
Dossier de projet

Projet *MovieLovers*

<https://www.cefii-developpements.fr/arnaud1450/projetCinema/public/>

Du 31 mars au 30 mai 2025

Arnaud BERTHOLLET

Titre Professionnel Développeur Web et Web Mobile

Sommaire

- **Introduction** page 3
- **Contexte du projet** page 5
- ***Eléments significatifs coté Front-End*** page 9
- **Maquettes de l'application** page 9
- **Schéma d'enchainement des maquettes** page 12
- **Intefaces utilisateur, adaptation Web et Web Mobile** page 13
- **Parties dynamiques des interfaces utilisateur** page 19
- ***Eléments significatifs coté Back-End*** page 27
- **Présentation de la base de données** page 27
- **Composants d'accès aux données** page 32
- **Composants métier** page 35
- **Eléments de sécurité de l'application** page 40
- **Jeu d'essai de tests fonctionnels** page 42
- **Déploiement** page 43
- **Description de la veille effectuée** page 44
- **Références** page 45
- **Conclusion** page 46
- **Annexes** page 47

Introduction

Dans le cadre de la formation **Développeur Web et Web Mobile**, j'ai eu l'opportunité de réaliser un projet personnel de fin de parcours visant à mettre en pratique les compétences techniques et méthodologiques acquises tout au long de la formation. Ce projet m'a permis de concevoir, développer et structurer un site web dynamique autour d'une thématique qui me tient à cœur : le cinéma.

Inspiré de plateformes telles que SensCritique et AlloCiné, ce site propose aux utilisateurs de **découvrir des films**, consulter des fiches détaillées sur ces films et les personnes y ayant participé (acteurs et réalisateurs). Le site propose également de lire et publier des critiques.

Le site comporte **2 types d'utilisateurs** : les utilisateurs "simples" et les administrateurs (ayant accès à des fonctionnalités spécifiques).

Il s'agit d'un projet complet qui m'a permis de mobiliser les technologies du **développement web full stack**, avec une **architecture MVC en PHP**, une base de données, du **JavaScript** pour les fonctionnalités dynamiques, **Bootstrap** pour le positionnement, ainsi qu'un **CSS personnalisé** pour l'identité visuelle, en veillant à respecter le **Responsive Web Design** et à corriger les principales **failles de sécurité** vues au cours de la formation : XSS (Cross-site scripting, injections SQL, détournement de session et faille CSRF (Cross Site Request Forgery).

Au-delà des aspects techniques, ce projet m'a également permis de développer des compétences en modélisation de base de données, en gestion de projet, en réflexion UX/UI, et en **autonomie** dans la recherche de solutions.

Liste des compétences du référentiel couvertes par le projet :

Activités types	Compétences professionnelles
Développer la partie front-end d'une application web ou web mobile sécurisée	Installer et configurer son environnement de travail en fonction du projet web ou web mobile
	Maquetter des interfaces utilisateur web ou web mobile
	Réaliser des interfaces utilisateur statiques web ou web mobile
	Développer la partie dynamique des interfaces utilisateur web ou web mobile
Développer la partie back-end d'une application web ou web mobile sécurisée	Mettre en place une base de données relationnelle
	Développer des composants d'accès aux données SQL et NoSQL
	Développer des composants métier coté serveur
	Documenter le déploiement d'une application dynamique web ou web mobile

Contexte du projet

N'étant pas rattaché à une entreprise, **j'ai défini moi-même les besoins du projet et les contraintes techniques** en fonction de mes connaissances, de mes capacités et de mon envie de m'entraîner à pratiquer certaines notions vues pendant la formation.

J'ai endossé le rôle complet du développeur, de la phase de conception à la mise en œuvre technique : développement et déploiement de l'application.

Expression des besoins / fonctionnalité principales

Expression des besoins:

Mon objectif principal a été de concevoir une **plateforme web et web mobile** dédiée aux passionnés de cinéma, et contenant 3 types d'utilisateurs :

- les **visiteurs** n'ayant de compte sur le site : ils ont seulement la possibilité de consulter les différentes pages du site.
- les **simples utilisateurs** qui se sont créés un compte : ils peuvent publier des critiques sur les films une fois qu'ils sont connectés à leur compte.
- les **administrateurs**, disposant de fonctionnalités de CRUD.

L'application doit pouvoir permettre, aux utilisateurs de consulter des fiches détaillées de films, acteurs et réalisateurs ; de consulter des critiques de films laissés par d'autres utilisateurs et d'en publier s'ils sont connectés à leur compte après s'être inscrit.

L'application doit permettre aux administrateurs du site de gérer le CRUD (lire, créer, modifier et supprimer) des 3 principales entités de l'application (films, acteurs et réalisateurs), et aussi de gérer la modération des critiques.

L'application doit être **dynamique**, notamment avec des requêtes **AJAX**, et respecter le principe du **Responsive Web Design**, afin de rendre agréable la navigation de pages en pages et ainsi améliorer **l'expérience utilisateur**.

Le développement de l'application doit également se faire dans un souci d'empêcher les **principales failles de sécurité** d'être exploitées.

Fonctionnalités principales :

- Utilisateurs simples :

- Créer un compte et se connecter sur le site
- Consulter des listes de films regroupés par genres.
- Consulter des fiches détaillées sur les films comprenant une partie "présentation", une partie "casting", et une partie "critiques".
- Consulter des fiches détaillées sur les acteurs et réalisateurs comprenant une partie "présentation", une partie "biographie", et une partie "filmographie".
- Consulter les critiques d'un film (contenant une note et un avis) publiés par d'autres utilisateurs.
- Consulter les critiques d'un film (contenant une note et un avis) publiés par d'autres utilisateurs.

- fonctionnalités spécifiques aux administrateurs :

- Créer d'autres comptes de type administrateur.
- Afficher, ajouter, modifier et supprimer des films, acteurs et réalisateurs.
- Supprimer les critiques de film ne respectant pas les règles de bienséance.

Contraintes techniques / livrables attendus

- Structure de l'application :

L'application est réalisée en **PHP** avec une structure **MVC**.

WampServer permet de l'interpréter coté navigateur et de stocker la base de données via phpMyAdmin.

L'application contient des **composants d'accès aux données** (exemple : UserModel pour accéder aux données utilisateur de la base données).

L'application contient des **composants métier** (exemple : UserController pour manipuler les données utilisateur depuis une vue en utilisant le UserModel).

L'application contient des **vues** (exemple : formLogin pour permettre à un utilisateur de se connecter en utilisant le composant métier UserController).

- Style de l'interface :

- Utilisation de **Bootstrap** pour le positionnement, et pour gérer le responsive lorsque cela est possible.
- **CSS personnalisé** pour donner une entité visuelle à l'application.

- Languages et technologies utilisées :

- Front-end : HTML, CSS, JavaScript
- Back-end : PHP, MySQL et Bootstrap

- Versioning :

Utilisation de **Git** et **GitHub** pour garder un **historique** des différentes versions de l'application durant le développement. Je peux ainsi revenir en arrière si une erreur importante est commise et assurer le maintien d'un code fonctionnel.

Cela m'évite aussi de perdre tout son travail en cas de problème avec mon ordinateur personnel (perte, vol, destruction).

Etant seul à travailler sur le projet et dans un souci de gain de temps, je n'ai pas jugé nécessaire de créer des branches sur mon repository et j'ai donc fait tous les commit directement sur la branche "main".

- Planification des tâches et gestion du projet :

Utilisation de Clickup* pour, premièrement, décomposer le travail en fonctionnalités, puis en tâches et sous-tâches.

Ensuite, Clickup permet de planifier le travail et enfin de suivre son avancement.

exemple : fonctionnalité = Ajouter un film, une des tâche = formulaire d'ajout, une des sous-tâche = vérifier les champs du formulaire

*Clickup est un équivalent de Trello que j'ai trouvé plus pratique à utiliser

- Livrables attendus du projet

- Les maquettes de l'interface utilisateur
- Le modèle conceptuel de données (MCD)
- Le modèle logique de données (MLD)
- Le modèle physique de données (MPD)

- Environnement humain et technique

N'étant pas rattaché à une entreprise, J'ai utilisé des outils personnels ainsi que les ressources fournies par le CEFii :

- Un PC windows 11 :
 - Editeur de code : Visual Studio Code
 - Figma pour les maquettes et schémas
 - Looping pour les modèles de données
 - FileZilla (pour le déploiement)
 - WampServer et ses outils
 - ChatGpt pour créer les données initiales (films, acteurs, réalisateurs, critiques)
- Les ressources fournies par le CEFii :
 - Un serveur pour exécuter nos applications web et héberger nos bases de données

Environnement humain : J'ai travaillé seul sur ce projet depuis chez moi et depuis les locaux du CEFii.

Coté front-end

Les maquettes de l'interface graphique

Les maquettes de l'interface utilisateur ont été réalisés avec Figma

- Maquette de la page d'accueil version desktop :

The wireframe shows the layout of the MovieLovers website. At the top, there's a header with the 'MovieLovers movies' logo, a 'mon profil' dropdown, and a dark mode switch. Below the header are three navigation links: 'Films', 'Acteurs', and 'Réalisateur'. The main content area starts with a section titled 'Les films' which displays movie posters for 'Blade Runner', 'Alien', 'The Matrix', 'Interstellar', and 'Inception'. Below this, there's a 'Science-fiction' category heading and a 'Action' category heading with an ellipsis (...). At the bottom, there's a social media link with icons for Facebook, Twitter, and Instagram, followed by a copyright notice: '© 2025 MovieLovers - Production interne - © Tous droits réservés.'

- Maquette de la Page d'accueil version mobile :

The wireframe illustrates the mobile version of the MovieLovers website. At the top, there's a header bar with a menu icon (three horizontal lines), the "movies" logo (featuring a film strip icon), a user profile icon with a dropdown arrow, and a dark mode moon icon.

The main content area starts with a section titled "Les films". Below it, there are two movie categories: "Science-fiction" and "Action". Under "Science-fiction", there are two movie posters: "Blade Runner" (Ridley Scott, 1982) and "Alien" (Ridley Scott, 1979). Under "Action", there is a placeholder text "...".

At the bottom of the page, there's a footer section encouraging users to follow them on social media, featuring icons for Facebook, Twitter, and Instagram. The footer also includes the copyright notice: "© 2025 MovieLovers - Production interne - © Tous droits réservés."

- Maquette de la page des détails d'un film :

 **MovieLovers**

mon profil ▾ 

Films Acteurs Réalisateur

Interstellar

Accueil Casting Critiques



Sorti en 2014 | 2 h 49 min | Drame, Science-fiction

Réalisé par Christopher Nolan

Avec Matthew McConaughey, Anne Hathaway, Jessica Chastain

Spectateurs

4.6 ★

Synopsis :

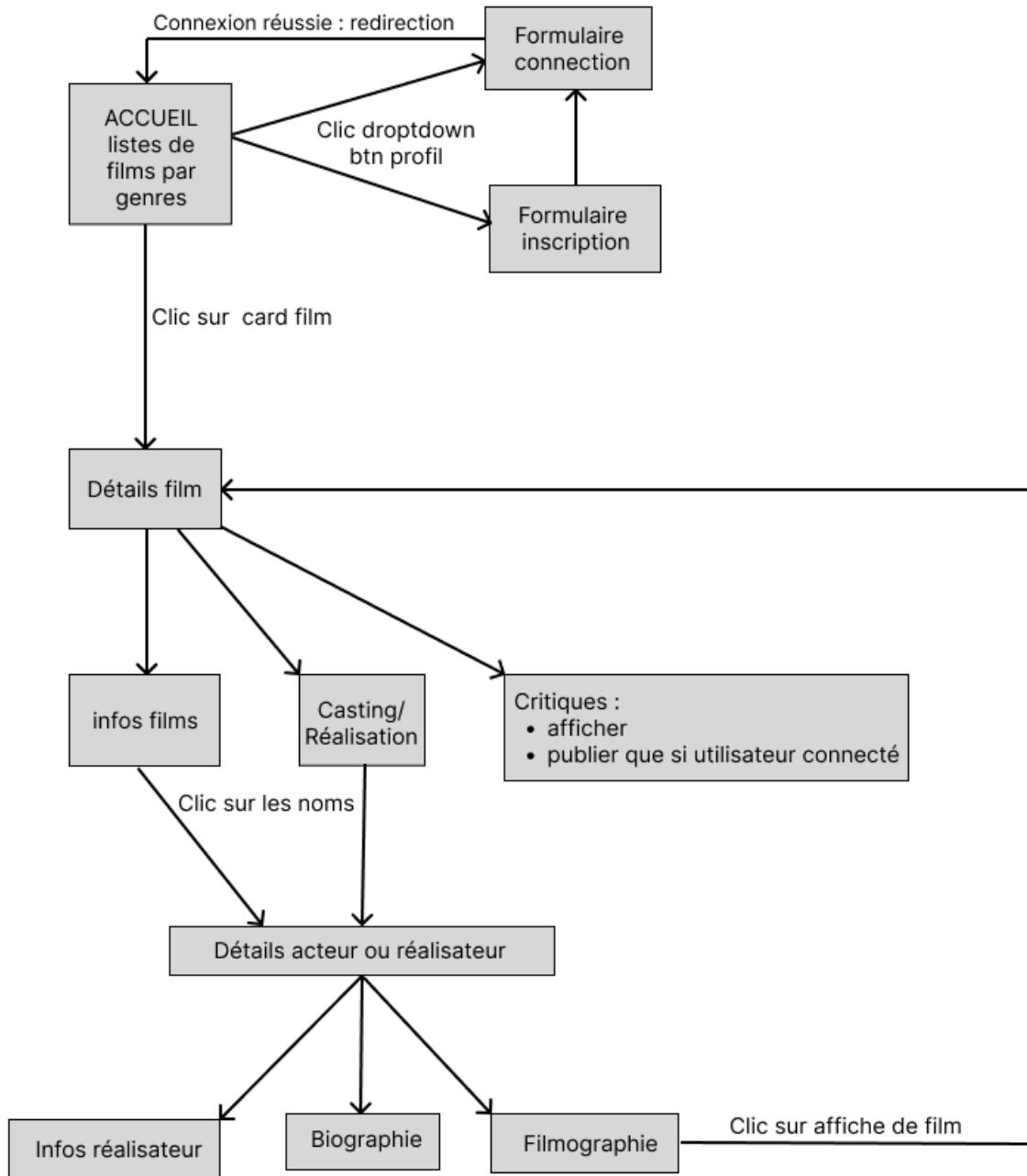
Alors que la Terre dépérît, un ancien astronaute part en mission pour trouver une nouvelle planète habitable pour l'humanité.

