My Watching Companion

« Création d’une plateforme de gestion de visionnage de films et séries en ligne. »

Par Vasco, Guillaume et Baptiste – Au cours du BTS – Projet de Formation

Table des matières

1. Introduction 3
2. Objectifs du Projet et Cahier des Charges 3
   2.1 Contexte de Développement 3
   2.2 Objectifs Principaux et Fonctionnalités Clés (Cahier des Charges) 3
3. Architecture de l'Application 4
   3.1. Style Architectural MVC 4
   3.2. Modules et Composants Clés 4
   3.3. Fichiers Essentiels et Leurs Rôles 4
   3.4. Choix Technologiques 5
4. Gestion et Structure des Données 6
   4.1. Structures de Données Clés 6
   4.2. Mécanisme de Stockage et Sécurité 7
   4.3. Flux de Données Principal 7
5. Fonctionnalités Principales et Mon Rôle dans leur Implémentation 8
   5.1. Mon Travail sur le Système d'Authentification et de Sécurité 8
   5.2. Ma Contribution au Système de Watchlists 9
   5.3. Développement des Pages d'Œuvres (Artworks) 11
   5.4. Interface Utilisateur Responsive 13
6. Interface Utilisateur (UI) 13
   6.1. Description des Écrans/Vues Principales 13
   6.2. Définition de l'UI et Composants Clés 16
   6.3. Flux d'Interaction Utilisateur Principaux 16
7. Points Particuliers et Choix de Conception 17
8. Stratégies de Gestion des Erreurs et de Journalisation 17
   8.1 Gestion des Erreurs 17
   8.2 Journalisation (Logging) 18
9. Mon Rôle Spécifique dans le Projet 18
10. Perspectives d'Évolution et Améliorations Futures 19
11. Bilan Personnel et Apprentissages 19
12. Conclusion 20

1. Introduction

J'ai participé au développement de My Watching Companion, une application web que nous avons conçue pour permettre aux utilisateurs de créer et de gérer facilement leurs listes de visionnage personnalisées pour les films et séries. L'application s'appuie sur l'API TMDB pour fournir des informations détaillées sur les œuvres et comprend un système complet d'authentification et de gestion des comptes utilisateurs. L'idée est de proposer un outil simple et intuitif pour organiser son suivi de contenus audiovisuels.

2. Objectifs du Projet et Cahier des Charges

2.1 Contexte de Développement

Nous avons développé My Watching Companion dans le cadre de notre formation en BTS SIO, option SLAM. L'objectif était de mettre en pratique nos compétences sur les technologies web modernes en créant une plateforme personnelle et utile pour la gestion de contenu multimédia. Ce projet fait partie de nos 2 projets à réaliser pour valider notre année.

2.2 Objectifs Principaux et Fonctionnalités Clés (Cahier des Charges)

Notre but était de proposer un outil permettant à chaque utilisateur de gérer ses séries et films de manière personnalisée. Les objectifs majeurs étaient :

- **Créer un système de gestion de watchlists personnalisées** : Chaque utilisateur peut créer plusieurs listes (ex : "À regarder entre amis", "Films primés") ;
- **Ajout, modification et suppression de titres** : Permettre aux utilisateurs d'enrichir leurs listes avec des informations essentielles (titre, type, statut) et de les gérer ;
- **Suivi de l'avancement des contenus** : Chaque œuvre peut être marquée comme "Pas vu", "En cours", ou "Fini" ;
- **Implémenter un système d'authentification robuste** : Ceci inclut l'inscription, la connexion, la déconnexion, la gestion des sessions et un système de récupération de mot de passe via des questions de sécurité. C'est une partie sur laquelle j'ai particulièrement travaillé ;
- **Intégrer des APIs externes (TMDB)** : Pour récupérer automatiquement les informations détaillées des films et séries (posters, résumés, notes, etc.). J'ai également contribué à la création des pages dédiées à ces œuvres ;
- **Développer une interface utilisateur moderne et responsive** : Accessible via https://mywatchingcompanion.com, l'interface est dynamique et conviviale ;
- **Fonction de partage de watchlists** : Permettre aux utilisateurs de rendre leurs listes visibles à leurs amis pour favoriser l'échange et les recommandations.

3. Architecture de l'Application

3.1. Style Architectural MVC

Pour My Watching Companion, nous avons adopté une architecture **MVC (Model-View-Controller)**. Ce choix a été fait pour organiser le code de manière structurée, claire et maintenable, ce qui est essentiel pour un projet collaboratif et évolutif.

- **Models (Modèle)** : Cette couche gère les données et les interactions avec notre base de données SQL Server. J'ai travaillé sur les requêtes SQL pour les fonctionnalités d'authentification (j'ai développé les requêtes déjà existantes, créées par mon collègue pour qu'elles soient plus adaptées à nos besoins), de gestion des watchlists et des artworks. Elle comprend la structure des tables et les relations entre elles (utilisateurs, watchlists, œuvres, statuts) ;
- **Views (Vue)** : J'ai participé à la création des vues en utilisant le moteur de template **EJS (Embedded JavaScript)**. Cela nous permet de générer dynamiquement les pages HTML côté serveur, en injectant les données spécifiques à l'utilisateur ou au contenu demandé ;
- **Controllers (Contrôleur)** : Développés avec **Express.js**, les contrôleurs font le lien entre le modèle et la vue. J'ai été responsable de la création de nombreux contrôleurs, à la fois pour la partie app (rendu des vues) et backend (logique API), notamment pour l'authentification, les watchlists et les artworks. Ils interprètent les requêtes des utilisateurs, interagissent avec le modèle pour manipuler les données, et passent les résultats aux vues EJS.

