Université de Bordeaux

Master 2 Génie Logiciel

Rapport de Projet Web :

Hub de Films et Séries

---

https://github.com/gduboureau/CineInfo

Réalisé par

Ephrem Jennifer
Loustau Valentin
Duboureau Guillaume

Année universitaire 2023-2024

Table des matières

1 Description du Projet

1.1 Objectif

L'objectif de ce projet est de créer une application web dédiée aux films et séries, permettant aux utilisateurs de découvrir les contenus du moment, de les ajouter à leur liste de favoris, de les suivre dans leur watchlist, de les marquer comme visionnés et de les noter. Plus précisemment, nous souhaitons pouvoir afficher des films et séries et pouvoir en afficher les détails : trailers, acteurs, description... De plus, nous souhaitons mettre en place une partie "utilisateur connecté" afin qu'il puisse réaliser des actions supplémentaires telle que la personnalisation de son compte par exemple.

1.2 Fonctionnalités

Voici les différentes fonctionnalités mises en place au sein de notre application web :

- **Recherche par titre.**
- **Système de catégorie :** 'Populaires', 'À Venir', 'En Ce Moment', etc.
- **Filtre par genre :** 'Action', 'Aventure', etc.
- **Présentation complète d'un contenu :** Description, notes des utilisateurs, acteurs, images, trailers, recommandations, avis.
- **Création de compte :** déblocage d'un ensemble de fonctionnalités personnalisées.
  - Ajouter des films/séries à sa liste de favoris.
  - Ajouter des films/séries à sa watchlist.
  - Rédiger des avis sur des films/séries (+ saisons pour les séries).
  - Noter les films/séries (+ épisodes).
  - Cocher des films/séries (uniquement épisodes) comme 'vu' ou 'non vu' depuis sa watchlist.
- **Depuis la page de son compte :**
  - Consulter sa watchlist, ses favoris, les notes données.
  - Paramètres de compte : modification du prénom, nom, pseudo, photo de profil, adresse mail et mot de passe.

1.3 Technologies Utilisées

Afin de réaliser ce projet, nous avons utilisé les technologies suivantes :

- **Backend** : Node.js avec Express (Express v4.18.2, Node v20.8.0)
- **Frontend** : React.js (v18.2.0)
- **Base de données** : PostgreSQL (v16)
- **API tierce** : The Movie Database (TMDB)
- **Outil de containerisation** : Docker (v3)
- **Cache des données** : Redis (v4.6.12)
- **Travail en équipe** : Git/Github

Nous avons également utilisé des librairies/dépendances dans les deux côtés du projet. Par exemple, pour le backend, nous avons utilisé *bcrypt* pour pouvoir crypter les mots de passe de la base de données. Un autre exemple, pour le frontend cette fois-ci, est *slick-carousel* pour pouvoir construire les différents carousels de la page d'acceuil.

2 Description de l'API Tierce (TMDB)

2.1 Fonctionnalités

The Movie Database (TMDB) est une API permettant aux développeurs d'accéder à une vaste base de données qui fournit des informations détaillées sur les films et séries. Les fonctionnalités clés de l'API utilisées dans notre projet incluent :

— **La recherche de contenu** : Quand les utilisateurs effectuent une recherche par titre sur l'application, on fait appel à l'API pour avoir tous les films et séries correspondants. Cette fonctionnalité permet aux utilisateurs d'accéder rapidement à des contenus spécifiques.

```
1 /* Exemple dans le code lorsqu'un utilisateur effectue une recherche */
2 const searchQuery = 'Berlin';
3 const searchMovieResults = await fetch(`https://api.themoviedb.org/3/
    search/movie?api_key=${apiKey}&query=${searchQuery}&include_adult=
    false&language=fr-FR&region=FR`);
4 const searchTVResults = await fetch(`https://api.themoviedb.org/3/
    search/tv?api_key=${apiKey}&query=${searchValue}&include_adult=false
    &language=fr-FR&region=FR`);
```