Suivez nous sur les réseaux sociaux

© 2025 MovieLovers - Production interne - © Tous droits réservés.

Schéma du parcours utilisateur



Note : Ce schéma illustre le parcours d'un simple utilisateur/visiteur du site.

Il ne comprend pas toutes les fonctionnalités comme par exemple les fonctionnalités spécifiques à l'administrateur.

Réalisations front-end significatives : interfaces statiques

- Interface statique : la page d'accueil

The screenshot shows a static front-end application for movie reviews. At the top, there's a header with the logo 'MovieLovers' and a user dropdown 'superbob (admin)'. Below the header, there are navigation links for 'Films', 'Acteurs', and 'Réalisateur'. The main content area is titled 'Les films' and features two sections: 'Science-fiction' and 'Action'. The 'Science-fiction' section displays seven movie cards with their posters, titles, ratings, and durations:

Titre	Rating	Durée
Blade Runner	3.5 ★	1 h 57 min
Alien - Le Huitième Passager	4.9 ★	1 h 57 min
Matrix	4.0 ★	2 h 16 min
Interstellar	4.6 ★	2 h 49 min
Inception	3.2 ★	2 h 28 min
Total Recall	3.8 ★	1 h 53 min
2001 - L'Odyssey de l'espace	4.1 ★	2 h 29 min

The 'Action' section shows four movie cards with their posters, titles, and names partially visible.

Pour réaliser cette **page statique**, j'ai placé premièrement une balise HTML `<div>` pour englober le contenu à afficher dans la page et à laquelle j'ai attribué la classe bootstrap "**container**" qui a permis ici de **centrer le contenu** de la page.

Les données envoyées par le back-end sur cette page sont contenues dans une variable se présentant sous la forme d'un **tableau à 2 dimensions** contenant :

- Dimension 1 (D1) :
 - key = nom du genre ;
 - value = tableau de films (cf. dimension 2)
- Dimension 2 (D2) :
 - key = index du tableau de cette dimension (pas utilisé) ;
 - value = objet film

Exemple :

`D1['sci-fi']= D2[]` (key= genre "sci-fi", value= D2[genre 0])

`D2[0]= film 0 du genre 0` object film

`D2[1]= film 1 du genre 0` object film

`D2[2]= film 2 du genre 0` object film

`D1['action']= D2[]` (key= action, value= D2[genre 1])

`D2[0]= film 0 du genre 1` object film

`D2[1]= film 1 du genre 1` object film

Pour pouvoir afficher ces données sous la forme désirée, une succession de listes horizontales (scrollables horizontalement avec du JS), j'utilise **2 boucles foreach imbriquées** pour parcourir tous les genres de films, et pour chaque genre, parcourir tous les films qu'il contient.

Les **flexbox** de bootstrap sont utilisées (la classe "d-flex") pour pouvoir afficher les listes horizontales.

Pour garantir la **sécurité** de l'application dans cette page (une des View de mon MVC), j'utilise la fonction PHP **htmlspecialchars()** qui convertit certains caractères prédéfinis (notamment les "<" et ">" chevrons ouvrants et fermants) en entités HTML. Cela évite que des scripts soient lancés en lisant ces variables, dans l'éventualité où des scripts malveillants seraient envoyées en base de données via les formulaires. Cela constitue la pratique recommandée pour se prémunir de l'exploitation de la faille **XSS** (qui redirige via ces scripts les utilisateurs vers des pages "miroirs" pour subtiliser des informations personnelles par exemple).

L'extrait de code :

```
<!-- Conteneur principal de la page -->
<div class="container position-relative">
    <!-- Titre principal de la page -->
    <h2 class="text-center fw-bolder">Les films</h2>

    <?php
    // Parcours dimension 1 : les genres et leurs films
    foreach ($filmsByGenres as $genre => $films) { ?>

        <!-- Titre du "carroussel netflix" : nom du genre -->
        <h3 class="mb-0"><?= htmlspecialchars($genre, ENT_QUOTES, "UTF-8") ?></h3>

        <!-- Carroussel container -->
        <div class="filmScroll d-flex overflow-auto py-3 px-2 gap-3">

            <!-- Flèche gauche (masquée sur petit écran) -->
            <button class="scrollLeft btn darkBtn btnWithBorders position-absolute start-0 translate-middle-y d-none d-md-block z-3"
                style="z-index: 1; margin-top: 100px">
                <i class="bi bi-chevron-left"></i>
            </button>

            <?php
            // Parcours dimension 2 : les films du genre courant
            foreach ($films as $film) { ?>
                <a href="index.php?controller=Film&action=details&id_film=<?= htmlspecialchars($film->id_film, ENT_QUOTES, "UTF-8") ?>"
                    class="filmlink darkTypo" style="text-decoration: none;">
                    <div class="flex-shrink-0" style="width: 150px;">
                        <object data="img/img_films/<?= htmlspecialchars($film->picture, ENT_QUOTES, "UTF-8") ?>">
                            <img alt="<?= htmlspecialchars($film->title, ENT_QUOTES, "UTF-8") ?>" src="img/nopicture.jpg" class="img-fluid rounded shadow-sm mb-1" alt="no picture" style="width: 150px;"/>
                        </object>
                    <p class="text-center fw-bold mt-2 mb-0"><?= htmlspecialchars($film->title, ENT_QUOTES, "UTF-8") ?></p>
                    <?php if (isset($film->average_rating)) { ?>
                        <p class="text-center my-0">
                            <?= htmlspecialchars($film->average_rating, ENT_QUOTES, "UTF-8") ?>
                            <i class="bi bi-star-fill text-warning" style="font-size: small;"></i>
                        </p>
                    <?php
                } ?>
                <p class="text-center my-0"><?= htmlspecialchars($film->duration, ENT_QUOTES, "UTF-8") ?></p>
            </div>
        </a>
    <?php
} ?>

        <!-- Flèche droite (masquée sur petit écran) -->
        <button class="scrollRight btn darkBtn btnWithBorders position-absolute end-0 translate-middle-y d-none d-md-block z-3"
            style="z-index: 1; margin-top: 100px">
            <i class="bi bi-chevron-right"></i>
        </button>
    </div>
<?php
} ?>
</div>
```

- **Interface statique : la page des détails d'un film**

The screenshot shows a static web page for the movie "Interstellar". At the top, there's a header with the "MovieLovers" logo, a user dropdown showing "superbob (admin)", and a moon icon. Below the header is a navigation bar with links for "Films", "Acteurs", and "Réalisateur". The main content area has a title "Interstellar" and a "Modifier" button. Below the title is a movie poster for "INTERSTELLAR". To the right of the poster, it says "Sorti en 2014 | 2 h 49 min | Drame, Science-fiction", "Réalisé par Christopher Nolan", and "Avec Matthew McConaughey, Anne Hathaway, Jessica Chastain". There's also a rating box showing "Spectateurs 4.6 ★ 11 critiques". A "Synopsis" section follows, with the text: "Alors que la Terre dépit, un ancien astronaute part en mission pour trouver une nouvelle planète habitable pour l'humanité."

Pour la partie “Accueil”, qui correspond à la présentation générale du film j’utilise à nouveau les flexbox de bootstrap : une flexbox contenant 2 items : l’affiche du film positionnée sur la gauche et les infos du film sur la droite.

J’utilise la fonction “displayNames” pour formater l’affichage des genres et des noms d’acteurs/réalisateur ; et ce afin qu’ils soient séparés par des virgules et qu’il n’y ait pas de virgule après le dernier élément.

J’englobe ces éléments dans des balises `<a>` afin de pouvoir naviguer vers les pages correspondantes.

Pour le sous-menu menant vers le casting et les critiques, j’utilise la classe “nav-pills” de bootstrap en plus de la classe “nav”, ce qui permet d’afficher seulement le contenu dont l’utilisateur clique sur la “pills” correspondante dans le menu et ainsi de simuler de l’asynchrone ; bootstrap utilisant parfois du javascript.

Des boutons réservés aux administrateurs (modification et suppression) sont intégrés à la page si la variable php super-globale `$_SESSION` contient un utilisateur connecté et si la valeur de la clé “type” (`$_SESSION[“user”][“type”]`) est “admin”.

Pour la partie “Casting”, j’utilise les flexbox pour afficher 2 listes horizontales : une pour les réalisateurs et une pour les d’acteurs du film.

Pour la partie “Critiques”, j’utilise de nouveau les flexbox pour afficher toutes les critiques sous la forme d’une liste verticale, avec comme dernier item le formulaire pour publier des critiques.

J’ai mis en place une structure HTML similaire pour les pages de détails des acteurs et réalisateurs

Comme pour la page "Home", cette page "filmDetails" reçoit la variable "\$film", qui est de type tableau à 1 dimension. Cette variable provient du back-end.

Cette variable contient un tableau de 6 éléments: "détails", "genres", "acteurs", "réalisateur", "reviews" et "average_rating".

Ces 6 éléments sont constitués de:

- "détails" : Un object qui contient des champs "id_film", "title", ...
- "genres" : Un tableau d'objets dont chaque élément contient les champs "id_genre", "name", ...
- "acteurs" : Un tableau d'objets dont chaque élément contient les champs "id_actor", "name", ...
- "réalisateur" : Un tableau d'objets dont chaque élément contient les champs "id_director", "name", ...
- "reviews" : Un tableau d'objets dont chaque élément contient les champs "id_review", "content", "rating", ...
- "average_rating" : Un float qui représente la moyenne des critiques du film courant.

