MOUGAMADOUBOUGARY Mohamed 28604611
MARSSO Denn 28608482

PROJET PC3R
RAPPORT

MOUGAMADOUBOUGARY Mohamed 28604611
MARSSO Denn 28608482

Le lien du site est : https://pc3r-1.onrender.com/. Veuillez attendre 5-10 min pour que le serveur se réveille (Render qui fonctionne comme cela) après avoir fait la première requête, par exemple appuyez sur le bouton "Catégories" et attendez l'affichage des catégories. Le lien du GitHub est sur le site en bas à gauche.

1. Introduction

Ce projet s'inscrit dans le cadre du cours PC3R (Programmation Concurrente, Réactive, Répartie et Réticulaire). Le thème choisi est celui d'un site web permettant aux utilisateurs de noter des films et de donner leur avis, en s'appuyant sur l'API publique TMDb (The Movie Database) pour obtenir les données cinématographiques. Le projet se compose de plusieurs modules déployés séparément, notamment une base de données relationnelle PostgreSQL, un script Python de remplissage, un serveur web écrit en Go et la partie frontend du site statique codé en React.

2. Conception de la base de données

La base de données repose sur un schéma relationnel, hébergé sur Render, conçu pour représenter les différentes entités manipulées par l'application : les utilisateurs, les films, les genres et les avis. Elle garantit la cohérence et l'extensibilité des données (voir le fichier ***Projet/backend/Db.sql***).

La table users stocke les identifiants et les noms des utilisateurs de la plateforme. Chaque utilisateur est identifié de manière unique par un champ id, généré automatiquement, et un nom d'utilisateur username qui est soumis à une contrainte d'unicité.

La table movies contient les informations essentielles sur les films récupérés depuis TMDb, comme leur identifiant TMDb (id), leur title, leur release_date et un lien vers leur poster_path. Ce choix de structure permet une compatibilité directe avec les données brutes de TMDb.

Le modèle de données prend aussi en compte les genres cinématographiques, présents dans une table genres qui liste tous les genres possibles (action, comédie, drame, etc.). La relation many-to-many entre movies et genres est représentée à l'aide d'une table de jointure movie_genres, assurant une modélisation fidèle à la réalité des contenus.

Enfin, les avis et notes des utilisateurs sont stockés dans la table ratings. Elle relie un user_id à un movie_id via une note entière (rating, bornée entre 1 et 10) et un commentaire facultatif (comment). Des contraintes d'intégrité référentielle assurent que les notes ne concernent que des utilisateurs et films existants, garantissant la fiabilité des données.

MOUGAMADOUBOUGARY Mohamed 28604611
MARSSO Denn 28608482

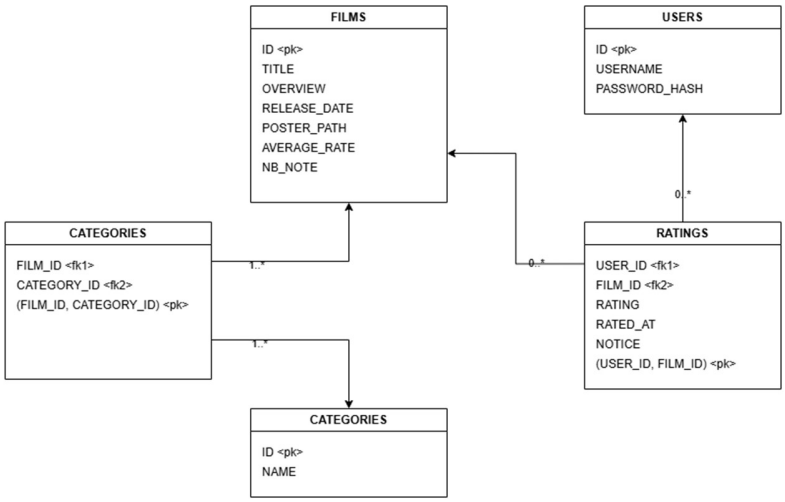


Figure 1 : Diagramme UML de la base de données

3. Script Python de remplissage de la base

Un script Python (voir le fichier ***Projet/backend/fillMovies.py***) a été développé afin de récupérer dynamiquement les données de TMDb et de les insérer ou mettre à jour dans la base de données. Ce script constitue un pont automatisé entre l'API TMDb et la base PostgreSQL. Il est capable de récupérer les films les plus populaires, ou encore les films d'un genre spécifique, ainsi que les attributs associées (titre, date de sortie, image, genres, etc.).