— **Obtention de détails** : L'API nous fournit des détails exhaustifs sur les films et séries. Cette fonctionnalité nous permet d'afficher une page détaillé d'un film ou d'une série en particulier en incluant la description, la note utilisateur, les bandes-annonces, les acteurs, les recommandations et pour les séries les saisons avec leurs épisodes. Cette fonctionnalité est utilisé quand l'utilisateur veut voir un contenu spécifique.

```
1 /* Exemple d'utilisation dans le code pour obtenir les détails du film
    Oppenheimer avec son id */
2 const movieId = 872585;
3 const movieDetails = await fetch(`https://api.themoviedb.org/3/movie/${
    movieId}?api_key=${apiKey}&language=fr-FR`);
4 /* Exemple pour obtenir les détails de la série Breaking Bad */
5 const serieId = 1396
6 const serieDetails = await fetch(`https://api.themoviedb.org/3/tv/${
    serieId}?api_key=${apiKey}&language=fr-FR`);
```

— **Listes** : L'API propose des listes de films et séries pré-configurées comme les films populaires, les films à venir, les séries en cours de diffusion, etc. Ces listes sont configurées pour fournir aux utilisateurs des recommandations en fonction des tendances actuelles.

```
1 /* Exemple d'utilisation dans le code pour obtenir les films populaires
     */
2 const popularMovies = await fetch(`https://api.themoviedb.org/3/movie/
    popular?api_key=${apiKey}&language=fr-FR&page=1&region=FR`);
3 /* Exemple d'utilisation dans le code pour obtenir les séries les mieux
     notées */
4 const topRatedSeries = await fetch(`https://api.themoviedb.org/3/tv/
    top_rated?api_key=${apiKey}&language=fr-FR&page=1&region=FR`)
```

Pour accéder à l'API, on doit obtenir une clé API qui est gratuite mais avec des limitations de requêtes, elle nous autorise 50 requêtes par seconde.

Pour intéragir avec l'API nous utilisons des requêtes HTTP telles que GET ou POST, qui nous renvoient une réponse au format JSON. Dans la requête on ajoute notre clé API pour être reconnu et on peut également ajouter des paramètres pour spécifier des critères ou filtrer les résultats, par exemple nous avons ajouté "language=FR-fr" dans nos requêtes pour avoir les résultats en français.

2.2 Intégration dans le Projet

L'intégration de l'API tierce dans le projet permet d'assurer une mise à jour constante des données liées aux films et séries.
Nous utilisons les données de l'API pour afficher les films et séries de toutes les listes présentes dans notre en-tête : Films populaires, films du moments, films à venir, films les mieux notés, séries populaires, séries diffusées aujourd'hui, séries en cours de diffusion, séries les mieux notées. Ces données sont également utilisées pour afficher de manière détaillée les informations sur les films et séries, incluant la description, les acteurs, les bandes-annonces, les images, et les évaluations des utilisateurs. Pour les pages de découverte des films et séries, nous offrons un filtre par genres, obtenant ces informations grâce à une requête à l'API.

A chaque fois que nous effectuons une requête du frontend au backend, notre application vérifie si les données requises sont présentes dans le cache Redis avant de faire appel à l'API TMDB. Si les données sont en cache, elles sont directement envoyées au frontend. Sinon, une requête est effectuée vers l'API, les données sont mises en cache et transmises au frontend.

L'intégration de l'API TMDB dans notre projet présente plusieurs avantages :

— **Mise à Jour Régulière** : Les données sur les films et séries sont constamment actualisées, garantissant aux utilisateurs les informations les plus récentes.

— **Richesse des Détails** : Les détails complets fournis par l'API TMDB enrichissent l'expérience utilisateur en offrant des informations détaillées sur chaque contenu.

— **Optimisation des Performances** : La gestion intelligente du cache et la limitation des requêtes à l'API contribuent à optimiser les performances globales de l'application.

L'intégration de l'API TMDB se positionne comme un élément clé dans la réussite de notre application en fournissant un accès fluide et actualisé aux informations sur les films et séries.