3.2. Modules et Composants Clés

La structure de notre projet est organisée comme suit :

3.3. Fichiers Essentiels et Leurs Rôles

- **Points d'entrée principaux** :
    - index.js : Notre serveur Express principal, qui configure les middlewares et les routes ;

    - config.js : Fichier de configuration centralisé pour les variables d'environnement que nous utilisons ;
    - db.js : Le module que j'ai en partie développé pour la connexion et l'exécution des requêtes à notre base de données SQL Server.
- **Fichiers de configuration** :
    - package.json : Gère les dépendances de notre projet et les scripts npm ;
    - .env : Contient nos variables d'environnement sensibles (non versionné sur Git).
- **Interface utilisateur** :
    - views/ : Nos templates EJS pour chaque page (connexion, inscription, watchlists, détails d'œuvre, etc.). J'ai personnellement créé les vues pour signin, signup, forgot-password, my-watchlists, artwork, et d'autres ;
    - public/styles/ : Les fichiers CSS, avec un style spécifique par page pour une meilleure modularité ;
    - public/scripts/ : Le JavaScript côté client que j'ai écrit pour ajouter de l'interactivité aux pages.
- **Gestion des données** :
    - controllers/backend/ : Les API REST que j'ai développées pour la gestion des utilisateurs, watchlists, et artworks ;
    - db.js : La couche d'abstraction pour interagir avec SQL Server.

3.4. Choix Technologiques

Nous avons basé notre projet sur un ensemble de technologies web modernes et éprouvées :

- **Langage(s) de Programmation Principal(aux)** : JavaScript (Node.js pour le backend, et JavaScript vanilla pour le frontend), HTML5, CSS3, et SQL pour les requêtes base de données.
- **Frameworks et Bibliothèques Majeures** :
    - **Express.js** : Le framework web Node.js que nous avons utilisé pour construire notre backend et gérer les routes ;
    - **EJS** : Notre moteur de template pour générer dynamiquement les vues HTML côté serveur ;
    - **Express-session** : Pour la gestion des sessions utilisateur ;
    - **Crypto-js** : Utilisé pour le chiffrement AES des mots de passe ;
    - **Multer** : Pour la gestion de l'upload des images de profil ;
    - **mssql** : Le driver Node.js pour interagir avec notre base de données Microsoft SQL Server ;
    - **dotenv** : Pour la gestion sécurisée des variables d'environnement.

- **Base(s) de Données** : **Microsoft SQL Server**, un SGBD relationnel que nous avons choisi pour sa robustesse. Il gère les utilisateurs, watchlists, titres, et statuts d'avancement ;
- **API Utilisées** : **TMDB (The Movie Database) API** pour récupérer les informations détaillées (posters, résumés, notes, etc.) sur les films et séries ;
- **Environnement de Développement et Outils Notables** : Node.js, npm, et le flag --watch (via nodemon implicitement) pour le rechargement automatique en développement ;
- **Système de Gestion de Version** : Git et Github.

4. Gestion et Structure des Données

4.1. Structures de Données Clés

J'ai travaillé avec les structures de données suivantes, notamment pour la session utilisateur et la composition des watchlists :

- **Structure utilisateur en session (req.session.user)** :

```
user: {
  id: number,
  username: string,
  email: string,
  firstName: string,
  lastName: string,
  profilePicture: string,
  confidentiality: number, // Niveau de confidentialité du profil
  bio: string,
  gender: string,
  roleID: number // Rôle de l'utilisateur (admin, user)
```

- **Structure d'une watchlist (telle que retournée par l'API et utilisée dans les vues)** :

```
watchlist: {
  list_id: number,
  list_name: string,
  username: string,          // Propriétaire de la liste
  cover_url: string,         // URL de l'image de couverture (premier artwork)
  artworks: [{               // Tableau des œuvres dans la liste
    artwork_id: number,
    artwork_name: string,
    artwork_api_link: string, // Lien vers l'API TMDB
    artwork_poster: string,   // URL du poster
    artwork_liked: boolean,   // Si l'utilisateur a aimé l'œuvre
    artwork_watch_status: string // Ex: "Pas vu", "En cours", "Fini"
  }]
}
```

4.2. Mécanisme de Stockage et Sécurité

- **Sessions Utilisateur** : Elles sont gérées en mémoire côté serveur grâce à express-session. J'ai configuré des durées de session variables selon que l'utilisateur choisit "Se souvenir de moi" ou non, les sessions sont automatiquement prolongées lorsque l'utilisateur interagit avec la page web ;
- **Mots de passe** : Pour la sécurité, les mots de passe des utilisateurs sont chiffrés en base de données en utilisant l'algorithme AES fourni par crypto-js. La clé de chiffrement est stockée dans les variables d'environnement ;
- **Base de données SQL Server** : Toutes les données persistantes (utilisateurs, listes, œuvres, relations) sont stockées ici. J'ai veillé à utiliser des requêtes paramétrées via le module mssql pour prévenir les injections SQL ;
- **Images de Profil** : Les images de profil uploadées par les utilisateurs sont stockées localement sur le serveur dans le dossier public/UsersProfilePicture/.

4.3. Flux de Données Principal

J'ai contribué à définir et implémenter plusieurs flux de données essentiels :

1. **Authentification** : L'utilisateur soumet ses identifiants. Le backend vérifie ces informations en base de données (après déchiffrement du mot de passe stocké). Si la vérification est réussie, une session est créée pour l'utilisateur, et il est redirigé vers la page demandée ou son tableau de bord ;
2. **Affichage des Watchlists** : Lorsqu'un utilisateur accède à ses watchlists, sa session est validée. Une requête est envoyée au backend (API que j'ai développée) qui interroge la base de données pour récupérer ses listes et les œuvres associées. Ces données sont retournées au format JSON, puis le frontend (via EJS ou JavaScript client) les utilise pour afficher les listes et leurs couvertures ;
3. **Affichage d'une Œuvre (Artwork)** : Quand l'utilisateur consulte une page d'œuvre, l'application récupère d'abord les informations de base de notre propre base de données. Ensuite, elle interroge l'API TMDB en utilisant le lien API stocké pour

obtenir des données enrichies (détails, casting, images). Ces informations combinées sont ensuite utilisées pour rendre la page de l'œuvre, y compris un thème de couleurs dynamique basé sur le poster.

5. Fonctionnalités Principales et Mon Rôle dans leur Implémentation

J'ai joué un rôle clé dans le développement de plusieurs fonctionnalités majeures de My Watching Companion, tant côté backend que frontend.