Le script repose sur la bibliothèque requests pour effectuer les appels à l'API, et sur psycopg2 pour l'interaction avec la base de données. Il traite les données récupérées de manière structurée, effectuant des vérifications pour éviter les doublons. L'insertion dans la base est conçue de façon stable grâce à l'utilisation de clauses SQL de type ON CONFLICT, permettant une mise à jour sans écrasement si un film existe déjà. De plus, le script ajoute seulement les films qui sont sortis après le film le plus récent de la base de données.

Les genres sont également synchronisés à chaque exécution. Le script vérifie l'existence de chaque genre retourné par TMDb avant de l'ajouter à la table genres. Il insère ensuite les correspondances film-genre dans la table film_categories. Cela garantit que la base reste cohérente avec les données les plus récentes de TMDb, tout en évitant les redondances.

MOUGAMADOUBOUGARY Mohamed 28604611
MARSSO Denn 28608482

4. Serveur Go – API REST

Le serveur web, développé en Go, est responsable de l'exposition d'une API RESTful que le frontend consomme. Il joue un rôle central dans l'architecture du projet en orchestrant les échanges entre la base de données et l'interface utilisateur. Le choix de Go pour le backend permet de profiter de ses performances élevées, de sa gestion native de la concurrence et de sa simplicité syntaxique.

Le serveur utilise les packages standards net/http pour créer les routes, database/sql pour interagir avec PostgreSQL via le pilote lib/pq, et encoding/json pour encoder/décoder les données échangées au format JSON.

4.1. Endpoints

L'API backend développée en Go expose plusieurs endpoints RESTful permettant l'accès et la manipulation des données liées aux films, catégories et utilisateurs. Tous les endpoints acceptent et renvoient des données au format **JSON**.

4.1.1. Films

- **GET** `/films/getById/:id`
  Permet de récupérer les informations détaillées d'un film à partir de son identifiant.
- **GET** `/films/getByAlphabetLetter`
  Récupère la liste des films dont le titre commence par une lettre donnée.
- **GET** `/films/getByText`
  Effectue une recherche textuelle sur les titres et descriptions de films.
- **POST** `/films/rate`
  Enregistre la note attribuée par un utilisateur à un film.
- **POST** `/films/checkIfRated`
  Vérifie si un utilisateur a déjà noté un film donné.
- **POST** `/films/getRating`
  Récupère la note moyenne et le nombre de votes pour un film.

4.1.2. Catégories

- **GET** `/categories`
  Récupère l'ensemble des catégories de films disponibles dans la base de données.
- **GET** `/categories/getFilms`
  Retourne les films associés à une catégorie spécifique.

4.1.3. Utilisateurs

- **POST** `/users/getAccount`
  Authentifie un utilisateur à partir de ses identifiants (nom d'utilisateur et mot de passe).
- **POST** `/users/checkUsername`
  Vérifie la disponibilité d'un nom d'utilisateur lors de l'inscription.
- **POST** `/users/create`
  Crée un nouveau compte utilisateur.
- **DELETE** `/users/delete`
  Supprime un compte utilisateur existant et supprime aussi les commentaires mais pas les notes.