3 Description de l'API Interne

3.1 Requête depuis le Frontend

Lorsqu'une page est chargée, une série de requêtes est envoyée au backend pour récupérer des données spécifiques selon les données voulues *(la documentation swagger se trouve à l'adresse http://localhost:8080/api-docs)*
Par exemple pour récupérer la liste des films populaires :

```
fetch("http://localhost:8080/movies/popular", {
    method: "GET",
    headers: {
        "Content-Type": "application/json",
    },
})
```

Une fois les données récupérées, elles sont utilisées pour alimenter différents composants de la page, tels que des carrousels affichant les films correspondants.

3.2 Envoie depuis le Backend

Lorsqu'une demande est reçue, le backend utilise une fonction prédéfinie selon les données voulues pour les récupérer à partir d'une source externe, en l'occurrence ici notre API TMDB.
Avant de faire une requête à l'API externe, le backend vérifie si les données correspondantes sont déjà présentes dans le cache Redis. Si c'est le cas, il les renvoie immédiatement au frontend sans interroger l'API externe. En cas d'absence de données en cache, le backend effectue la requête à l'API externe, sauvegarde les données dans le cache Redis pour une utilisation future, puis les renvoie au frontend.
Voici un exemple de fonction :

```
const fetchMovies = async (req, res, cacheKey, apiUrl) => {
    try {
        await connectToRedis(); // Connexion á Redis
        /* Récupération des données depuis le cache */
        const cachedData = await getFromCache(cacheKey);
        if (cachedData) {
            return res.json(cachedData);
        }
        /* Effectuer une requéte á l'API externe (TMDB) */
        const tmdbResponse = await fetch(apiUrl);
        const tmdbData = await tmdbResponse.json();
        /* Extraire les résultats des films de la réponse TMDB */
        const movies = tmdbData.results;
        /* Sauvegarde des données dans le cache avec une expiration de 86400
            secondes (1 jour) */
        await saveToCache(cacheKey, movies, 86400);
        res.json(movies); /* Puis renvoyer les données */
    } catch (error) {/* gestion d'erreur ... */}
};
```

3.3 Gestion du cache Redis

La gestion du cache est centralisée avec Redis (*cf redis.js*), un système de stockage clé-valeur. Cette approche vise à améliorer les performances en minimisant les requêtes coûteuses à l'API externe. Lorsqu'une requête du front-end est traitée par le backend, le cache Redis est consulté en premier.
L'utilisation de Redis offre une solution efficace pour la gestion du cache, permettant de réduire la charge sur l'API externe tout en garantissant des temps de réponse rapides et une expérience utilisateur fluide. Nous allons voir cela en détail dans la partie suivante.

4 Choix Techniques

4.1 Backend

Pour le backend, nous avons donc utilisé NodeJS avec Express pour pouvoir déployer notre serveur. Pour cela, nous avons organisé notre code afin de rassembler les routes ayant le même préfixe ensemble. Par exemple, nous avons la route générale */movies*, et ensuite nous rassemblons dans le fichier *movieRoutes.js* toutes les routes qui découllent de celle-ci (*/popular...*). Ainsi le code est mieux réparti et plus compréhensible à lire. Maintenant, pour la gestion du contenu (réception de données, envoie de données..), on traite cela dans le dossier controllers. Comme précedemment, les controlleurs des mêmes routes sont rassemblés.

Comme expliqué précédemment, nous utilisons redis dans le backend pour pouvoir stocker nos valeurs. En effet, Redis est conçu pour offrir des perfomances rapides en stockant les données en mémoire. Dans notre cas, nous avons utilisé le cache pour stocker les données de l'API externe TMDB. En effet, les données pouvant être parfois volumineuses, il est très intéressant de les stocker en mémoire. Nous actualisons les données présentes dans le cache tous les 24h, à l'exception des 'genres' qui sont toujours les mêmes (action, aventure, comédie...). Ainsi, en actualisant nos données, nous récupérons les derniers films et séries ajoutés à l'API. Redis permet donc d'une part d'améliorer le temps de récupération des données et aussi d'améliorer l'expérience utilisateur. D'autre part, le nombre de requêtes étant limité, un trop grand nombre d'utilisateurs provoquerait des problèmes sans le cache.