5.1. Mon Travail sur le Système d'Authentification et de Sécurité

C'est l'une des premières parties sur lesquelles j'ai travaillé intensivement avec mon collègue, avant de prendre en charge une grande partie de son développement et de son perfectionnement.

- **Description** : J'ai mis en place un système complet permettant aux utilisateurs de s'inscrire, de se connecter, de se déconnecter, et de gérer leur session. Une fonctionnalité importante que j'ai développée est la récupération de mot de passe via des questions de sécurité ;
- **Fichiers Impliqués** :
    - controllers/backend/users.js : Logique pour l'inscription, la connexion ;
    - controllers/backend/security.js : Logique pour les questions de sécurité et la réinitialisation du mot de passe ;
    - controllers/app/sign-in-up.js : Rendu des vues EJS pour les pages d'authentification ;
    - public/scripts/sign-up.js, sign-in.js, forgot-password.js : Scripts frontend pour l'interactivité des formulaires ;
    - Vues EJS : views/sign-in.ejs, views/sign-up.ejs, views/forgot-password.ejs. J'ai créé toutes ces vues en respectant notre charte graphique.
- **Logique Clé et Mon Implémentation** :
    - **Chiffrement des Mots de Passe :** Mon collègue a utilisé CryptoJS.AES.encrypt() et CryptoJS.AES.decrypt() pour sécuriser les mots de passe en base de données, avec une clé de chiffrement stockée dans les variables d'environnement (CRYPTO_KEY) ;
    - **Gestion des Sessions :** J'ai configuré express-session avec des durées de session variables (1 heure par défaut, 1 semaine si "Se souvenir de moi" est coché), et j'ai mis en place la logique pour maintenir l'utilisateur connecté et gérer les redirections après connexion/déconnexion. La fonction ChangeSession dans controllers/functions.js a été centrale pour cela ;
    - **Récupération de Mot de Passe :** J'ai conçu le processus en plusieurs étapes : saisie de l'email, affichage de la question de sécurité associée, vérification de la réponse, et enfin, possibilité de définir un nouveau mot de passe ;

    - **Validations :** J'ai implémenté des validations côté client et serveur pour les formulaires (format email, complexité mot de passe, champs obligatoires).

- **Extrait de Code (Connexion - controllers/backend/users.js)** :

```
// Mon code pour la connexion de l'utilisateur
exports.loginOK = async (req, res) => {
  const { username, password, remember } = req.body;
  try {
    const query = await executeQuery( // Fonction de mon module db.js
      `SELECT UserID, Username, EmailAddress, FirstName, LastName, ProfilePictureURL,
Password, ConfidentialityID, Bio, GenderID, RoleID from Users where Username =
@username`,
      [{ name: "username", type: sql.NVarChar, value: username }]
    );
    const user = query[0];

    if (!user) return res.json({
      status: "ERROR", message: "L'utilisateur n'existe pas."
    });

    // J'utilise ici CryptoJS pour déchiffrer et comparer
    if (password === CryptoJS.AES.decrypt(user.Password,
process.env.CRYPTO_KEY).toString(CryptoJS.enc.Utf8)) {
      ChangeSession(req, user.UserID, user.Username, user.EmailAddress, user.FirstName,
user.LastName, user.ProfilePictureURL, user.ConfidentialityID, user.Bio, user.GenderID,
user.RoleID);

      const sessionDuration = remember ? REMEMBER_SESSION_LIFETIME : SESSION_LIFETIME;
      req.session.cookie.expires = new Date(Date.now() + sessionDuration);

      const redirectTo = req.session.returnTo || "/";
      delete req.session.returnTo;
      // La redirection est gérée côté client après la réponse JSON
      return res.json({ status: "OK", message: "Connexion réussie.", redirectTo:
redirectTo });
    } else {
      return res.json({ status: "ERROR", message: "Mot de passe incorrect." });
    }
  } catch (e) {
    TraceError(e, `Login user ${username}`);
    return res.json({ status: "KO", message: `Erreur interne du serveur.` });
  }
};
```

5.2. Ma Contribution au Système de Watchlists

J'ai pris en charge une grande partie du développement de la logique métier et des interfaces pour la gestion des watchlists.

- **Description** : Cette fonctionnalité permet aux utilisateurs de créer, nommer, et peupler des listes de visionnage personnalisées. J'ai fait en sorte que la première œuvre ajoutée à une liste puisse dynamiquement servir de "cover" pour cette liste dans la page de présentation. Le partage de listes est également une fonctionnalité clé qui a été développée par mon collègue ;

- **Fichiers Impliqués** :

    - controllers/backend/watchlists.js : Mon API pour créer, récupérer, mettre à jour, supprimer des watchlists et leur contenu ;
    - controllers/app/watchlists.js : Le contrôleur pour le rendu de la page my-watchlists.ejs ;
    - public/scripts/my-watchlists.js : Le script client que j'ai écrit pour gérer l'affichage dynamique, les interactions (ajout/suppression d'œuvres), et l'effet visuel des "shadows" colorées autour des posters ;
    - views/my-watchlists.ejs et views/watchlist.ejs : Les templates EJS que j'ai conçus pour afficher les listes et leur contenu.
- **Logique Clé et Mon Implémentation** :
    - **Création/Gestion CRUD :** J'ai développé les routes API (POST /api/watchlist, GET /api/user/:userId/watchlists, DELETE /api/watchlist/:listId, etc.) pour toutes les opérations sur les watchlists et les œuvres qu'elles contiennent ;
    - **Affichage Dynamique des Covers :** J'ai implémenté la requête SQL qui sélectionne la première œuvre d'une liste pour servir de cover_url ;
    - **Couleurs Dominantes et Effet "Shadow" :** Côté client, dans public/scripts/my-watchlists.js, j'ai utilisé la librairie ColorThief pour extraire la couleur dominante du poster de la watchlist et générer dynamiquement un box-shadow de cette couleur autour de l'image, avec un effet de "hover ;
    - **Sécurité :** J'ai veillé à ce que toutes les routes de gestion des watchlists soient sécurisées, en vérifiant que l'utilisateur authentifié est bien le propriétaire de la liste avant toute modification ou suppression.
- **Extrait de Code (Récupération des watchlists de l'utilisateur - controllers/backend/watchlists.js)** :

```
// Mon API pour récupérer les watchlists d'un utilisateur avec leur cover
exports.getUserWatchlists = async (req, res) => {
  try {
    const user = req.session.user;
    if (!user) return res.status(401).json({
      error: "Vous devez être connecté pour obtenir ces informations."
    });

    // Requête SQL optimisée pour récupérer la première image comme cover
    const queryResult = await executeQuery(
      `SELECT
            L.ListID AS list_id,
            L.ListName AS list_name,
            U.Username AS username,
            (SELECT TOP 1 A.ArtworkPosterImage
             FROM Ref_ListArtwork RA
             JOIN Artwork A ON A.ArtworkID = RA.ArtworkID
             WHERE RA.ListID = L.ListID
             ORDER BY RA.RefListArtworkID) AS cover_url
        FROM Ref_UsersList RUL
        INNER JOIN Users U ON U.UserID = RUL.UserID
        INNER JOIN List L ON RUL.ListID = L.ListID
        WHERE U.UserID = @userID
        ORDER BY L.ListID`, // Assurer un ordre cohérent
      [{ name: "userID", type: sql.Int, value: user.id }]
    );

    res.status(200).json(queryResult);
  } catch (error) {
    TraceError(error, `Getting watchlists for user ${user ? user.id : 'undefined'}`);
    res.status(500).json({
      error: "Une erreur est survenue lors de la récupération des listes."
    });
  }
};
```