4.2. Cas d'utilisation

- Un utilisateur non connecté peut consulter la fiche détaillée d'un film en cliquant sur sa carte depuis la page d'accueil, cela déclenche un appel à l'API qui renvoie les informations complètes du film, affichées ensuite dans une page dédiée.
- Un utilisateur curieux peut explorer le catalogue en naviguant par lettre alphabétique, en cliquant sur une lettre, l'application lui présente les films dont le titre commence par celle-ci, grâce à une requête au backend.
- Un utilisateur connecté peut noter un film après l'avoir visionné : il sélectionne une note sur l'interface et écrit un commentaire, qui sont ensuite envoyés via une requête POST à l'API, laquelle enregistre le commentaire et la note et met à jour la moyenne du film.

L'API est conçue pour renvoyer des réponses JSON standardisées, accompagnées des codes HTTP appropriés pour indiquer le succès, les erreurs client ou serveur. L'accent est mis sur la robustesse et la sécurité, avec une gestion des erreurs et l'utilisation de requêtes préparées pour éviter les injections SQL.

MOUGAMADOUBOUGARY Mohamed 28604611
MARSSO Denn 28608482

5. Site statique

La partie frontend du projet a été entièrement développée avec React, une bibliothèque JavaScript populaire pour la création d'interfaces utilisateurs interactives. L'application repose sur une architecture composée de composants réutilisables, ce qui facilite la lisibilité, la maintenance et l'évolution du code. La navigation entre les différentes pages est assurée par React Router (`react-router-dom`), à travers l'utilisation de BrowserRouter, Routes et Route, permettant une expérience utilisateur fluide en mode Single Page Application (SPA). Parmi les routes principales, on retrouve l'accueil, les catégories de films, un classement alphabétique, des pages statiques telles que l'équipe et à propos, ainsi qu'un système de connexion, de déconnexion, et de visualisation détaillée des films. L'application intègre également un contexte d'authentification (`AuthContext`) via le composant AuthProvider, permettant de gérer l'état de connexion de l'utilisateur à travers l'ensemble de l'application.

Par ailleurs, les données sont récupérées dynamiquement depuis le backend développé en Go à l'aide de la méthode `fetch`. Par exemple, dans la page d'accueil ou dans la page de détail d'un film, une requête du type :

```
fetch("http://localhost:8080/films/123")
  .then(response => response.json())
  .then(data => setFilm(data))
  .catch(error => console.error("Erreur lors du chargement du film :", error));
```

permet de récupérer les informations d'un film spécifique en fonction de son identifiant. Les données reçues sont ensuite stockées dans un `state` React avec `useState`, ce qui permet une mise à jour automatique de l'interface graphique.