4.2 Frontend

Pour le frontend, nous avons donc utilisé React. L'un des avantages de react est la réutilisation de composants. Ainsi, nous pouvons diviser nos interfaces en composants indépendants, et les réutiliser à plusieurs endroits dans le code. Par exemple, sur la page d'un film ou d'une série, nous affichons de nombreux détails tels que le nom et la description par exemple dans la première partie de la page. Dans la deuxième partie, nous affichons les acteurs, les vidéos, les images etc., qui sont des composants communs aux films et séries. Cela permet ainsi d'éviter de dupliquer du code et d'appeler ces composants sur la page d'un film ou d'une série. Un autre exemple est dans le header du site. Nous affichons les films du moment, populaires, à venir et les mieux notés. Nous faisons la même chose pour les séries. Ainsi nous utilisons le même composant pour pouvoir afficher nos informations.

Ensuite, React offre de nombreuses librairies utilisables dans notre code. Par exemple, nous utilisons la bibliothèque FontAwesome pour pouvoir afficher des icônes tels que des flèches, des croix et le composant react-slick-carousel pour afficher des carousels d'images sur notre page d'accueil.

4.3 Base de Données

Nous avons créés des tables afin de stocker nos différents données : les données des comptes utilisateurs et les données liées aux films et séries (favoris, commentaires, avis...).

Pour les données liées au compte d'un utilisateur, on stocke dans une table user, les données de l'utilisateur telles que son nom ou prénom ainsi qu'un pseudo et une adresse mail pour servir d'identifiant lors de la connection. Le mot de passe lui est crypté avant d'être stocké dans la base de données (il sera décrypté dans le backend et sera comparé au mot de passe envoyé par l'utilisateur pour vérifier qu'il est bon).

Nous avons également la possibilité d'ajouter des films dans nos favoris, de les noter... Pour cela, nous avons créés des tables afin de stocker toutes ces données. Nous avons donc des tables pour les films : FavoritesMovies, MovieRatings, MovieComments et WatchlistMovies. Cela correspond aux différentes actions que nous pouvons faire sur les films dans notre site. Pour cela, on recupère l'id du film en question que l'on stockera dans la

base de données, avec l'id de l'utilisateur qui aura réalisé l'action ainsi que la valeur/entrée associée. Concernant les séries, le fonctionnement est similaire à la différence que pour ajouter une série dans notre watchlist, nous pouvons ajouter les épisodes et saisons à regarder. Pour cela nous avons ajouté ces champs dans notre table (c'est la même chose pour noter les séries et laisser des commentaires).

4.4 Autres Technologies

Pour pouvoir gérer l'authentification dans notre projet, nous avons utilisé JWT (JSON Web Token). Voici le fonctionnement : lorsque l'utilisateur se connecte, il envoie son login et mot de passe au backend. Celui-ci vérifie les informations et si elles sont valides, va générer un token et le renvoyer au frontend. Ce dernier va alors stocker ce token dans stockage local (local storage) du navigateur. Ainsi, lorsque que l'on voudra aller sur un contenu nécessitant d'être connecté, le frontend va envoyer la requête voulue contenant le token en en-tête. Le backend peut alors vérifier la signature du token pour vérifier qu'il soit valide (sinon, il renverra un code HTTP forbidden indiquant au frontend que l'utilisateur ne pourra pas accéder à cette ressource). Nous avons également rajouter un système permettant de garder la connection de l'utilisateur au delà d'une heure s'il le souhaite. En effet, lorsque le backend génére le token il défini une durée de vie de 1h. Dans le frontend, on déclenche alors un timer, qui au bout de 55 minutes, demandera à l'utilisateur s'il souhaite rester connecter. Si oui, alors on générera un nouveau token qui remplacera le précédemment, sinon l'utilisateur sera déconnecté.