5.3. Développement des Pages d'Œuvres (Artworks)

J'ai également été responsable de la création des pages dédiées à l'affichage des informations détaillées sur les films et séries ("artworks").

- **Description** : Ces pages offrent une vue complète d'une œuvre, en combinant les données stockées localement avec celles récupérées en temps réel de l'API TMDB. Une caractéristique que j'ai implémentée est l'adaptation dynamique du thème de la page aux couleurs dominantes du poster de l'œuvre ;
- **Fichiers Impliqués** :
    - controllers/app/artworks.js : Contrôleur pour le rendu de la vue artwork.ejs ;
    - controllers/backend/artworks.js : API pour récupérer les données d'une œuvre (pouvant impliquer un appel à TMDB si l'œuvre n'est pas entièrement en cache dans notre BD) ;
    - public/scripts/artwork.js : Mon script client pour l'effet de thème dynamique et autres interactions ;

    - views/artwork.ejs : Le template EJS pour afficher les détails de l'œuvre.

- **Logique Clé et Mon Implémentation** :
    - **Intégration TMDB :** Lorsqu'une page d'œuvre est demandée, mon contrôleur (controllers/app/artworks.js) récupère les infos de base de notre SQL Server. Ensuite, il utilise le ArtworkAPILink (qui est l'URL de l'API TMDB pour cette œuvre) pour fetcher les données complètes et à jour de TMDB (casting, genres, résumé, notes, etc.). Ces données sont ensuite passées à la vue ;
    - **Thème Dynamique :** Dans public/scripts/artwork.js, j'ai utilisé ColorThief pour analyser le poster de l'œuvre une fois chargé. J'extrais la couleur dominante et la palette de couleurs, puis je les applique dynamiquement comme variables CSS (--primary-color, etc.) pour adapter l'arrière-plan, les titres, et d'autres éléments de la page aux couleurs de l'affiche, créant une expérience plus immersive ;
    - **Gestion du Statut de Visionnage et "Like" :** J'ai aussi implémenté la logique (backend et frontend) pour que l'utilisateur puisse marquer s'il a vu/est en train de voir/veut voir l'œuvre, et s'il l'a aimée. Ces informations sont spécifiques à l'utilisateur et à sa relation avec l'œuvre.
- **Extrait de Code (Thème dynamique - public/scripts/artwork.js)** :

```
// Mon script pour le thème dynamique de la page artwork
document.addEventListener("DOMContentLoaded", function() {
  const poster = document.querySelector('.artwork-poster img');
  if (poster) { // S'assurer que l'élément existe
    poster.addEventListener('load', function() {
      try { // Ajout d'un try-catch pour ColorThief
        const colorThief = new ColorThief();
        const dominantColor = colorThief.getColor(poster);
        // const palette = colorThief.getPalette(poster, 5); // Palette si besoin de
plus de couleurs

        // J'applique la couleur dominante comme variable CSS pour le thème
        document.documentElement.style.setProperty(
          '--primary-color-artwork-theme', // Variable CSS spécifique
          `rgb(${dominantColor[0]}, ${dominantColor[1]}, ${dominantColor[2]})`
        );
        // On pourrait aussi définir --secondary-color, etc. à partir de la palette
      } catch (e) {
        console.error("Erreur avec ColorThief: ", e);
      }
    });
    // Si l'image est déjà chargée (cache), déclencher manuellement
    if (poster.complete) {
      poster.dispatchEvent(new Event('load'));
    }
  }
});
```