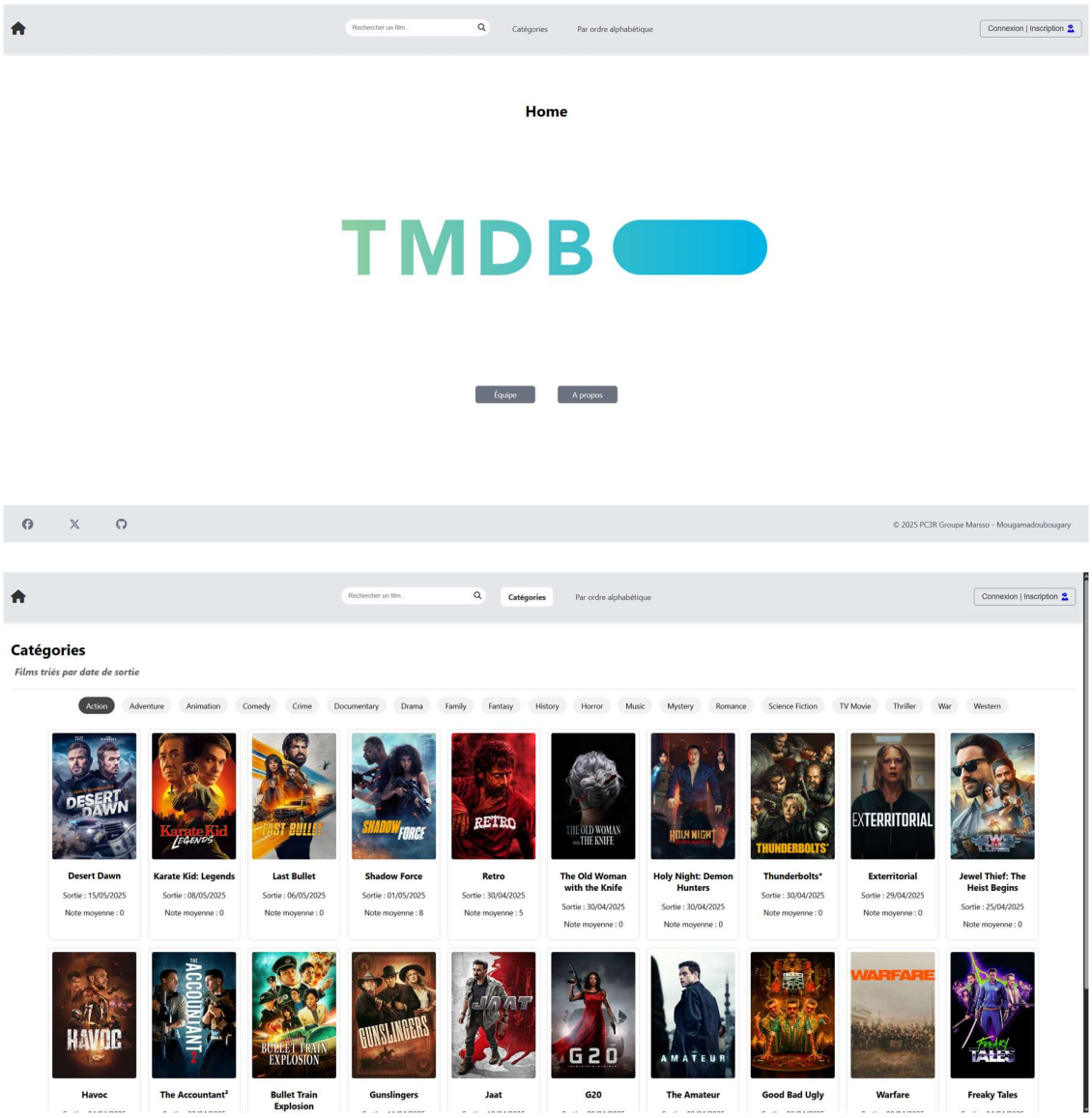
Enfin, la structure visuelle de l'interface repose sur des composants principaux comme `Navbar` et `Footer`, qui encadrent dynamiquement le contenu selon la page affichée.

6. Déploiement et integration

L'ensemble des composants est déployé sur Render, une plateforme cloud gratuite et simple d'utilisation. La base PostgreSQL est hébergée comme un service indépendant, tandis que le serveur Go est déployé en tant que web service. Le script Python peut être exécuté localement ou bien automatisé à l'aide d'un cron externe ou d'un job Render, afin de mettre à jour régulièrement les films disponibles. Un job Render est payant, donc on a utilisé un crontab local qui lance le script python tous les jours à 20h.

MOUGAMADOUBOUGARY Mohamed 28604611
MARSSO Denn 28608482

Cette séparation des responsabilités respecte les principes de modularité et de répartition, fondamentaux dans le cadre du cours PC3R. Chaque composant peut être maintenu ou redéployé indépendamment, facilitant l'évolutivité de l'ensemble du projet.