Les extraits de code pour cette page :

```
<?php

use App\Core\CSRFTokenManager as CSRFTokenManager;

function displayNames($items, $max = null, $controller = null)
{
    $count = count($items);
    $max = $max == null ? $count : $max;

    for ($i = 0; $i < $count; $i++) {
        if ($controller == 'Genre') {
            $key = "id_genre";
            $id = $items[$i]→id_genre;
        } elseif ($controller == "Actor") {
            $key = "id_actor";
            $id = $items[$i]→id_actor;
        } elseif ($controller == "Director") {
            $key = "id_director";
            $id = $items[$i]→id_director;
        }

        if ($i == $max - 1) {
            // affichage du dernier élément
            $name = $items[$i]→name;
            <a href="index.php?controller=<?= $controller ?>&action=details&lt;?= $key ?>=<?= htmlspecialchars($id, ENT_QUOTES, "UTF-8") ?>" class="darkTypo menuLinks linksOnHover">
                <b><?= htmlspecialchars($name, ENT_QUOTES, "UTF-8") ?></b>
            </a>
        }
        <?php
            break;
        } else {
            // affichage du ier à l'avant dernier élément
            $name = $items[$i]→name . ", ";
            <a href="index.php?controller=<?= $controller ?>&action=details&lt;?= $key ?>=<?= htmlspecialchars($id, ENT_QUOTES, "UTF-8") ?>" class="darkTypo menuLinks linksOnHover">
                <b><?= htmlspecialchars($name, ENT_QUOTES, "UTF-8") ?></b>
            </a>
        }
    }
}
```

```

    <?php if (!$film) { ?>
        <p class="text-center">Aucune donnée trouvée pour ce film</p>
    <?php } else { ?>

        <!-- TITRE DU FILM -->
        <h2 class="text-center fw-bolder"><?= htmlspecialchars($film["details"]→title, ENT_QUOTES, "UTF-8") ?></h2>

        <!-- MENU DES DÉTAILS DU FILM -->
        <nav class="d-flex justify-content-center mt-4 mb-4">
            <div class="nav nav-pills d-flex flex-wrap overflow-auto pb-3" id="pills-tab" role="tablist">
                <button class="nav-link active menuLinks darkTypo" id="pills-home-tab" data-bs-toggle="pill"
                    data-bs-target="#pills-home" type="button" role="tab" aria-controls="pills-home" aria-selected="true">Accueil</button>
                <button class="nav-link menuLinks darkTypo" id="pills-casting-tab" data-bs-toggle="pill"
                    data-bs-target="#pills-casting" type="button" role="tab" aria-controls="pills-casting" aria-selected="false">Casting</button>
                <button class="nav-link menuLinks darkTypo" id="pills-reviews-tab" data-bs-toggle="pill"
                    data-bs-target="#pills-reviews" type="button" role="tab" aria-controls="pills-reviews" aria-selected="false">Critiques</button>
                <!-- BOUTON MODIFIER ET BOUTON SUPPRIMER -->
                <?php if (isset($_SESSION["user"]) && $_SESSION["user"]["type"] == "admin") { ?>
                    <a class="ms-2 p-2 darkBtn btnWithBorders" href="index.php?controller=Film&action=updateForm&id_film=<?= htmlspecialchars($film["details"]→id_film, ENT_QUOTES, "UTF-8") ?>">Modifier</a>
                    <a class="deleteLink btn btn-danger" href="#" data-entity="Film" data-item="ce film"
                        id="deleteFilm-<?= htmlspecialchars($film["details"]→id_film, ENT_QUOTES, "UTF-8") ?>">Supprimer</a>
                </?php
            <?php } ?>
        </div>
    </nav>

        <!-- CONTENU PRINCIPAL DE LA PAGE -->
        <div class="tab-content" id="pills-tabContent">

            <!-- ACCUEIL DETAILS -->
            <div class="tab-pane fade show active" id="pills-home" role="tabpanel" aria-labelledby="pills-home-tab" tabindex="0">
                <div class="d-flex mb-3">
                    <!-- AFFICHE FILM -->
                    <div id="filmPicture" style="width: 200px;">
                        <object data="img/img_films/<?= htmlspecialchars($film["details"]→picture, ENT_QUOTES, "UTF-8") ?>" class="img-fluid rounded shadow-sm">
                            
                        </object>
                    </div>
                    <!-- INFOS SUR LE FILM -->
                    <div class="ms-3">
                        <!-- ANNEE DE SORTIE | DUREE | GENRES -->
                        <p>
                            Sorti en <b><?= htmlspecialchars($film["details"]→release_year, ENT_QUOTES, "UTF-8") . "</b> | "<?
                                <?php
                                    displayNames($film["genres"], null, "Genre");
                                ?>
                            </p>
                        <!-- REALISATEUR(S) -->
                        <p>
                            Réalisé par <?php displayNames($film["directors"], null, "Director") ?>
                        </p>
                        <!-- LES 3 ACTEURS PRINCIPAUX -->
                        <p>
                            Avec <?php displayNames($film["actors"], 3, "Actor") ?>
                        </p>
                    <?php if (isset($film["average_rating"])) { ?>
                        <!-- NOTE TELESPECTATEURS -->
                        <div class="card text-center p-1" style="width: 115px;">
                            <p class="card-title my-0 fs-6 fw-bold">Spectateurs</p>
                            <p class="fs-4 mb-0 fw-bolder">
                                <?= htmlspecialchars($film["average_rating"], ENT_QUOTES, "UTF-8") ?>
                                <i class="bi bi-star-fill text-warning"></i>
                            </p>
                            <p class="card-text my-0 py-0" style="font-size:x-small;"><small><?= count($film["reviews"]) . " critiques" ?></small></p>
                        </div>
                    <?php
                    } ?>
                </div>
                <!-- SYNOPSIS -->
                <h4>Synopsis :</h4>
                <p><?= htmlspecialchars($film["details"]→synopsis, ENT_QUOTES, "UTF-8") ?></p>
            </div>
        </div>
    
```

La partie casting dans les détails d'un film :

note 1: les acteurs sont affichés sous les réalisateurs et ne sont pas visibles ici...

note 2: les boutons "Modifier" et "Supprimer" ne sont accessibles qu'aux administrateurs

The screenshot shows the casting details for the movie 'Interstellar'. At the top, there's a navigation bar with the 'MovieLovers' logo, a user dropdown 'superbob (admin)', and a dark mode switch. Below the navigation, the movie title 'Interstellar' is displayed. A horizontal menu bar includes 'Accueil', 'Casting' (which is highlighted in blue), 'Critiques', 'Modifier' (disabled), and 'Supprimer' (disabled). Under the movie title, it says 'Réalisateur(s)' and has a button 'Ajouter réalisateur'. Below this, there's a photo of Christopher Nolan and his name 'Christopher Nolan' with a 'Retirer du film' button.

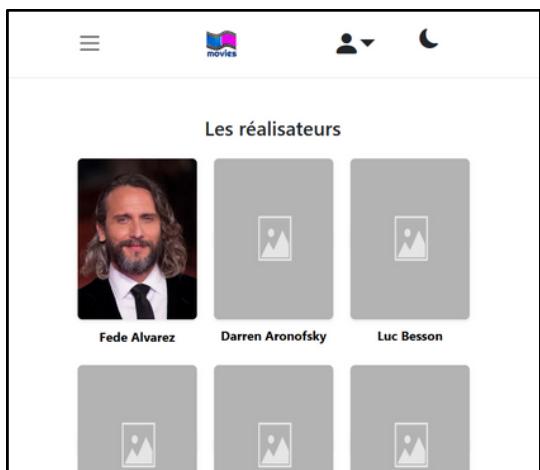
La page qui affiche les réalisateurs :

The screenshot shows a grid of director profiles under the heading 'Les réalisateurs'. The grid consists of two rows of six items each. The first row contains profiles for Fede Alvarez, Darren Aronofsky, Luc Besson, Neill Blomkamp, Danny Boyle, and Park Chan-wook. The second row contains profiles for other directors whose names are not visible. Each profile item has a placeholder image icon.

Sur cette page je dispose les éléments à afficher en utilisant le système de grille de bootstrap avec les classes "grid", "row" et "col".

Comme pour les vues précédemment décrites, j'utilise une boucle "foreach" pour parcourir les éléments à afficher.

La version web mobile de la page “Réaliseurs” :



Pour l'adaption web mobile:

- La barre de navigation est adaptée au web mobile.
- J'utilise les classes col-6 col-sm-4 col-md-3 col-lg-2 (2 colonnes en mobile device, 3 en small device, 4 en medium device = tablette ...).

Pour toutes les pages qui contiennent une image, j'utilise la balise HTML “object” pour afficher l'image, et où l'attribut “data” contient l'URL de l'image.

Avantage de ce mécanisme : Il affiche une image par défaut lorsque l'URL de l'image n'est pas valide. L'image par défaut est indiqué dans une balise à l'intérieur de la balise .

Réalisations front-end significatives : parties dynamiques

Chargements des scripts JS

J'utilise un système générique pour charger dynamiquement les scripts nécessaires à chaque vue du MVC: Les contrôleurs définissent le(s) script(s) à charger pour leur page correspondante, puis le footer charge les scripts lors du chargement de la page.

Example: La classe “FilmController” (méthode “Home”) ajoute une variable “\$script” dans “\$data” qui, lui, est passée via la méthode render().
“\$script” est un tableau où chaque élément contient les attributs “type” (= “module”) et “src” (= URL du script JS).

La page “homeFilm.php” utilise ces attributs pour charger le(s) script(s) au chargement de la page : Dans le footer, de manière générique, une boucle foreach() charge chaque script dans des balises <script>.

Validation dynamique du formulaire d'inscription sur le site

Pour le formulaire pour création de compte, je vérifie les 3 champs de mon formulaire dynamiquement :

1. la validité de l'adresse mail à l'aide d'une RegEx.
2. la longueur du pseudo.
3. la validité du mot de passe choisi par l'utilisateur à chaque nouvelle lettre écrite dans le champ <input> de type password .

Mot de passe

Il est indiqué à l'utilisateur si son mot de passe respecte les 5 critères classiques de sécurité du mot de passe :

- au moins 8 caractères
- au moins une lettre minuscule
- au moins une lettre majuscule
- au moins un chiffre
- au moins un caractère spécial

Formulaire HTML: Une structure permet d'afficher le message d'erreur, pour chaque critère, en cas de pattern non vérifié / condition non remplie (avec des icônes bootstrap pour succès ou échec).

Script JS: Pour chacun des critères, je définis des constantes de type RegEx qui définissent les patterns que l'on souhaite rechercher/vérifier.

Je sélectionne dans le DOM le formulaire avec `querySelector()` et attache un écouteur d'évènement, avec `addEventListener()` sur l'évènement submit.

Premièrement, J'empêche la soumission du formulaire avec `e.preventDefault()`.

Ensuite, la variable `isValid` (définie à vrai par défaut) passe à faux en cas de condition(s) non remplie(s) dans les boucles `if()` qui vérifient les champs du formulaire.

Pour le mot de passe je définis donc 2 fonctions :

- `validateOne()` qui vérifie un des critères de validation du mot de passe (ex: au moins une lettre minuscule) avec `regex.exec(password)`. Cette fonction nous retourne 1 en cas de correspondance entre la RegEx et la chaîne de caractère à vérifier et 0 sinon.
- `validateAll()` qui valide l'ensemble du mot de passe, en appelant `validateOne()` pour toutes les chaînes de caractères à vérifier. Chaque appel incrémenté la variable `checkCount` définie par défaut à 0.

Bootstrap: J'utilise une barre de progression.

Dans la fonction `validateAll()`, un `switch()` sur la variable `checkCount`, passée en argument, modifie la couleur de la barre de progression : `checkCount < 3` → rouge, `checkCount < 5` → orange, et sinon vert). La couleur est gérée en ajoutant/enlevant les classes bootstrap `bg-success` / `bg-danger`.

Pour la largeur de la barre de progression, j'ai défini une valeur "percent" (qui a comme valeur la valeur de `checkCount` * 20, et ainsi obtenir une variable en pourcentage qui sert pour modifier dans le DOM la largeur de l'élément ayant la classe "progress-bar".

Enfin, après avoir appelé `validateAll()`, et si la variable `isValid` est toujours true, je soumet le formulaire avec la fonction `validateForm()` située dans mon `main.js` (en méthode AJAX) ; si `isValid` est false, j'affiche les messages d'erreurs adéquats.

les extraits de code :

pour l'HTML :

```
<!-- CHAMP MDP -->
<div class="mb-3">
  <label for="password" class="form-label">Mot de passe</label>
  <input id="password" class="form-control form-control-sm" type="password" name="password" autocomplete="new-password" placeholder="Entrez votre mot de passe">

  <!-- ZONE DE CONTROLE DU MDP -->
  <div id="psw-strength" class="w-100 px-1 py-4" hidden>
    <!-- BARRE DE PROGRESSION -->
    <div class="progress mb-3" style="height: 5px">
      <div class="progress-bar bg-success" role="progressbar" style="width: 0" aria-valuenow="2" aria-valuemin="0" aria-valuemax="5">
      </div>
    </div>
    <!-- PASSWORD STRENGTH INDICATORS -->
    <div>
      <i id="xmarkLength" class="fa-solid fa-xmark text-danger"></i>
      <i id="checkLength" class="fa-solid fa-check text-success"></i>
      <label class="ms-2">8 caractères minimum</label>
    </div>
    <div>
      <i id="xmarkMaj" class="fa-solid fa-xmark text-danger"></i>
      <i id="checkMaj" class="fa-solid fa-check text-success"></i>
      <label class="ms-2">Une majuscule</label>
    </div>
    <div>
      <i id="xmarkMin" class="fa-solid fa-xmark text-danger"></i>
      <i id="checkMin" class="fa-solid fa-check text-success"></i>
      <label class="ms-2">Une minuscule</label>
    </div>
  </div>
```