5.4. Interface Utilisateur Responsive

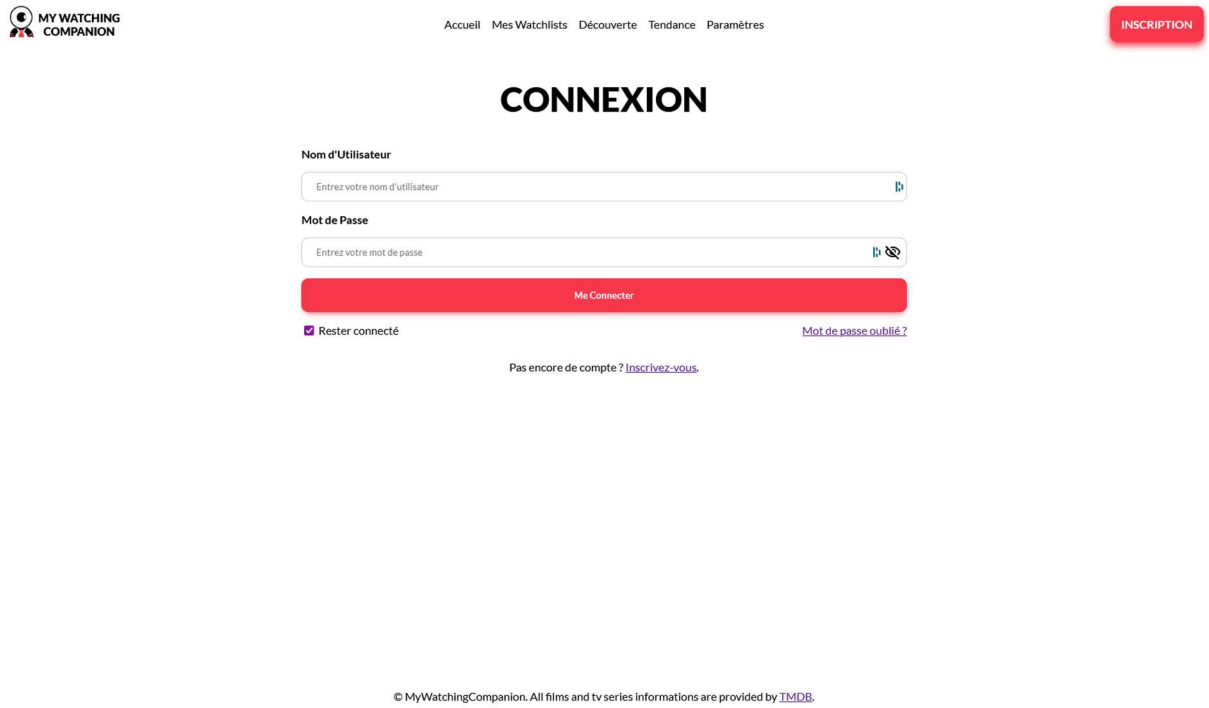
Bien que ce soit un effort d'équipe, j'ai veillé à ce que les pages sur lesquelles je travaillais (authentification, watchlists, artwork) soient responsives et utilisent des composants réutilisables (header, footer, sidebar, que j'ai créés dès les débuts de ce projet pour assurer une interface homogène) définis dans views/components/. Nous avons utilisé une approche CSS modulaire par page.

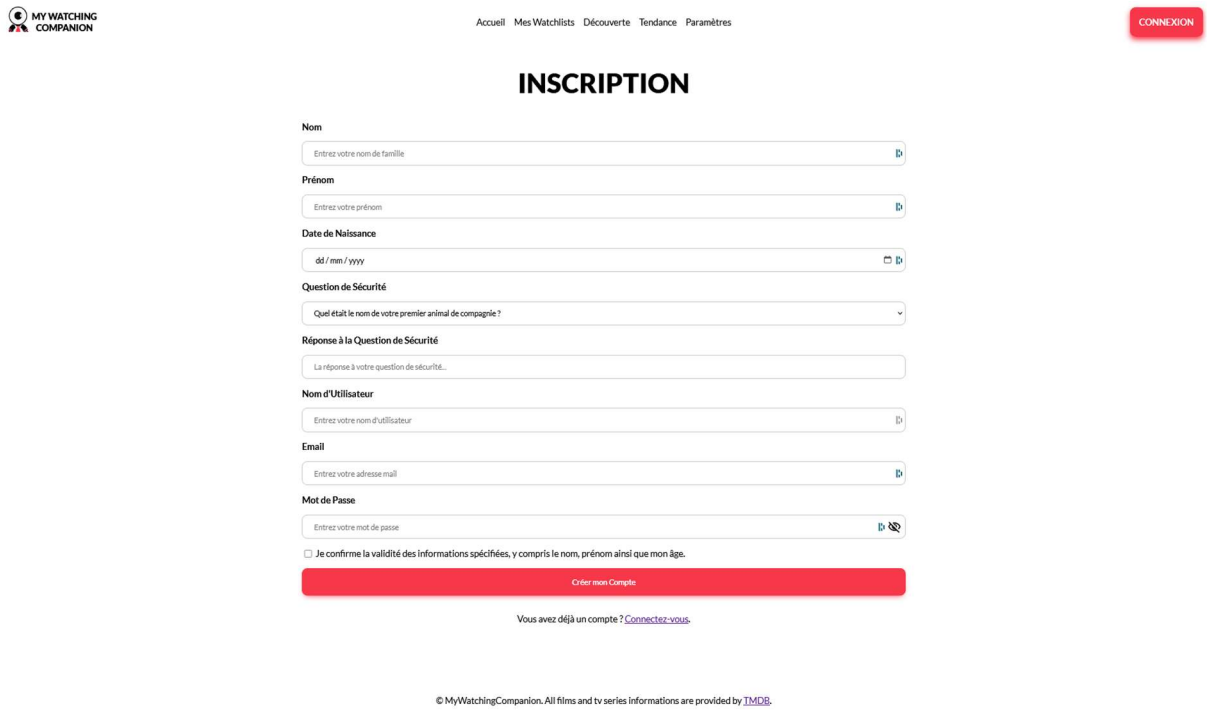
6. Interface Utilisateur (UI)

6.1. Description des Écrans/Vues Principales

J'ai contribué à la conception et à la réalisation de plusieurs vues clés :