MOUGAMADOUBOUGARY Mohamed 28604611
MARSSO Denn 28608482

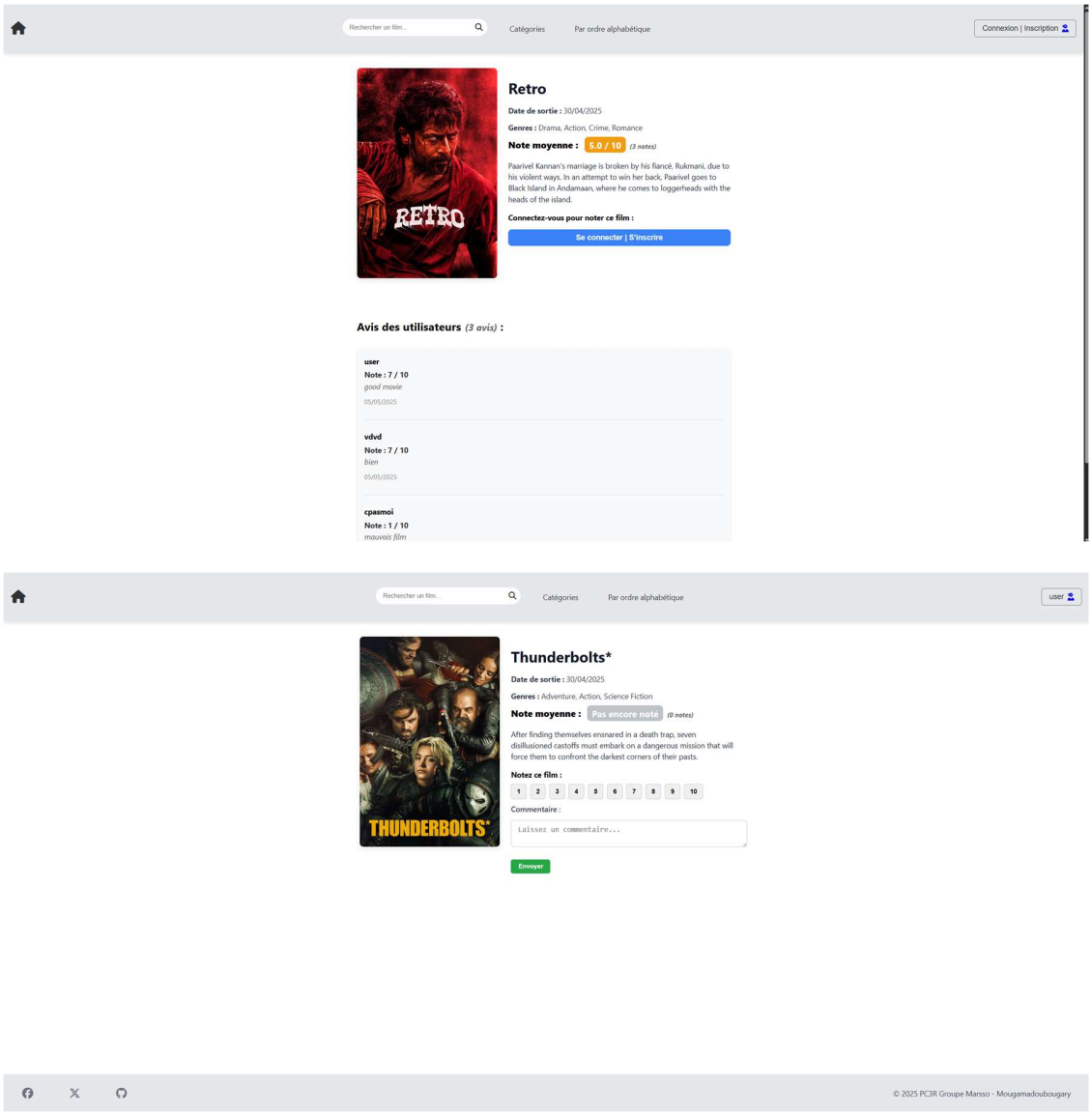


Figure 2 : Captures d'écran du site

7. Conclusion

Le backend développé pour ce projet met en œuvre de manière cohérente les concepts abordés en PC3R : réactivité, grâce à l'utilisation de l'API TMDb ; répartition, avec le découpage en services indépendants ; concurrence, potentiellement exploitée côté Go via des routines ; et réticulation, dans le maillage entre les entités (films, genres, utilisateurs, avis).