **Qualité des recommandations** :
  - Précision : 72.2% / 71.8%
  - Rappel : 69.1% / 68.6%
- **Respect de la vie privée** :
  - Échanges bruités par Differential Privacy
- **Mesure des performances (temps)** :
  - Initialisation des données : 0.14 s
  - Entrainement du modèle : 0.72 s
  - Temps de réponse moyen : 0.03 s

Résultats obtenus : Deep Learning

Notre système a permis d'atteindre plusieurs objectifs clés, tant sur le plan fonctionnel que technique :

- **Prototype opérationnel** : système complet de bout en bout avec un modèle d'apprentissage profond.
- **Qualité des recommandations** :
  - Précision : 72.6 % / 71.3%
  - Rappel : 69.3% / 67.5%
- **Respect de la vie privée** :
  - Échanges bruités par Differential Privacy
- **Mesure des performances (temps)** :
  - Initialisation des données : 0.08 s
  - Entrainement du modèle : 0.68 s
  - Temps de réponse moyen : 0.03 s

Résultats obtenus : Clustering non supervisé

Nos Objectifs étant de favoriser la diversité tout en représentant les élèments de la base de données : (609 test réalisés avec une réduction à 150 dimensions et 20 films demandés et une même seed aléatoire)

- **Qualité des recommandations** : (non bruité / bruité)
  - Diversité des genres : 60.9 % / 60.4 %
  - Sélections films peu noté moyennes : 23.5% / 26.3%
  - Couverture des recommendations : 23% / 24.7%
  - Nombre moyen de clusters : 8.87 / 8.8 (pour 10 clusters)
  - Ratio films par utilisateur : 2.57 / 2.63
- **Mesure des performances (temps)** :
  - Initialisation des données : 0.02 s
  - Entrainement du modèle : 180 s
  - Temps de réponse moyen : 0.006 s

Perspectives

Perspectives d'évolution

- **Explorer d'autres approches de recommandation à froid** : améliorer la rapidité et la précision et explorer l'ajout de statistiques de données contextuelles pour la recommendation par clustering.
- **Repenser l'architecture microservices** : réduire les fortes dépendances entre les modules pour gagner en modularité, maintenabilité et résilience.
- **Automatiser l'initialisation des données** : remplacement des appels manuels aux endpoints par des scripts d'initialisation ou des pipelines de données.
- **Intégrer un modèle de langage (LLM)** : générer des recommandations contextuelles plus riches grâce à l'exploitation de texte ou de préférences complexes.
- **Améliorer l'interface utilisateur** : créer un front-end plus moderne, esthétique et responsive pour renforcer l'engagement utilisateur.

UNIVERSITÉ DE TOULON

Merci pour votre attention !

N'hésitez pas à poser vos questions !