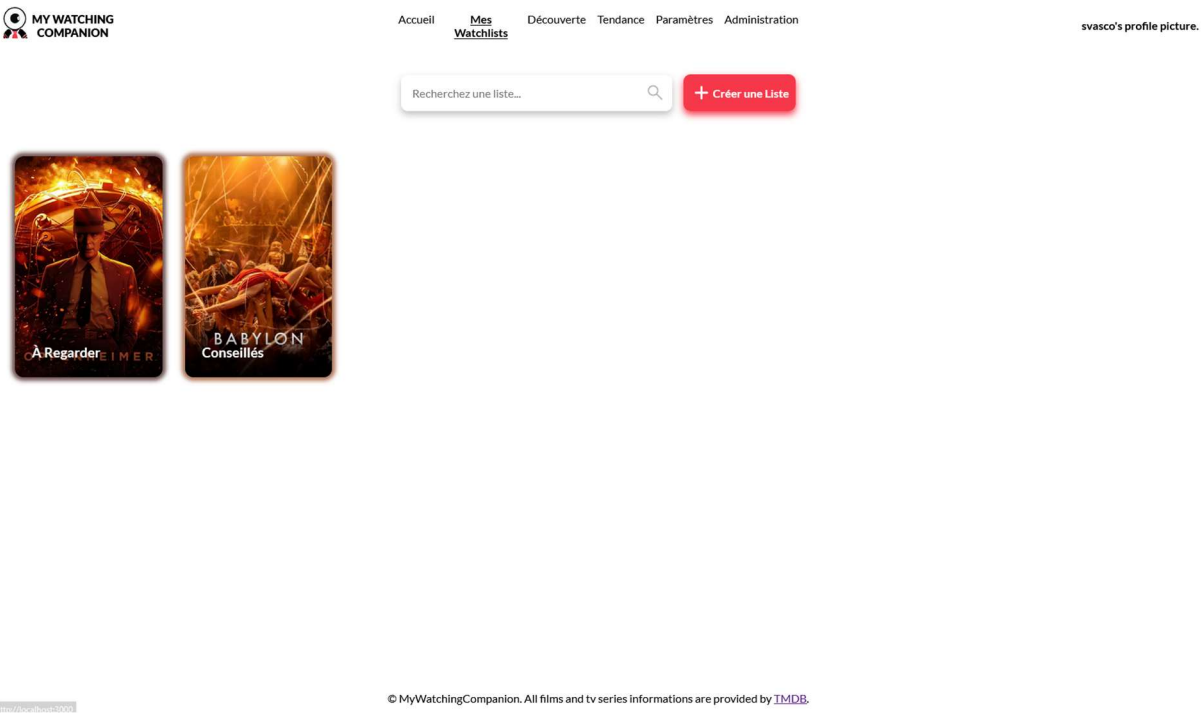
- **Pages d'Authentification** : Des formulaires clairs pour la connexion, l'inscription, et la procédure de mot de passe oublié ;



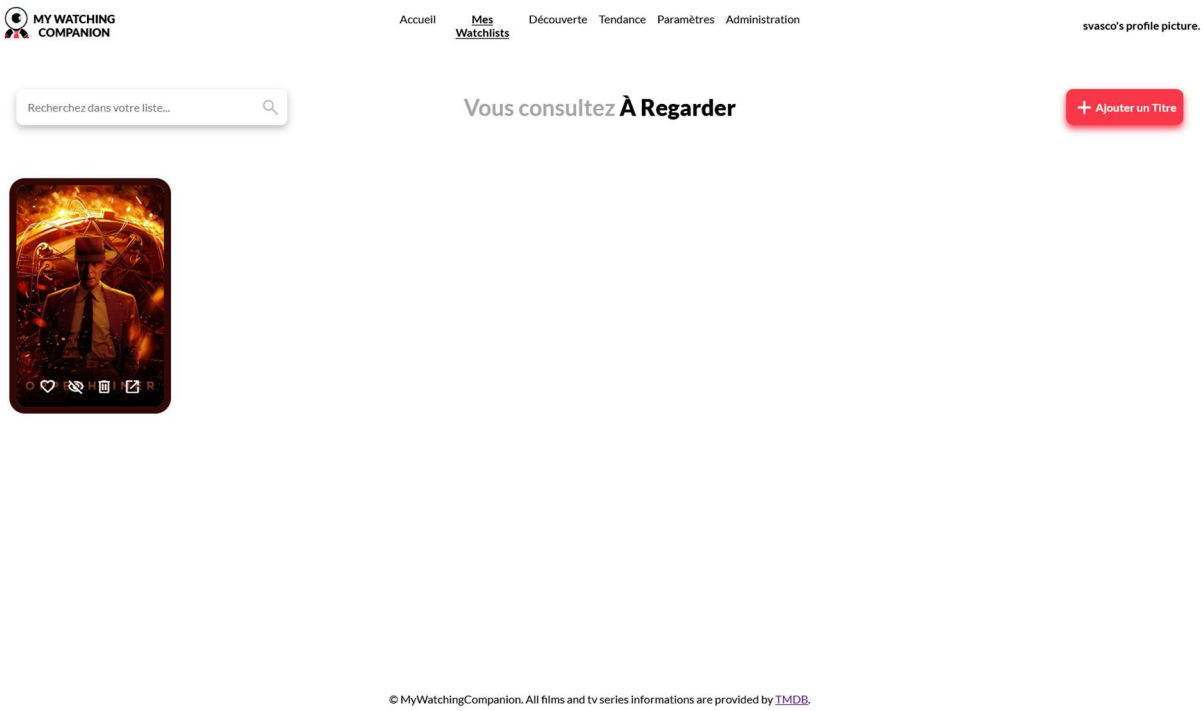


(Taille de la page réduite à 80% pour un meilleur rendu)