Pour le fichier js

```
const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[\W_]).{8,}$/;
const lengthRegex = /^.{8,}$/;
const minRegex = /(.*[a-z])/;
const majRegex = /(.*[A-Z])/;
const numberRegex = /(.*\d)/;
const specialCharRegex = /(.*[\W_])/;
```

```

// _____
// validation du formulaire
// _____
document.querySelector('#formSignUp').addEventListener('submit', function (e) {
  e.preventDefault();
  let isValid = true;

  // Validation email
  if (emailRegex.test(email.value) === false) {
    emailError.textContent = "L'adresse mail n'est pas valide.";
    emailError.hidden = false;
    isValid = false;
  } else {
    emailError.hidden = true;
  }
  // Validation de la longueur du pseudo
  if (pseudo.value.length < 3) {
    pseudoError.textContent = "Le pseudo doit contenir au moins 3 caractères";
    pseudoError.hidden = false;
    isValid = false;
  } else {
    pseudoError.hidden = true;
  }

  // Validation password
  if (!validateAll(inputPassword.value)) {
    isValid = false;
  }

  // _____ Formulaire validé : soumission du formulaire
  validateForm(new FormData(this), isValid);
})

```

```

function validateAll(password) {
  pswStrength.hidden = false;
  let res = false;
  let checkCount = 0;

  checkCount += validateOne(lengthRegex, password, checkLength, xmarkLength); // Validation de la longueur
  checkCount += validateOne(majRegex, password, checkMaj, xmarkMaj); // Présence d'une majuscule
  checkCount += validateOne(minRegex, password, checkMin, xmarkMin); // Présence d'une minuscule
  checkCount += validateOne(numberRegex, password, checkNumber, xmarkNumber); // Présence d'une chiffre
  checkCount += validateOne(specialCharRegex, password, checkSpecialChar, xmarkSpecialChar); // Présence d'un caractère spécial

  const percent = `${checkCount * 20}%`;
  bsProgressBar.style.width = percent;

  bsProgressBar.classList.remove('bg-success', 'bg-warning', 'bg-danger');

  switch (checkCount) {
    case 0:
    case 1:
    case 2:
      //red
      bsProgressBar.classList.add('bg-danger');
      break;

    case 5:
      // green
      bsProgressBar.classList.add('bg-success');
      res = true
      break;

    default:
      // orange
      bsProgressBar.classList.add('bg-warning');
      break;
  }
  return res;
}

```

```

    // _____
    // validation du mot de passe
    // _____
    function validateOne(regex, string, check, xmark) {
        if (regex.exec(string)) {
            check.hidden = false;
            xmark.hidden = true;
            return 1;
        } else {
            check.hidden = true;
            xmark.hidden = false;
            return 0;
        }
    }

```

Ajout d'acteurs ou réalisateurs à un film par l'administrateur

Pour ajouter du dynamisme dans l'ajout de réalisateurs ou d'acteurs, j'utilise une fenêtre modale masquée par défaut et qui s'ouvre lors du clic sur le bouton "ajouter réalisateur" ou le bouton "ajouter acteur", dans la partie "Casting" de la vue des détails d'un film.

Si l'acteur clique sur "ajouter acteur", cela ajoute une information passée dans l'attribut "id" du lien cliqué, que l'on exploite dans le fichier JS pour déterminer si l'utilisateur veut ajouter un acteur ou un réalisateur, et ainsi modifier le contenu de la modale.

Le contenu de la modale est modifié dynamiquement pour afficher un formulaire. Elle contient une barre de recherche d'acteurs ou réalisateurs qui sont obtenus en asynchrone avec AJAX. Les personnages choisis sont ensuite ajouté au film en asynchrone quand l'administrateur appuie sur le bouton "ajouter".

les extraits de code HTML pour cette fonctionnalité :

```


<div class="container-fluid mt-5">
    <div class="d-inline-flex align-items-center mb-4">
        <h3 class="mb-0">Acteurs</h3>
        
        <?php if (isset($_SESSION["user"]) && $_SESSION["user"]["type"] == "admin") { ?>
            <a href="#" id="addActor" class="fs-6 ms-3 p-2 addToFilmOpenModal darkBtn btnWithBorders">Ajouter acteur</a>
        <?php
        } ?>
    </div>

    <div class="row">
        <?php if ($film["actors"] == []) { ?>
            <p>Aucun acteur associé à ce film pour l'instant</p>
            <?php
        } else {
            <foreach ($film["actors"] as $actor) : ?>...
            <?php endforeach;
        } ?>
    </div>
</div>

```

```

<!-- Modal addToFilm -->
<div id="myModal" class="modal">
  <div class="modal-content lightForm formDarkMode">
    <div hidden id="filmID<?= htmlspecialchars($film["details"])>><id_film, ENT_QUOTES, "UTF-8") ?>"></div>
    <div class="d-flex justify-content-center">
      <h2 id="modalTitle" class="text-center pb-0"></h2>
      <a href="#" class="close-btn darkBtn btnWithBorders px-2" title="Retour en arrière"><i class="bi bi-x-lg"></i></a>
    </div>
    <p id="resultMsg"></p>
    <div class="d-flex mt-3">
      <input type="text" id="modalSearch" class="form-control">
    </div>
    <div id="searchResults" ></div>
  </div>
</div>

```

L'extrait de code pour le script JS pour l'ouverture et fermeture de la modale :

```

const modal = document.querySelector("#myModal");
const openBtns = document.querySelectorAll(".addToFilmOpenModal");
const closeBtn = document.querySelector(".close-btn");
const title = document.querySelector("#modalTitle");
const searchInput = document.querySelector("#modalSearch");
const searchResults = document.querySelector("#searchResults");
const id_film = document.querySelector(".modal-content").firstElementChild.id.replace("filmID", "");

let entity = "";

// OUVERTURE DE LA MODALE
openBtns.forEach(btn => {
  btn.addEventListener("click", function () {
    modal.style.display = "flex";
    document.body.classList.add("noscroll");

    // Détection de l'entité recherchée
    if (this.id === "addActor") {
      entity = "Actor";
      title.textContent = "Ajouter acteur";
      searchInput.setAttribute("placeholder", "Rechercher un acteur");
    } else if (this.id === "addDirector") {
      entity = "Director";
      title.textContent = "Ajouter réalisateur";
      searchInput.setAttribute("placeholder", "Rechercher un réalisateur");
    }

    searchInput.value = "";
    searchResults.innerHTML = "";
    document.querySelector('#resultMsg').innerHTML = "";
    searchInput.focus();
  });
});

// FERMETURE DE LA MODALE
function closeModal() {
  modal.style.display = "none";
  document.body.classList.remove("noscroll");
  searchResults.innerHTML = "";
  searchInput.value = "";
  document.querySelector('#resultMsg').innerHTML = "";
  entity = "";
}

closeBtn.addEventListener("click", closeModal);
window.addEventListener("click", (e) => {
  if (e.target === modal) closeModal();
});

```

L'extrait de code pour la recherche d'acteurs/réalisateurs :

```
// RECHERCHE EN DIRECT
let searchTimeout;
searchInput.addEventListener("input", function () {
  clearTimeout(searchTimeout); // évite les appels trop rapides
  const query = this.value.trim();

  if (query.length < 2) {
    searchResults.innerHTML = "";
    return;
  }

  searchTimeout = setTimeout(async () => {
    try {
      const response = await fetch(`index.php?controller=${entity}&action=search&query=${encodeURIComponent(query)}`);
      const data = await response.json();

      searchResults.innerHTML = ""; // nettoyage précédent

      if (data.length === 0) {
        searchResults.innerHTML = "<li>Aucun résultat</li>";
        return;
      }

      data.forEach(person => {
        const li = document.createElement("li");
        li.id = `${entity.toLowerCase()}${person.id}Li`;
        li.className = "text-end my-3";
        li.innerHTML = `<b>${person.name}</b>
          <button type="button" id="${entity.toLowerCase() + person.id}" class="btnAddToFilm darkBtn btnWithBorders ms-2 p-1">Ajouter</button>`;
        searchResults.appendChild(li);
      });
    } catch (error) {
      searchResults.innerHTML = "<p class='text-danger'>Erreur lors de la recherche</p>";
    }
  }, 300); // délai pour éviter les appels trop fréquents
});
```

J'utilise ici de l'asynchrone pour la recherche grâce à l'API *fetch* et au mots clé *async/await* :

- le mot clé *async* avant une fonction indique que la fonction retourne une promesse qui sera résolue une fois le travail asynchrone de la fonction terminé.
- Le mot clé *await* peut seulement être utilisé dans une fonction *async*. Il met en pause l'exécution du code situé à droite du mot clé. Une fois la promesse résolue, le code reprend son exécution.

Je récupère depuis le Back-End, dans la variable *data*, un tableau d'objets "acteurs" ou "réaliseurs".

Je parcours ensuite *data* avec la boucle *foreach* pour manipuler les acteurs ou réalisateurs (variable "person" dans le script ci-dessus).

Pour chaque acteur/réalisateur, on modifie dans le DOM le contenu de la modale pour y afficher une liste de personnes (l'élément ** prendra pour id le *id* de "person").

L'extrait de code pour l'ajout au film :

Pour ajouter un film, j'utilise l'API Fetch, en passant dans l'URL

- l'entité ("Actor" ou "Director") pour appeler la bonne méthode de contrôleur
- Le *id* de l'acteur ou réalisateur récupéré grâce à e.target

Note: Le *id* a été récupérée plus haut dans le code, lors de la recherche asynchrone de tous les acteurs ou réalisateurs existants.

```
// DELEGATION SUR LES BOUTONS "AJOUTER"
searchResults.addEventListener("click", function (e) {
  if (e.target && e.target.classList.contains("btnAddToFilm")) {
    const btn = e.target;
    const id = btn.id.replace(entity.toLowerCase(), "");

    fetch(`index.php?controller=Film&action=add${entity}ToFilm&id_film=${id_film}&id_${entity.toLowerCase()}=${id}`, {
      method: "GET"
    })
      .then(response => response.json())
      .then(data => {
        if (data.success === true) {
          document.querySelector(`#${entity.toLowerCase() + id}Li`).remove();
          document.querySelector('#resultMsg').innerHTML =
            '<p class="text-success">' + data.message + '</p>';
        } else {
          document.querySelector('#resultMsg').innerHTML =
            '<p class="text-danger">' + data.message + '</p>';
        }
      })
      .catch(error => {
        document.querySelector('#resultMsg').innerHTML =
          '<p class="text-danger">Une erreur est survenue lors de l\'ajout</p>';
      });
  }
});
```

Coté back-end

Présentation de la base de données

Règles de gestion et dictionnaire de données

Règles de gestions :

- Un film est rattaché à un ou plusieurs réalisateurs.
- Un réalisateur est rattaché à un ou plusieurs films.
- Un film est rattaché à plusieurs acteurs.
- Un acteur est rattaché à un ou plusieurs films.
- Un film est rattaché à un ou plusieurs genres.
- Un genre est rattaché à un ou plusieurs films.
- Une sélection de films est rattachée à un utilisateur et à plusieurs films.
- Un utilisateur peut être rattaché à aucune ou plusieurs sélections de films.
- Un commentaire est rattaché à un seul utilisateur et à une seule sélection de films.
- Un utilisateur peut être rattaché à aucun ou plusieurs commentaires.
- Une critique est rattachée à un utilisateur et à un film.
- Un utilisateur peut être rattaché à aucune ou plusieurs critiques.