Pour pouvoir gérer la perte de mot de passe d'un utilisateur, nous avons utilisé la dépendance "nodemailer" dans le backend. Pour cela on récupère l'adresse email rentrée par l'utilisateur, on vérifie qu'elle soit bien associée à un compte et si c'est le cas, on envoie un mail à cette adresse contenant un nouveau mot de passe provisoire. L'utilisateur pourra ensuite changer ce mot de passe dans son espace personnel.

Nous avons également chiffrer les mots de passe des utilisateurs afin qu'ils n'apparaissent pas en clair dans la base de donnée. Pour cela, nous avons utilisé le module "bcrypt" dans le backend. Lorsque l'utilisateur se connecte, nous vérifions que le mot de passe saisie correspond au mot de passe stocké dans la base de données, que nous déchiffrons au préalable.

5 Conclusion

5.1 Problèmes Rencontrés

Au cours du développement de notre application sur les films et séries un problème a été rencontré. Cela concernait la mise en place initiale du projet à l'aide de Docker. Comprendre le fonctionnement de Docker et configurer le projet de manière optimale a constitué une étape initiale exigeante, nécessitant une période d'apprentissage et de résolution de problèmes.

5.2 Limitations Connues

Actuellement, certaines limitations sont observées au sein de l'application qui sont essentielles à identifier pour orienter nos efforts d'amélioration :

— **Absence de Section sur les Acteurs** : Actuellement, l'application ne propose pas de section dédiée aux acteurs, limitant ainsi l'accès aux détails spécifiques sur leurs rôles dans les films et séries.

— **Traitement des Images Volumineuses** : Le traitement des images volumineuses, en particulier pour les utilisateurs connectés, peut entraîner des temps de chargement prolongés sur des connexions Internet plus lentes, impactant ainsi l'expérience utilisateur.

— **Gestion des Erreurs** : La gestion des erreurs pourrait être améliorée pour fournir des messages d'erreur plus explicites et des solutions de contournement en cas de défaillance temporaire de l'API tierce.

— **Limitation de la Gestion des Profils Utilisateurs** : Actuellement, la fonctionnalité de gestion des profils utilisateurs ne permet qu'un choix limité, se limitant à la sélection d'une photo de profil sans offrir la possibilité de personnaliser davantage avec une couleur ou une photo de fond.

5.3 Évolutions Possibles

Pour assurer la croissance continue de notre application, plusieurs pistes d'évolution prometteuses ont été identifiées :

— **Amélioration de l'Expérience Utilisateur** : Enrichir l'expérience utilisateur en introduisant davantage de filtres dans les catégories, offrant ainsi aux utilisateurs des options plus précises pour affiner leurs recherches de films et séries.

— **Système de Recommandations Avancé** : Implémenter un système de recommandations plus sophistiqué, prenant en compte les habitudes de visionnage et les genres préférés des utilisateurs pour offrir des suggestions encore plus personnalisées.

— **Support Multi-langues** : Élargir la portée de l'application en introduisant un support multi-langues, avec l'ajout d'au moins l'anglais en plus, pour atteindre un public plus diversifié.

— **Système de Discussion Collaboratif** : Intégrer un système de discussion en complément des avis, offrant aux utilisateurs un espace pour débattre et échanger sur leurs préférences cinématographiques.

— **Chat Intelligent** : Mettre en place un chat disponible en permanence pour les utilisateurs permettant de faire des recherches particulière sur du contenu à la manière d'un moteur de recherche ; exemple : "Je cherche le film Marvel le plus regardé", "Les films de The Rock humoristique des 5 dernières années", etc.

Ces évolutions potentielles visent à répondre aux besoins changeants des utilisateurs et à enrichir significativement leur expérience au sein de l'application. Ces ajustements, orientés vers la personnalisation et une meilleure accessibilité, contribueront à améliorer le statut de notre application au sein du monde cinématographique en ligne.