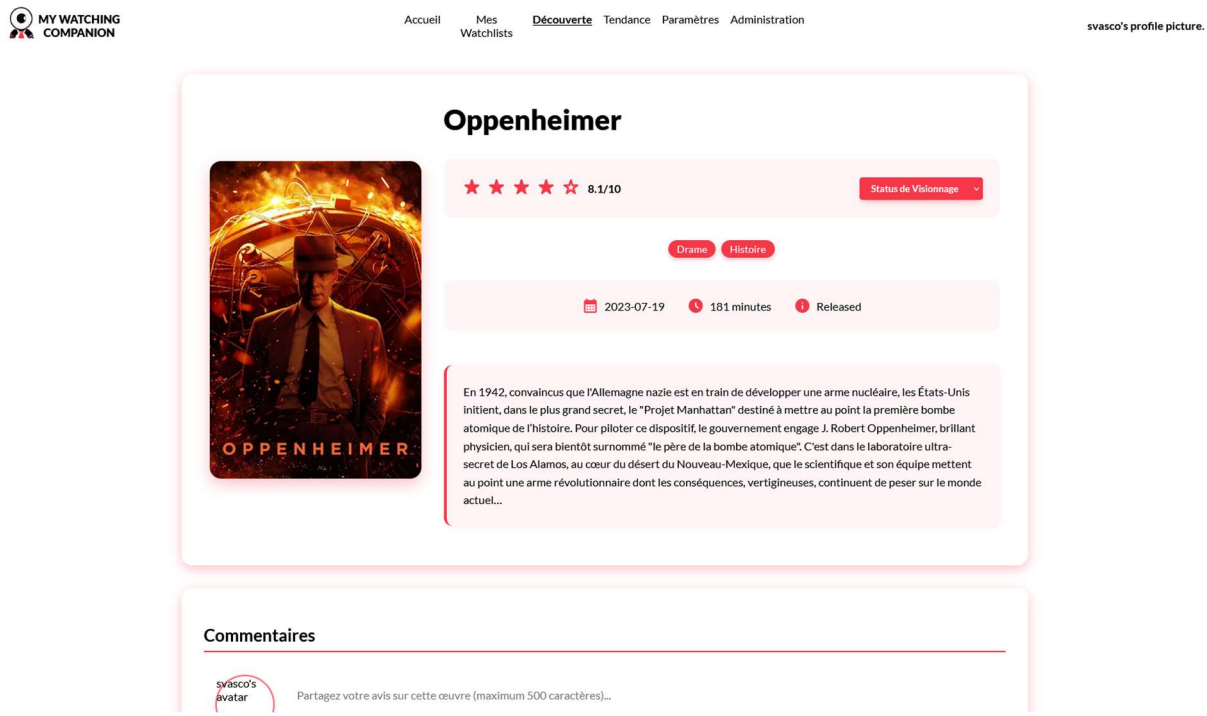
- **Mes Watchlists (my-watchlists.ejs)** : Une vue en grille affichant les différentes watchlists de l'utilisateur, chacune représentée par une "carte" avec une image de couverture et le nom de la liste. J'ai particulièrement travaillé sur l'effet visuel des couleurs dominantes pour ces cartes ;



- **Page d'une Watchlist spécifique (watchlist.ejs)** : Affiche les œuvres contenues dans une liste, avec leur statut de visionnage ;



- **Page d'une Œuvre (artwork.ejs)** : Présente toutes les informations détaillées d'un film ou d'une série, avec le thème de couleurs adaptatif que j'ai implémenté ;



- **Tableau de Bord (Page d'accueil post-connexion)** : Une vue d'ensemble, potentiellement avec des suggestions ou un résumé de l'activité ;
- **Profil et Paramètres Utilisateur** : Pour la gestion des informations personnelles, de la photo de profil, et des questions de sécurité.

6.2. Définition de l'UI et Composants Clés

- Nous avons utilisé **EJS** pour la structure HTML de nos pages ;
- Le style est géré par des fichiers **CSS modulaires** (un par page ou par composant majeur) ;
- Pour l'interactivité côté client, j'ai écrit du **JavaScript vanilla** ;
- Nous avons créé des **composants EJS réutilisables** (header, footer, sidebar) pour assurer la cohérence de l'interface sur l'ensemble de l'application ;
- **Composants UI spécifiques sur lesquels j'ai travaillé :**
    - Les cartes de watchlists avec effet de "shadow" coloré ;
    - Les formulaires d'authentification avec validation ;
    - L'affichage des détails d'artwork avec le thème adaptatif ;
    - Les modales pour la création/édition d'éléments.

6.3. Flux d'Interaction Utilisateur Principaux

J'ai participé à la définition des flux suivants :

1. **Connexion/Inscription :** L'utilisateur remplit le formulaire, les données sont validées, une session est créée, puis il est redirigé ;

2. **Création de Watchlist :** Via un modal, l'utilisateur nomme sa liste. Elle apparaît ensuite sur sa page "Mes Watchlists" ;
3. **Ajout/Gestion d'Œuvres dans une Watchlist :** Recherche d'une œuvre (via TMDB), ajout à une liste, mise à jour du statut de visionnage ;
4. **Consultation d'une Œuvre :** Clic sur une œuvre depuis une liste ou un résultat de recherche, affichage de la page dédiée avec les informations TMDB et le thème dynamique.

7. Points Particuliers et Choix de Conception

Lors du développement, plusieurs choix de conception ont été faits :

1. **Séparation des contrôleurs app et backend** : J'ai trouvé cette distinction utile pour séparer clairement la logique de rendu des vues de la logique des API ;
2. **Sessions avec durées variables** : L'option "Se souvenir de moi" que j'ai implémentée offre plus de flexibilité à l'utilisateur ;
3. **Couleurs dynamiques pour l'UI** : L'utilisation de ColorThief pour adapter le thème des pages d'artwork et l'ombre des cartes de watchlist a été un choix esthétique important pour moi afin de rendre l'interface plus vivante et personnalisée ;
4. **Requêtes SQL paramétrées** : Essentiel pour la sécurité, un point sur lequel j'ai été vigilant.

8. Stratégies de Gestion des Erreurs et de Journalisation

8.1 Gestion des Erreurs

Nous avons mis en place un pattern de gestion d'erreur assez standard dans les contrôleurs backend :

- Utilisation de blocs try...catch pour les opérations asynchrones, notamment les requêtes à la base de données ou aux API externes ;
- Retour de statuts HTTP appropriés (401 pour non autorisé, 404 pour non trouvé, 500 pour erreur serveur) ;
- Envoi de messages d'erreur clairs au format JSON pour les requêtes API ;
- Validation des entrées côté client (dans mes scripts public/scripts/) et côté serveur pour assurer l'intégrité des données.

```
// Mon pattern de gestion d'erreur typique dans les contrôleurs backend
try {
  // Logique métier...
  const result = await executeQuery(query, params);
  res.status(200).json(result);
} catch (error) {
  TraceError(error, `Description de l'opération échouée`); // Ma fonction de log
  res.status(500).json({
    error: "Une erreur est survenue lors de l'opération."
  });
}
```

8.2 Journalisation (Logging)

Nous avons implémenté des fonctions de journalisation simples :

- TraceLogs() : Pour enregistrer les actions utilisateur réussies ou les étapes importantes ;
- TraceError() : Pour enregistrer les erreurs survenues, avec le contexte et la pile d'appel si possible ;
- console.log : Utilisé abondamment en phase de développement pour le débogage.