Dictionnaire de données :

film :

- *id* : l'identifiant du film ; INT AUTO-INCREMENT ; NOT NULL.
- *title* : le titre du film ; VARCHAR(250) ; NOT NULL.
- *release_year* : année de sortie du film ; INT(4) ; NOT NULL.
- *duration* : durée du film en nombres de minutes ; INT ; NOT NULL.
- *synopsis* : synopsis du film ; TEXT ; NULL.
- *picture* : affiche du film ; VARCHAR(250) ; le chemin vers la photo dans la structure de l'appli ; NULL.

director :

- *id* : l'identifiant du réalisateur. INT AUTO-INCREMENT ; NOT NULL.
- *firstname* : prénom du réalisateur ; VARCHAR(50) ; NOT NULL.
- *lastname* : nom de famille du réalisateur ; VARCHAR(50) ; NOT NULL.
- *birth_date* : date de naissance du réalisateur ; DATE(format : YYYY/MM/DD) ; NOT NULL.
- *death_date* : date de décès du réalisateur ; DATE(format : YYYY/MM/DD) ; NULL.
- *biographie* : biographie du réalisateur ; TEXT ; NULL.
- *nationality* : nationalité du réalisateur ; VARCHAR(50) ; NULL.
- *picture* : photo du réalisateur ; VARCHAR(250) ; le chemin vers la photo dans la structure de l'appli ; NULL.

selection :

- *id* : identifiant de la sélection ; INT AUTO-INCREMENT ; NOT NULL.
- *title* : titre de la sélection faite par un utilisateur ; VARCHAR(250) ; NOT NULL.
- *id_user* : identifiant de l'utilisateur ayant publié la sélection ; INT ; NOT NULL.
- *publication_date* : date de publication de la sélection de films et/ou séries ; DATE (format : YYYY/MM/DD) ; NOT NULL.

review :

- *id* : identifiant de la critique d'un film ou d'une série; INT AUTO-INCREMENT ; NOT NULL.
- *content* : texte contenant la critique d'un film ou d'une série ; TEXT ; NULL.
- *rating* : note donnée au film ou à la série par un utilisateur ; INT(1) ; NOT NULL.
- *publication_date* : date de publication de la critique d'un film ou d'une série ; DATE (format : YYYY/MM/DD) ; NOT NULL.
- *id_user* : identifiant de l'utilisateur ayant publié la critique ; INT ; NOT NULL.
- *id_film* : identifiant du film ou de la série faisant l'objet de la critique ; INT ; NOT NULL.

comment :

- *id* : identifiant du commentaire d'une sélection de films et/ou séries ; INT AUTO-INCREMENT ; NOT NULL.
- *content* : texte du commentaire d'une sélection de films et/ou séries ; TEXT ; NOT NULL.
- *publication_date* : date de publication d'un commentaire d'une sélection de films et/ou séries ; DATE (format : YYYY/MM/DD) ; NOT NULL.
- *id_user* : identifiant de l'utilisateur ayant publié le commentaire : INT ; NOT NULL.
- *id_selection* : identifiant de la sélection faisant l'objet du commentaire ; INT ; NOT NULL.

actor :

- *id* : l'identifiant de l'acteur. INT AUTO-INCREMENT ; NOT NULL.
- *firstname* : prénom de l'acteur ; VARCHAR(50) ; NOT NULL.
- *lastname* : nom de famille de l'acteur ; VARCHAR(50) ; NOT NULL.
- *birth_date* : date de naissance de l'acteur ; DATE (format : YYYY/MM/DD) ; NOT NULL.
- *death_date* : date de décès de l'acteur ; DATE(format : YYYY/MM/DD) ; NULL.
- *biographie* : biographie du réalisateur ; TEXT ; NULL.
- *nationality* : nationalité du réalisateur ; VARCHAR(250) ; NULL.
- *picture* : photo de l'acteur ; VARCHAR(250) ; le chemin vers la photo dans la structure de l'appli ; NULL.

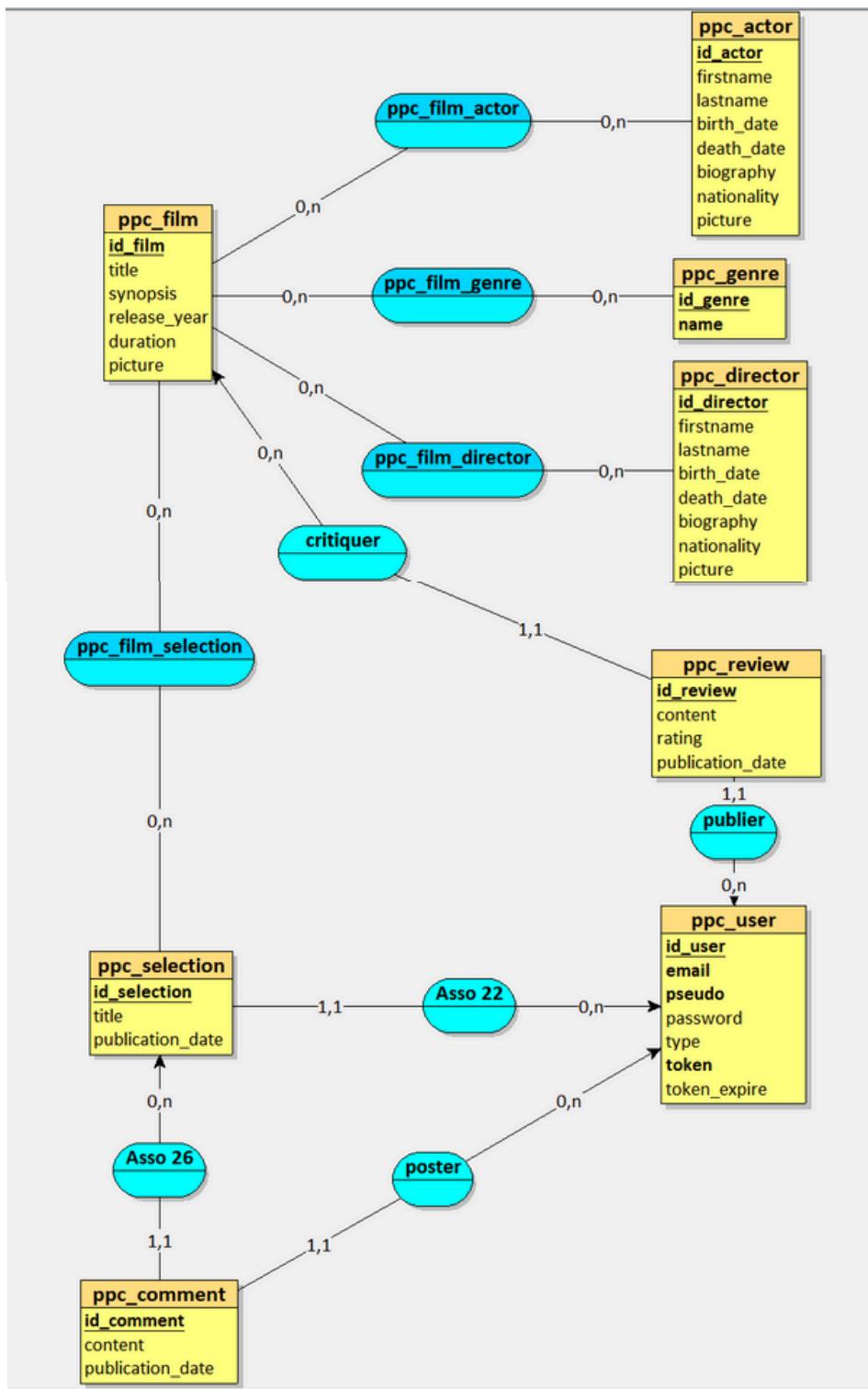
genre :

- *id* : l'identifiant du genre des films et séries; INT AUTO-INCREMENT ; NOT NULL.
- *name* : le nom du genre des films et séries ; VARCHAR(50) ; NOT NULL.

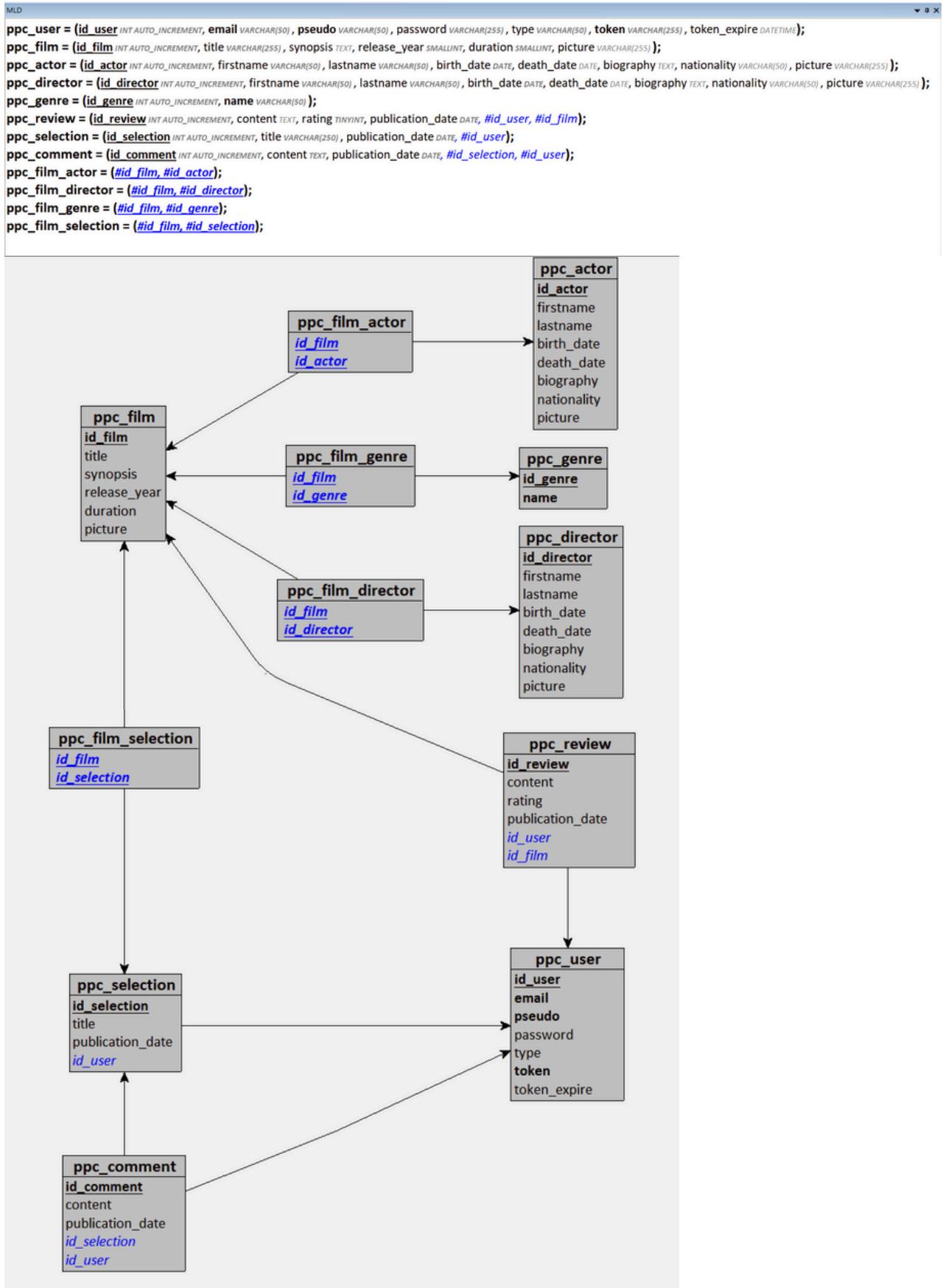
user :

- *id* : identifiant de l'utilisateur du site ; INT AUTO-INCREMENT ; NOT NULL.
- *email* : email de l'utilisateur ; VARCHAR(50) ; NOT NULL.
- *pseudo* : pseudo choisi par l'utilisateur ; VARCHAR(50) ; NOT NULL.
- *password* : mot de passe de l'utilisateur ; VARCHAR(250) ; NOT NULL.
- *type* : type d'utilisateur (administrateur ou simple utilisateur) ; NOT NULL.
- *expiration_token* : jeton d'expiration pour utilisé pour la fontionnalité de réinitialisation du mot de passe en cas de mot de passe oublié par l'utilisateur ; VARCHAR ; NOT NULL.

Modèle Conceptuel de données (MCD)



Modèle logique de données (MLD)



extrait du script de création des tables (MPD)

```
CREATE TABLE ppc_film_genre(
    id_film INT,
    id_genre INT,
    PRIMARY KEY(id_film, id_genre),
    FOREIGN KEY(id_film) REFERENCES ppc_film(id_film),
    FOREIGN KEY(id_genre) REFERENCES ppc_genre(id_genre)
);

CREATE TABLE ppc_film_selection(
    id_film INT,
    id_selection INT,
    PRIMARY KEY(id_film, id_selection),
    FOREIGN KEY(id_film) REFERENCES ppc_film(id_film),
    FOREIGN KEY(id_selection) REFERENCES ppc_selection(id_selection)
);
```

J'ai créé la base de données avec phpMyAdmin : C'est une application de gestion pour les systèmes de gestion de base de données MySQL et MariaDB. Elle est intégrée dans WampServer qui est installé sur mon PC.

Le système de gestion de ma base de données est MySQL.

Présentation de composants d'accès aux données

Accès : Pour accéder / se connecter à ma base de données depuis l'application, j'ai défini une class abstraite *DbConnect* dans le fichier *DbConnect.php* (situé dans *app/core/*) dans laquelle je définis

- 2 attributs : *\$connection* et *\$request*
- un constructeur qui fait appel à PDO (PHP Data Objects)
- PDO est une extension qui définit l'interface d'accès à une base de données avec PHP. Elle est orientée objet, la classe s'appelle "PDO".

Requêtes : Dans les classes "*Model*" de mon MVC où sont situées les méthodes d'accès aux données qui contiennent les requêtes SQL, j'utilise les attributs *\$request* et *\$connection* de la classe *DbConnect* pour exécuter les requêtes avec PDO.

Sécurité : Pour protéger la base de données des injections SQL, j'utilise les requêtes préparées avec les méthodes de PDO : *prepare()*, *bindValue()*, et *execute()*. Je passe à *bindValue()* des marqueurs nommés.

Modèles et entités : Dans les classes "Model" du MVC, j'importe les classes "Entities" nécessaires, afin d'utiliser les *getters* pour récupérer les valeurs des attributs des instances d'entités.

Ces valeurs ont été préalablement hydratées dans les méthodes des classes "Controller", selon les besoins des différents méthodes des classes "Model".

Pour ces imports de classes, j'ai utilisé une classe *Autoloader* (située dans *Autoloader.php*) et les *namespaces*.

Ajout d' un film en base de donnée

extrait de code de la méthode `add()` de la class FilmModel :

```
// AJOUTER UN FILM
// _____
public function add(Film $film, $genresArray = [])
{
    try {
        // PREPARATION DE LA REQUETE SQL
        $this->request = $this->connection->prepare(
            "INSERT INTO
                ppc_film
            VALUES
                (null,
                :title,
                :synopsis,
                :release_year,
                :duration,
                :picture)"
        );
        $this->request->bindValue(":title", $film->getTitle(), PDO::PARAM_STR);
        $this->request->bindValue(":synopsis", $film->getSynopsis(), PDO::PARAM_STR);
        $this->request->bindValue(":release_year", $film->getRelease_year(), PDO::PARAM_INT);
        $this->request->bindValue(":duration", $film->getDuration(), PDO::PARAM_INT);
        $this->request->bindValue(":picture", $film->getPicture(), PDO::PARAM_STR);

        // EXECUTION DE LA REQUETE SQL ET RETOUR DE L'EXECUTION
        $success = $this->request->execute();
        if ($success) {
            $id_film = $this->connection->lastInsertId();
            foreach ($genresArray as $genre) {
                $this->addGenreToFilm($id_film, $genre->getId_genre());
            }
        }
        return $success;
    } catch (PDOException $e) {
        // return $e->errorInfo[1];
        return false;
    }
}
```

Ici , je passe comme arguments à la méthode `add()` :

- un objet `$film` qui est une instance de l'entité `Film` préalablement hydratée dans la méthode de Controller faisant appel à cette méthode de Model .
- Un tableau de genres (`$genres = []`) contenant les ID des genres à associer au film.

Association film et genres : J'utilise la fonction `lastInsertId()` qui renvoie le dernier id inséré en base de données, dans la table. Cet `id` est celui du du film nouvellement créé.

Une boucle foreach parcours le tableau des `id` des genres. Pour chaque genre parcouru, j'utilise la fonction `addGenreToFilm()`, située dans le même fichier, et qui insère dans la table de jointure entre les tables `Film` et `Genre` les associations mentionnées ci-dessus.

Récupération de tous les films associés à un genre

extrait de code de la méthode `readAllByGenre()` de la class FilmModel :

```
// _____  
// LIRE DES FILM PAR GENRE  
// _____  
public function readAllByGenre($id_genre)  
{  
    try {  
        // PREPARATION DE LA REQUETE SQL  
        $this->request = $this->connection->prepare(  
            "SELECT  
                f.*,  
                GROUP_CONCAT(DISTINCT g.name ORDER BY g.name SEPARATOR ', ') AS genres,  
                ROUND(AVG(r.rating), 1) AS average_rating  
            FROM ppc_film f  
            INNER JOIN ppc_film_genre fg ON f.id_film = fg.id_film  
            INNER JOIN ppc_genre g ON fg.id_genre = g.id_genre  
            LEFT JOIN ppc_review r ON f.id_film = r.id_film  
            WHERE f.id_film IN (  
                SELECT id_film  
                FROM ppc_film_genre  
                WHERE id_genre = :id_genre  
            )  
            GROUP BY f.id_film;"  
        );  
        $this->request->bindValue(":id_genre", $id_genre, PDO::PARAM_INT);  
  
        // EXECUTION DE LA REQUETE SQL  
        $this->request->execute();  
  
        // FORMATAGE DU RESULTAT DE LA REQUETE  
        $films = $this->request->fetchAll();  
  
        // RETOUR DU RESULTAT  
        return $films;  
    } catch (PDOException $e) {  
        return $e->errorInfo[1];  
    }  
}
```

Cette méthode récupère en base de données tous les films étant associé à un genre donné et identifié par son *id* (`$id_genre`).

J'utilise des `INNER JOIN` pour récupérer, en plus des données de la table film, tous les genres associés au film et les noms de ces genres.

J'utilise un `LEFT JOIN` pour récupérer même les films qui n'ont pas encore de critiques. S'il y a des critiques, je les relie; sinon, les champs de la table `ppc_review` vaudront `NULL`.

Je calcule la moyenne des notes d'un film dans cette requête afin de recevoir un tableau de films ayant déjà une note moyenne attribuée, et donc éviter de nouvelles requêtes ultérieures pour récupérer les critiques de films.

Présentation de composants métier

Méthode d'envoi vers la vue des détails d'un film

```
// NAVIGUE VERS LA PAGE DETAILS FILM
// _____
public function details()
{
    // RECUPERE ID FILM DEPUIS URL
    $id_film = isset($_GET["id_film"]) ? $_GET["id_film"] : null;
    if (!$id_film) {
        $message = "Erreur système: Contactez l'administrateur du système";
        header("Location: index.php?controller=Film&action=home&msgKO=" . urlencode($message));
        exit;
    }

    // RECUPERE DONNEES FILM
    $film = $this->getFilmDetails($id_film);

    // ENVOI DONNEES FILM ET SCRIPTS JS A LA VUE
    $data = [
        "film" => $film,
        "scripts" => [
            "type='module' src='js/reviews.js'",
            "type='module' src='js/delete.js'",
            "type='module' src='js/addToFilmModal.js'",
            "type='module' src='js/removeFromFilm.js'"
        ]
    ];
    // NAVIGATION VERS PAGE
    $this->render("film/filmDetails", $data);
}
```

Explication:

Dans une des vues du MVC qui affiche des films, l'utilisateur clique sur l'affiche d'un film. Cette affiche est englobée dans un lien avec la balise `<a>` ; son attribut `href` contient le `id` du film à afficher.

Ainsi, coté back-end, le `FilmController` récupère ce lien avec la variable superglobale `$_GET`.

Si aucun id de film n'est récupéré, c'est une erreur inattendue. J'exécute alors une redirection avec la fonction PHP `header()` en ajoutant dans l'URL une variable contenant le message d'erreur pour l'utilisateur.

Si un `id` de film a été récupéré, j'appelle la méthode `getFilmDetails()`, située juste en dessous dans le même contrôleur, et qui trouve un film par son `id`.

Ensuite, je place le film (stocké dans une variable `$film`) dans une variable `$data`.

Dans cette variable, je place aussi les scripts JS à charger dans la page pour gérer son comportement dynamique.

Enfin, j'appelle la méthode `render()`, provenant de la class Controller principale, et qui permet de naviguer vers les vues du MVC. Son 1^{er} argument est la vue sous la forme "nom dossier/nom fichier" ; le 2^{ème} argument (la variable `$data` de type tableau) est extraite grâce à la fonction native de PHP `extract()`.

Le code de la méthode `render()`:

```
<?php

namespace App\Controllers;

// _____
// CLASSE CONTROLEUR DE BASE
// Classe mère dont tous les controllers vont hériter
// _____
abstract class Controller
{
    // _____
    // LE RENDU VERS LES VUES
    // _____
    public function render($view, $data = [])
    {
        extract($data); // Les clés du tableau deviennent des noms de variables.

        require_once "../app/Includes/header.php";
        require_once "../app/views/" . $view . ".php";
        require_once "../app/Includes/footer.php";
        //exit();
    }
}
```

Méthode de récupération des détails d'un film

```
// RETOURNE DETAILS D'UN FILM POUR UN FILM DONNÉ
// _____
private function getFilmDetails($id_film)
{
    if (!$id_film) {
        // AUCUN FILM FOURNI : REDIRECTION AVEC MESSAGE D'ERREUR
        $message = "Erreur inattendue";
        header("Location: index.php?controller=Film&action=home&msgKO=" . urlencode($message));
        exit;
    }

    $film = [];

    // RECUPERE LE FILM
    $filmModel = new FilmModel();
    $details = $filmModel->readByID($id_film);
    if (!$details) {
        // RETOUR SI FILM INEXISTANT
        return null;
    }
    $film['details'] = $details;

    // CONVERSION DES MINUTES EN HEURES/MINUTES
    $film['details']->duration = $this->convertMinutesToHours($film['details']->duration);

    // RECUPERE LE(S) GENRE(S)
    $genreModel = new GenreModel();
    $genres = $genreModel->getAllByFilmId($id_film);
    $film["genres"] = $genres;
```

```

// RECUPERE LES ACTEURS DU FILM
$actorModel = new ActorModel();
$actors = $actorModel->getAllByFilmId($id_film);
$film['actors'] = $actors;

// RECUPERE LES REALISATEURS DU FILM
$directorModel = new DirectorModel();
$directors = $directorModel->getAllByFilmId($id_film);
$film['directors'] = $directors;

// RECUPERE LES CRITIQUES DU FILM
$reviewModel = new ReviewModel();
$reviews = $reviewModel->readAllByFilmId($id_film);
$film['reviews'] = $reviews;

// CALCUL NOTE MOYENNE DES CRITIQUES
$totalRating = 0;
$nbReviews = count($reviews);
if ($nbReviews > 0) {
    foreach ($reviews as $review) {
        $totalRating += $review->rating;
    }
    $average_rating = $totalRating / $nbReviews;
    $film["average_rating"] = round($average_rating, 1);
}

return $film;
}

```

Ici, si le *id* du film est *null*, c'est imprévu et anormal. J'effectue alors une redirection grâce à la fonction *header()* avec un message d'erreur utilisateur passé dans l'URL.

Données de la vue : Je déclare une variable *\$film*. C'est un tableau dans lequel on va pousser tous les éléments que la vue devra exploiter :

- Les données générales sur le film (les données de la table film)
- Les genres associés au film
- Les acteurs
- Les réalisateurs
- Les critiques

Note moyenne : Je calcule la moyenne des notes attribuées au film en incrémentant le nombre total des notes (*\$totalRating*) dans une boucle *foreach()* qui parcours les critiques, mais seulement s'il y a des critiques pour le film (*\$nbReviews*).

Retour : Enfin, je retourne la variable *\$film* qui est désormais un tableau à 2 dimensions contenant:

- Dans la dimension 1: Le film
- Dans la dimension 2: Les tableaux d'objets genres, acteurs, réalisateurs, critiques et la moyenne des notes.

Méthode de mise à jour des données d'un acteur

```
// MODIFIER UN ACTEUR
// 
public function update()
{
    // Vérification de la méthode de requête
    if ($_SERVER["REQUEST_METHOD"] != "POST") {
        echo json_encode([
            'success' => false,
            'message' => "Erreur : cette page doit être appelée via une requête POST"
        ]);
        exit();
    }

    // Récupération des données du formulaire
    $id_actor = $_POST['id_actor'] ?? null;
    $firstname = $_POST['firstname'] ?? null;
    $lastname = $_POST['lastname'] ?? null;
    $firstname = $_POST['firstname'] ?? null;
    $birth_date = $_POST['birth_date'] ?? null;
    $death_date = $_POST['death_date'] == "" ? null : $_POST['death_date'];
    $biography = $_POST['biography'] == "" ? null : $_POST['biography'];
    $nationality = $_POST['nationality'] == "" ? null : $_POST['nationality'];

    // GESTION DE L'UPLOAD
    if ($_FILES["picture"]["name"] != "") {}

        if (!Validator::validateFiles($_FILES, ["picture"])) {
            echo json_encode([
                'success' => false,
                'message' => "Erreur lors de l'upload de l'image : le fichier est peut être trop volumineux (poids max : 5 Mo)"
            ]);
            exit();
        }

        // Destination du fichier
        $uploadDir = 'img/img_actors/'; // S'assurer que ce dossier existe et est accessible en écriture
        $uploadName = $_FILES['picture'][['name']];
        $uploadFile = $uploadDir . basename($uploadName);

        // Déplacer le fichier uploadé
        $success = move_uploaded_file($_FILES['picture'][['tmp_name']], $uploadFile);
        if (!$success) {
            echo json_encode([
                'success' => false,
                'message' => "Erreur lors du déplacement de l'image vers sa destination"
            ]);
            exit();
        }
    } else {
        $uploadName = null;
    }

    // Hydratation de l'instance de l'entité Actor avec les données du formulaire
    $actor = new Actor();
    $actor->setId_actor($id_actor);
    $actor->setFirstname($firstname);
    $actor->setLastname($lastname);
    $actor->setBirth_date($birth_date);
    $actor->setDeath_date($death_date);
    $actor->setBiography($biography);
    $actor->setNationality($nationality);
    $actor->setPicture($uploadName);

    // Appel de la méthode de modification d'acteur dans la BDD
    $actorModel = new ActorModel();
    $success = $actorModel->update($actor);

    echo json_encode([
        'success' => $success,
        'message' => $success ? "L'acteur a été mis à jour avec succès" : "Echec lors de la mise à jour de l'acteur"
    ]);
    exit();
}
```

Le formulaire de mise à jour de l'acteur attend que la fonction `update()` retourne une promesse résolue. Ce retour est encodé en JSON du fait de l'API Fetch utilisée a exécuté cette méthode en asynchrone après validation dynamique des données de formulaire.