Les logs sont actuellement affichés dans la console du serveur.

9. Mon Rôle Spécifique dans le Projet

J'ai principalement axé mes efforts sur :

- **Système d'Authentification** : En collaboration initiale, puis j'ai pris en charge le développement complet de l'inscription, la connexion, la gestion des sessions (avec option "Se souvenir de moi"), et la fonctionnalité cruciale d'"oubli de mot de passe" avec questions de sécurité. Cela incluait la création des routes backend sécurisées et des vues EJS correspondantes, en respectant notre charte graphique ;
- **Gestion des Watchlists** : J'ai développé une grande partie de la logique métier pour cette fonctionnalité, incluant les routes Express pour l'ajout, la modification, la suppression de contenus dans les listes, ainsi que la création de la page de gestion et de visualisation de ces watchlists. J'ai particulièrement veillé à la sécurité et à l'optimisation de ces routes, qui nécessitent une authentification utilisateur ;
- **Gestion et Affichage des Œuvres (Artworks)** : J'ai créé les routes et la page dédiée à l'affichage des informations détaillées d'une œuvre. Cela comprend l'intégration avec l'API TMDB, l'affichage des notes, la gestion des commentaires utilisateurs, et l'affichage du statut de visionnage de l'utilisateur pour l'œuvre. L'implémentation du thème de couleurs dynamique basé sur le poster de l'œuvre est une de mes contributions à l'expérience utilisateur sur ces pages.

En résumé, j'ai été fortement impliqué dans le développement backend (routes API, logique métier, interaction avec la base de données) et frontend (création de vues EJS, scripts d'interactivité client) pour ces modules essentiels de l'application.

10. Perspectives d'Évolution et Améliorations Futures

En nous basant sur l'analyse de l'IA et nos propres réflexions, voici quelques pistes d'amélioration pour My Watching Companion :

- **Sécurité Renforcée** :
    - Mettre en place une validation encore plus stricte des entrées utilisateur côté serveur ;
    - Intégrer des mécanismes de limitation de débit (rate limiting) sur les API pour prévenir les abus ;
    - Utiliser des en-têtes de sécurité HTTP plus complets (via des middlewares comme Helmet.js).
- **Performance Optimisée** :
    - Mettre en cache les réponses fréquentes de l'API TMDB pour réduire la latence et le nombre d'appels ;
    - Optimiser la taille et le chargement des images (posters, photos de profil) ;
    - Activer la compression Gzip pour les réponses HTTP.
- **Nouvelles Fonctionnalités** :
    - Permettre le partage de watchlists spécifiques entre utilisateurs de l'application, avec différents niveaux de permission (lecture seule, édition collaborative) ;
    - Développer un système de recommandations personnalisé basé sur les habitudes de visionnage et les œuvres aimées ;
    - Intégrer des notifications en temps réel (par exemple, lorsqu'un ami partage une liste ou commente une œuvre) ;
    - Ajouter des options de tri et de filtrage plus avancées pour les watchlists et les œuvres ;
    - Permettre aux utilisateurs de noter et de rédiger des critiques pour les œuvres.

11. Bilan Personnel et Apprentissages

Le développement de MyWatchingCompanion a été une expérience extrêmement formatrice et enrichissante pour moi durant mon BTS SIO SLAM. Ce projet concret m'a offert l'opportunité de mettre en application un large éventail de compétences que j'avais acquises en cours, tout en me confrontant aux défis réels d'un projet web complet, de sa conception à sa mise en production.

J'ai particulièrement consolidé et approfondi mes compétences dans les domaines suivants :

- **Développement Back-end avec Express.js** : La gestion des routes, le traitement des requêtes, la mise en place de la logique métier et la communication avec la base de données ont été des aspects centraux de mon travail ;
- **Gestion de Bases de Données Relationnelles avec SQL Server** : J'ai pu concevoir la structure des données, écrire des requêtes SQL (parfois complexes, comme pour la récupération des covers de watchlists) et assurer l'intégrité des informations ;
- **Création d'Interfaces Utilisateur Dynamiques avec EJS** : Générer des pages HTML personnalisées en fonction des données utilisateur et de ses actions a été un apprentissage clé pour comprendre le rendu côté serveur ;
- **Déploiement et Hébergement d'une Application Web** : J'ai eu la chance d'assister et de participer à la configuration d'un VPS personnel pour notre projet, ce qui incluait le déploiement de l'application Node.js, la configuration du nom de domaine, la mise en place du HTTPS avec SSL, et la gestion basique du serveur. Ceci a entièrement été réalisé par mon collègue mais le voir faire et assister aux nombreuses erreurs rencontrées a été particulièrement formateur.

Ce projet m'a également permis de développer une vision plus globale de ce qu'implique un développement web structuré, en tenant compte à la fois des besoins fonctionnels de l'utilisateur (simplicité, personnalisation, partage) et des contraintes techniques (sécurité des données, performance, maintenabilité et évolutivité de l'application). L'organisation du code selon une architecture MVC a été un bon exercice pour maintenir un code clair.

Cet investissement personnel important, tant en termes de temps que d'autonomie, m'a offert une occasion précieuse de renforcer mes acquis et de gagner significativement en confiance dans ma capacité à gérer des projets techniques de bout en bout.

12. Conclusion

My Watching Companion est, à mon sens, un projet réussi qui répond bien aux objectifs que nous nous étions fixés dans le cadre de notre formation. En tant que développeur principal sur les fonctionnalités d'authentification, de gestion des watchlists et d'affichage des œuvres, j'ai pu appliquer concrètement mes connaissances et développer de nouvelles compétences en développement full-stack avec Node.js, Express, EJS et SQL Server. Je suis particulièrement satisfait de la manière dont nous avons réussi à intégrer l'API TMDB pour enrichir l'expérience utilisateur et de la mise en place d'un système d'authentification sécurisé. Ce projet a été une étape clé dans mon parcours d'apprentissage.