Cette validation vérifie que les champs ne soient pas vides, que l'acteur ait entre 5 et 120 ans, que la date de décès ne soit située pas avant la date de naissance ni dans le futur.

Vérifications : Je commence par vérifier que la méthode utilisée côté vue est bien une méthode “POST”, sinon j’encode en JSON la réponse d’échec avec `json_encode()`. La réponse contient une variable `success = false` et `message` = message d’erreur.

Cette vérification effectuée, je récupère les données du formulaire dans la variable superglobale `$_POST`.

Pour les champs optionnels “`biography`”, “`nationality`” et “`death_date`”, j’utilise un ternaire pour transformer les données vides des champs en `null` (c’est ce qu’attend la base de données quand une donnée n’est pas renseignée).

Sécurité upload : Si le champ “picture” contient une donnée, j’utilise la classe `Validator` et sa méthode `validateFiles()` pour vérifier que cette donnée est bien un lien vers un fichier image.

- Si non, je retourne l’échec et son message d’erreur.
- Si oui, la variable `$uploadName`, contient le lien vers le fichier image.

```
// Méthode permettant de tester les champs. Les paramètres représentent les valeurs en FILES et le nom des champs
public static function validateFiles(array $files, array $fields): bool
{
    // Chaque champ est parcouru
    foreach ($fields as $field) {
        // on teste si les champs sont déclarés et sans erreur
        if (isset($files[$field]) && $files[$field]['error'] == 0) {

            // on vérifie la taille du fichier
            if ($files[$field]['size'] > 5242880) { // 5 Mo
                return false;
            }

            // on vérifie le type MIME du fichier
            $allowedTypes = ['image/jpeg', 'image/png', 'image/webp', 'image/gif'];
            if (!in_array($files[$field]['type'], $allowedTypes)) {
                return false;
            }

            // on vérifie l'extension du fichier
            $allowedExtensions = ['jpg', 'jpeg', 'png', 'webp', 'gif'];
            $fileExtension = strtolower(pathinfo($files[$field]['name'], PATHINFO_EXTENSION));
            if (!in_array($fileExtension, $allowedExtensions)) {
                return false;
            }

            /** LA BONNE PRATIQUE SERAIT DE RENOMMER LES FICHIERS UPLOADÉS AVEC UN NOM UNIQUE COMME
             ** AVEC LE CODE SUIVANT, MAIS CELA NE M'ARRANGEAIT PAS ICI
             // on génère un nouveau nom de fichier unique
             // self::$newFileName = md5(uniqid()) . '.' . $fileExtension;

            /** LA BONNE PRATIQUE SERAIT DE RENOMMER LES FICHIERS UPLOADÉS AVEC UN NOM UNIQUE COMME
             ** AVEC LE CODE SUIVANT, MAIS CELA NE M'ARRANGEAIT PAS ICI
             // on génère un nouveau nom de fichier unique
             // self::$newFileName = md5(uniqid()) . '.' . $fileExtension;

            // on déplace le fichier vers le dossier de destination
            // if (!move_uploaded_file($files[$field]['tmp_name'], 'images/' . self::$newFileName)) {
            //     return false;
            // }

            return true;
        }
    }
    return false;
}
```

Note : C’est un vérification importante pour la sécurité de l’application et de la base de données. En vérifiant le poids, l’extension et le type MIME du fichier transmis, je m’assure qu’un fichier contenant un script malveillant ne peut pas être inséré en base de données.

Une fois cette validation réussie, je :

- Définis la destination dans l'arborescence de l'appli où déplacer le fichier image.
- Déplace le fichier vers cette destination.

Ensuite, j'hydrate l'instance de l'entité *Actor* avec les données vérifiées du formulaire et je la stocke dans la variable *\$actor*. Je la passe à la méthode *update()* de la classe *ActorModel*. La variable *\$success* contient le résultat de l'upload et je m'en sers pour retourner succès ou échec.

Si *\$success* est :

- *true*, on renvoie *success*= true et un message de succès.
- *false*, on renvoie *success*= false et un message d'échec.

Finalement, *exit()* après le *json_encode()* assure que le reste du code de la page n'est pas exécuté en sortant immédiatement de la fonction.

Elements de sécurité de l'application

Pour garantir la sécurité de l'application, j'ai corrigé les principales failles de sécurité "naturelles", qui ne sont pas corrigées automatiquement par les navigateurs web et qui sont donc présentes sur une application web dès lors qu'il y a des formulaires, des infos stockées en session, des comptes utilisateurs avec un système d'identification par mot de passe, etc...

J'ai corrigé sur l'ensemble de l'application :

- **la faille XSS** en utilisant la fonction native de PHP *htmlspecialchars()* pour lire/manipuler chaque variable à destination de la base de données, et ce dans chaque vue. Ceci afin d'éviter que des scripts malveillants soient exécutés dans l'éventualité où ils auraient été insérés en base de données via les formulaires.
- **la faille CSRF** en :
 - envoyant dans les vues affichant les formulaires un **token** généré à chaque fois que les pages sont chargés.
 - en vérifiant dans le back-end, dans les méthodes des *Controllers* traitant les formulaires, que le token correspond à celui envoyé lors du chargement de la page du formulaire.

Cela vérifie que c'est bien la personne qui a navigué vers la page du formulaire, qui l'utilise et cela empêche un pirate d'utiliser le formulaire depuis un autre site web "miroir".

- **l'injection SQL** en utilisant les **requêtes préparées** de la class PDO. Elles séparent la requête SQL de ses valeurs : la structure de la requête est d'abord envoyée au serveur, puis les valeurs sont liées proprement et automatiquement échappées. Même si un utilisateur essaie d'injecter du SQL, ce qu'il tape sera traité comme une valeur et non comme du code SQL.
- **le détournement de session** en re-générant l'ID de la session (la variable superglobale `$_SESSION`) avec la fonction native fournie avec PHP `session_regenerate_id()` lorsqu'un utilisateur se connecte à son compte après avoir vérifié son mot de passe, mais **avant** de stocker en session les infos de l'utilisateur comme son pseudo, son id, etc...
- **Le stockage des mots de passe en clair** en utilisant les fonction natives `password_hash()` pour hacher le mot de passe avant de l'insérer en base de données et `password_verify()` pour vérifier la correspondance entre le mot de passe provenant du formulaire de connexion et son empreinte hachée enregistrée en base de donnée.
- **L'upload de fichier dans les formulaires**, qui est en réalité une faille car un pirate pourrait uploader dans l'arborescence de l'application un script malveillant. Pour éviter cela, je vérifie **l'extension** du fichier à uploader : Dans mon application, je veux seulement uploader des images (et l'extension est facilement falsifiable) pour les colonnes *picture* de certaines tables. Je vérifie aussi que **le poids** du fichier n'exède pas 5Mo, et **son type MIME** pour détecter le contenu réel du fichier, même si l'extension est trompeuse.

Grâce au déploiement sur le serveur distant fourni par le CEFii, l' application bénéficie également de la protection apportée par le **protocole HTTPS**. C'est une version sécurisée du protocole HTTP qui utilise le chiffrement SSL/TLS pour sécuriser les échanges entre le navigateur de l'utilisateur et le serveur qui héberge le site.

Cela permet :

1. Le chiffrement des données
 - Les données échangées (login, mot de passe, messages, etc.) sont cryptées.
 - Cela empêche un pirate d'intercepter les données sensibles (par exemple, via un Wi-Fi public).
2. L'authentification du serveur
 - Le certificat SSL prouve que le site est bien le bon.
 - Cela protège contre les attaques de type "site miroir" ou "phishing".
3. De garantir l'intégrité des données
 - Les données ne peuvent pas être modifiées en cours de route sans que cela ne soit détecté.
 - Cela évite l'insertion de contenu malveillant (ex : pub, script) dans une page.

Jeu d'essai de tests fonctionnels

- Test frond-end :

N° de test	Cas de test	Etapes à suivre	Données d'entrée	Donnée attendue	Donnée obtenue	Commentaires
1.0	Ajouter un acteur à un film	1. Aller sur l'application MovieLovers				utilisation de l'URL du site
		2. Se connecter en tant qu'administrateur du site				
		3. aller à la partie "casting" sur la page des détails d'un film				
		4. Cliquer sur le bouton "ajouter acteur" pour ouvrir la fenêtre modale				
		5. Saisir le nom dans la barre de recherche	"john"	Les résultats de la recherche affichent de façon asynchrone les acteurs correspondant au prénom "john"	L'affichage est bien présent dans la modale	

- Test back-end :

N° de test	Cas de test	Etapes à suivre	Données d'entrée	Donnée attendue	Donnée obtenue	Commentaires
1.0	Ajouter une critique à un film	1. L'utilisateur rentre des données de le formulaire pour publier une critique	[{"review": "Bon film", "rating": 4}]	Une ligne doit être créée en base de données avec la note d'une valeur de 5 (INT) et la critique "Bon film" dans les colonnes rating et content.	La ligne à bien été créée en base de donnée dans la bonne table et avec les bonnes valeurs dans chaque colonne	

Déploiement

Dans le cadre de ce projet personnel, l'étape finale est le déploiement de l'application vers le serveur fourni par le CEFii, afin que l'application soit accessible depuis n'importe quel ordinateur, smartphone ou tablette.

Application : Le déploiement de mon application est réalisé en copiant des fichiers vers ce serveur distant, à l'aide du client FTP **FileZilla**. Ce transfert permet de copier l'ensemble de l'application (fichiers PHP, ressources, fichiers CSS/JS, images, etc.) depuis mon environnement local vers l'hébergement distant.

Base de données : Elle a été exportée depuis phpMyAdmin en local au format SQL, puis importée via phpMyAdmin du serveur distant. Cette opération a nécessité une adaptation des paramètres de connexion à la base de données dans le fichier de connexion à la base de données (DbConnect.php) pour correspondre aux identifiants du serveur distant.

Plan de déploiement : Il a été mis en œuvre individuellement, avec une phase de vérification post-transfert pour tester le bon fonctionnement de l'application en ligne. Cette démarche s'inscrit dans une méthodologie simple et manuelle, adaptée à un environnement de formation, sans outil d'automatisation (CI/CD). La validation finale s'est faite par des tests fonctionnels directs sur l'application en ligne, pour chaque fonctionnalité.

Veille

- **Sur les vulnérabilités de sécurité :**

La réécriture d'URL n'est pas corrigée dans mon application.

C'est une technique qui permet de transformer des URL techniques en URL plus lisibles et plus propres. C'est souvent fait avec .htaccess (Apache) ou via un routeur PHP.

Le fait de ne pas faire de réécriture d'URL n'est pas une faille en soi : Cela permet de masquer des paramètres critiques dans l'URL comme par exemple les noms de fichier ou les identifiants uniques. Cela contribue donc à sécuriser un peu plus une application.

- **Exemple d'une situation de travail ayant donné lieu à une recherche :**

Dans une vue, j'ai eu besoin de formatter des dates, reçues sous forme de chaînes de caractères, avec la classe DateTime pour pouvoir l'afficher au format français et calculer un âge à partir de ces dates (date de naissance et éventuellement décès). Ne connaissant pas toutes les subtilités du formattage des dates, cela m'a amené à faire une recherche.

Dans un souci de rapidité, j'ai utilisé ChatGPT pour répondre à ce problème, en lui expliquant bien la situation et en lui demandant d'expliquer en détails son raisonnement.

Références

- **Bibliographie**

J'ai utilisé comme référence pour les éléments de sécurité de l'application, les documents de cours vues sur les failles de sécurité dans l'année et disponibles sur la plateforme d'e-learning du CEFii *VirtualSchool*, pour laquelle nous gardons les droits d'accès à l'issue de la formation.

Pour définir le contexte du projet je me suis basé sur le cours de gestion de projet du CEFii sur *virtualSchool*.

- **Webographie**

- MDN
- W3schools
- StackOverflow
- ChatGpt
- Google

Conclusion

Bilan technique

Ce projet personnel m'a permis de mobiliser de nombreuses compétences du référentiel du titre professionnel "Développeur Web et Web Mobile", aussi bien côté frontend que backend.

J'ai conçu une application web complète, inspirée de plateformes comme SensCritique ou AlloCiné, et permettant de consulter des films, des critiques de film, des fiches acteurs/réalisateur, et d'obtenir des suggestions personnalisées.

D'un point de vue technique, j'ai mis en œuvre :

- o Une architecture MVC en PHP pour garantir une bonne séparation des responsabilités
- o Une base de données relationnelle structurée avec des jointures complexes entre films, acteurs, réalisateurs et utilisateurs
- o Des interfaces responsives grâce à Bootstrap et du CSS personnalisé
- o De l'interactivité via JavaScript, notamment pour les notations et filtres dynamiques
- o Des scripts automatisés pour insérer des données réalistes en masse (films, critiques, etc.).

J'ai aussi porté une attention particulière à la sécurité, avec la validation des entrées utilisateur, l'échappement des données affichées, et la prise en compte du protocole HTTPS pour protéger les échanges.

Bilan humain

Sur le plan humain, ce projet m'a permis de :

- o Développer mon autonomie dans la gestion d'un projet complet, de la planification à la réalisation
- o D'approfondir ma capacité à organiser le code et à documenter mes choix techniques
- o Me confronter aux exigences de qualité logicielle, de rigueur et de sécurité

Ce projet m'a aussi permis de consolider mes acquis tout en explorant de nouvelles notions par moi-même (veille technique sur les failles, pratiques de sécurisation, organisation des scripts SQL...).

Le projet pourrait évoluer avec l'ajout de nouvelles fonctionnalités comme un système de recommandation avancé basé sur les goûts utilisateurs, ou encore un espace communautaire avec des profils publics. Ces évolutions me permettraient d'approfondir encore mes compétences techniques et de tendre vers une application plus complète.

Annexes

Le composant métier lié aux films : la classe FilmController :

```
<?php

namespace App\Controllers;

use App\Controllers\Controller as Controller;
use App\Core\CSRFTokenManager as CSRFTokenManager;
use App\Core\Validator as Validator;
use App\Entities\Film as Film;
use App\Entities\Film_Genre as Film_Genre;
use App\Models\FilmModel as FilmModel;
use App\Models\GenreModel as GenreModel;
use App\Models\ActorModel as ActorModel;
use App\Models\DirectorModel as DirectorModel;
use App\Models\ReviewModel as ReviewModel;

class FilmController extends Controller
{
    // NAVIGUE VERS ACCEUIL FILMS
    // _____
    public function home()
    {

        $genres = [
            "Science-fiction" => 7,
            "Action" => 1,
            "Drame" => 5,
            "Horreur" => 6,
            "Thriller" => 9,
            "Crime" => 19
        ];

        $filmsByGenres = [];
        foreach ($genres as $name => $id) {

            $films = $this->getFilmsByGenre($id);      // renvoie un tableau de type D2 (cf exemple)
            $filmsByGenres[$name] = $films;             // ajoute le D2 au D1
        }

        // SCRIPTS JS
        $data = [
            "filmsByGenres" => $filmsByGenres,
            "scripts" => [
                "type='module' src='js/filmsByGenre.js'",
                "type='module' src='js/home.js'"
            ]
        ];

        // NAVIGATION VERS PAGE
        $this->render("film/homeFilm", $data);
    }
}
```

```

// RETOURNE LISTE DE FILMS POUR UN GENRE DONNE
// -----
private function getFilmsByGenre($id_genre)
{
    if (!$id_genre) {
        // AUCUN GENRE FOURNI : REDIRECTION AVEC MESSAGE D'ERREUR
        $message = "Erreur inattendue";
        header("Location: index.php?controller=Film&action=home&msgKO=" . urlencode($message));
        exit;
    }

    // VERIFIE QUE LE GENRE EXISTE EN BDD
    $genreModel = new GenreModel();
    $genre = $genreModel->readByID($id_genre);
    if (!$genre) {
        // ID GENRE FOURNI NE CORRESPONDANT A AUCUN GENRE DE LA BDD
        return [];
    }

    // GENRE OK : LECTURE DES FILMS PAR GENRE
    $filmModel = new FilmModel();
    $filmsByGenre = $filmModel->readAllByGenre($id_genre);
    if (!$filmsByGenre) {
        // AUCUN FILM POUR LE GENRE FOURNI
        return [];
    }

    // CONVERSION DES MINUTES EN HEURES/MINUTES
    foreach ($filmsByGenre as $film) {
        $film->duration = $this->convertMinutesToHours($film->duration);
    }

    // FILMS EXISTANTS : RETOUR DU RESULTAT
    return $filmsByGenre;
}

// NAVIGUE VERS LA PAGE DETAILS FILM
// -----
public function details()
{
    ...
}

// RETOURNE DETAILS D'UN FILM POUR UN FILM DONNE
// -----
private function getFilmDetails($id_film)
{
    ...
}

// NAVIGATION VERS FORMULAIRE D'AJOUT DE FILM
// -----
public function addForm()
{
    $token = CSRFTokenManager::generateCSRFToken();

    // Recupere tous les genres
    $genreModel = new GenreModel();
    $genres = $genreModel->readAll();

    $data = [
        "scripts" => [
            "type='module' src='js/filmForm.js''",
            "type='module' src='js/addGenreToFilm.js'"
        ],
        "genres" => $genres,
        "token" => $token,
        "controllerMethod" => "add"
    ];

    $this->render("film/addFilmForm", $data);
}

```

```

// NAVIGATION VERS FORMULAIRE D'AJOUT DE FILM
// _____
public function addForm()
{
    $token = CSRFTokenManager::generateCSRFToken();

    // Recupere tous les genres
    $genreModel = new GenreModel();
    $genres = $genreModel->readAll();

    $data = [
        "scripts" => [
            "type='module' src='js/filmForm.js'",
            "type='module' src='js/addGenreToFilm.js'"
        ],
        "genres" => $genres,
        "token" => $token,
        "controllerMethod" => "add"
    ];

    $this->render("film/addFilmForm", $data);
}

// AJOUTER UN FILM
// _____
public function add()
{
    // Verification de la methode de requête
    if ($_SERVER["REQUEST_METHOD"] != "POST") {
        echo json_encode([
            'success' => false,
            'message' => "Erreur : cette page doit être appelée via une requête POST"
        ]);
        exit();
    }

    // Récupération des données du formulaire
    $title = $_POST['title'] ?? null;
    $synopsis = $_POST['synopsis'] ?? null;
    $release_year = $_POST['release_year'] ?? null;
    $duration = $_POST['duration'] ?? null;
    $genres = $_POST['genres'] ?? [];
    $genresArray = [];
    foreach ($genres as $id_genre) {
        $genre = new Film_Genre();
        $genre->setId_genre($id_genre);
        $genresArray[] = $genre;
    }

    // Verification que le film ne soit pas déjà en BDD
    $filmModel = new FilmModel();
    $film = $filmModel->readByTitleAndYear($title, intval($release_year));
    if ($film) {
        echo json_encode([
            'success' => false,
            'message' => "Le film '" . $title . "' sorti en " . $release_year . " existe déjà"
        ]);
        exit();
    }

    // GESTION DE L'UPLOAD
    if ($_FILES["picture"]["name"] != "") { ...
    } else {
        $uploadName = null;
    }

    // Hydratation de l'instance de l'entité Film avec les données du formulaire
    $film = new Film();
    $film->setTitle($title);
    $film->setSynopsis($synopsis);
    $film->setRelease_year(intval($release_year));
    $film->setDuration(intval($duration));
    $film->setPicture($uploadName);

    // Appel de la méthode d'ajout de film dans la BDD
    $success = $filmModel->add($film, $genresArray);

    echo json_encode([
        'success' => $success,
        'message' => $success ? "Le film a été ajouté avec succès" : "Echec lors de l'ajout du film"
    ]);
    exit();
}

```

```

// NAVIGATION VERS FORMULAIRE DE MODIFICATION DE FILM
// -----
public function updateForm()
{
}

// MODIFIER UN FILM
// -----
public function update()
{
}

// SUPPRIMER UN FILM
// -----
public function delete()
{
    $id_film = isset($_GET["id_film"]) ? $_GET["id_film"] : "";

    // Récupération du réalisateur à supprimer
    $filmModel = new FilmModel();
    $film = $filmModel->readByID($id_film);
    if (!$film) {
        // Film non trouvé en BDD
        echo json_encode([
            "success" => false,
            "message" => "Erreur inattendue : veuillez contacter l'administrateur du système"
        ]);
        exit();
    }

    // Vérification que le film ne soit pas associé à des réalisateurs et/ou acteurs et/ou à des genres
    $actorModel = new ActorModel();
    $actors = $actorModel->getAllByFilmId($id_film);
    $directorModel = new DirectorModel();
    $directors = $directorModel->getAllByFilmId($id_film);
    $genreModel = new GenreModel();
    $genres = $genreModel->getAllByFilmId($id_film);
    if ($actors || $directors || $genres) {
        // Erreur : on demande à l'utilisateur d'aller supprimer les associations entre film et acteurs/réalisateur,
        // et entre film et genres avant de supprimer le film
        echo json_encode([
            "success" => false,
            "message" => 'Veuillez retirer les genres au film et les acteurs / réalisateurs du casting de ce film avant de le supprimer'
        ]);
        exit();
    }

    // Suppression du film en BDD
    $success = $filmModel->delete($id_film);
    echo json_encode([
        'success' => $success,
        'message' => $success ? 'Film supprimé avec succès' : 'Echec de la suppression du film'
    ]);
    exit();
}

// AJOUTER UN ACTEUR A UN FILM
// -----
public function addActorToFilm()
{
    $id_film = isset($_GET["id_film"]) ? $_GET["id_film"] : null;
    $id_actor = isset($_GET["id_actor"]) ? $_GET["id_actor"] : null;

    $filmModel = new FilmModel();
    $success = $filmModel->addActorToFilm($id_film, $id_actor);

    echo json_encode([
        'success' => $success,
        'message' => $success ? "L'acteur a été ajouté au casting du film avec succès" : "Echec lors de l'ajout de l'acteur au casting de ce film"
    ]);
    exit();
}

// AJOUTER UN REALISATEUR A UN FILM
// -----
public function addDirectorToFilm()
{
}

// RETIRER UN ACTEUR DU CASTING D'UN FILM (SUPPRIMER L'ASSOCIATION)
// -----
public function removeActorFromFilm()
{
    $id_film = isset($_GET["id_film"]) ? $_GET["id_film"] : null;
    $id_actor = isset($_GET["id_actor"]) ? $_GET["id_actor"] : null;

    $filmModel = new FilmModel();
    $success = $filmModel->removeActorFromFilm($id_film, $id_actor);

    echo json_encode([
        'success' => $success,
        'message' => $success ? "L'acteur a été retiré du film avec succès" : "Echec de la suppression de cet acteur du casting de ce film"
    ]);
    exit();
}

// RETIRER UN REALISATEUR D'UN FILM (SUPPRIMER L'ASSOCIATION)
// -----
public function removeDirectorFromFilm()
{
}

```

La classe pour gérer les Tokens CSRF :

```
<?php

namespace App\Core;

class CSRFTokenManager
{
    // Nom de la clé de session pour stocker le jeton CSRF
    const CSRF_TOKEN_KEY = 'csrf_token';
    const CSRF_TOKEN_EXPIRE = 'csrf_token_expire';

    /**
     * Génère un nouveau jeton CSRF et le stocke en session
     */
    public static function generateCSRFToken()
    {
        // Génère un nouveau jeton CSRF
        $token = bin2hex(random_bytes(32));
        // Durée d'utilisation du token : 15 minutes (900 secondes)
        $token_expiration = time() + 900;

        // Stocke le jeton et son délai d'expiration en session
        $_SESSION[self::CSRF_TOKEN_KEY] = $token;
        $_SESSION[self::CSRF_TOKEN_EXPIRE] = $token_expiration;

        // Retourne le jeton généré
        return $token;
    }

    /**
     * Vérifie si le jeton CSRF soumis correspond à celui stocké en session
     * Et si le délai d'expiration du token n'est pas dépassé
     */
    public static function validateCSRFToken($submittedToken)
    {
        // Vérifie si le jeton soumis est identique à celui stocké en session
        if (
            isset($_SESSION[self::CSRF_TOKEN_KEY])
            && hash_equals($_SESSION[self::CSRF_TOKEN_KEY], $submittedToken)
            && (time() < $_SESSION[self::CSRF_TOKEN_EXPIRE])
        ) {
            // Le jeton est valide et son délai d'utilisation n'est pas dépassé,
            // On le supprime de la session pour qu'il ne puisse être réutilisé
            unset($_SESSION[self::CSRF_TOKEN_KEY]);
            unset($_SESSION[self::CSRF_TOKEN_EXPIRE]);

            // Retourne vrai pour indiquer que la validation a réussi
            return true;
        } else {
            // Le jeton est invalide ou son délai d'utilisation est dépassé, on retourne faux
            return false;
        }
    }
}
```