

---

# 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

• <b>Introduction</b> .....	<b>page 3</b>
• <b>Contexte du projet</b> .....	<b>page 4</b>
• <b>Eléments significatifs coté Front-End</b> .....	<b>page 5</b>
• <b>Eléments significatifs coté Back-End</b> .....	<b>page 5</b>
• <b>Eléments de sécurité de l'application</b> .....	<b>page 5</b>
• <b>Jeu d'essai de tests fonctionnels</b> .....	<b>page 5</b>
• <b>Déploiement</b> .....	<b>page 5</b>
• <b>Description de la veille effectuée</b> .....	<b>page 5</b>
• <b>Références</b> .....	<b>page 5</b>
• <b>Conclusion</b> .....	<b>page 5</b>
• <b>Annexes</b> .....	<b>page 5</b>

# 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 :**

<b>Activités types</b>	<b>Compétences professionnelles</b>
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.

Elle 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 la conception et le développement et ainsi pouvoir revenir en arrière lorsque des erreurs importantes auraient été commises et d'assurer le maintien d'un code fonctionnel.

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.

# Côté front-end

Les maquettes de l'interface graphique, réalisées 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 menu, 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 section for 'Science-fiction' and another for 'Action'. A 'Suivez nous sur les réseaux sociaux' section at the bottom includes icons for Facebook, Twitter, and Instagram. The footer contains 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. Below the header, the main content area has a rounded rectangular shape. It features a section titled "Les films" with a "Science-fiction" category. Under "Science-fiction", there are two movie posters: "Blade Runner" (Ridley Scott, 1982) and "Alien" (Ridley Scott, 1979). Below this, there's an "Action" category followed by three dots indicating more content. At the bottom of the main area, there's a footer section with the text "Suivez nous sur les réseaux sociaux" and 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 Réaliseurs

## 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 :**

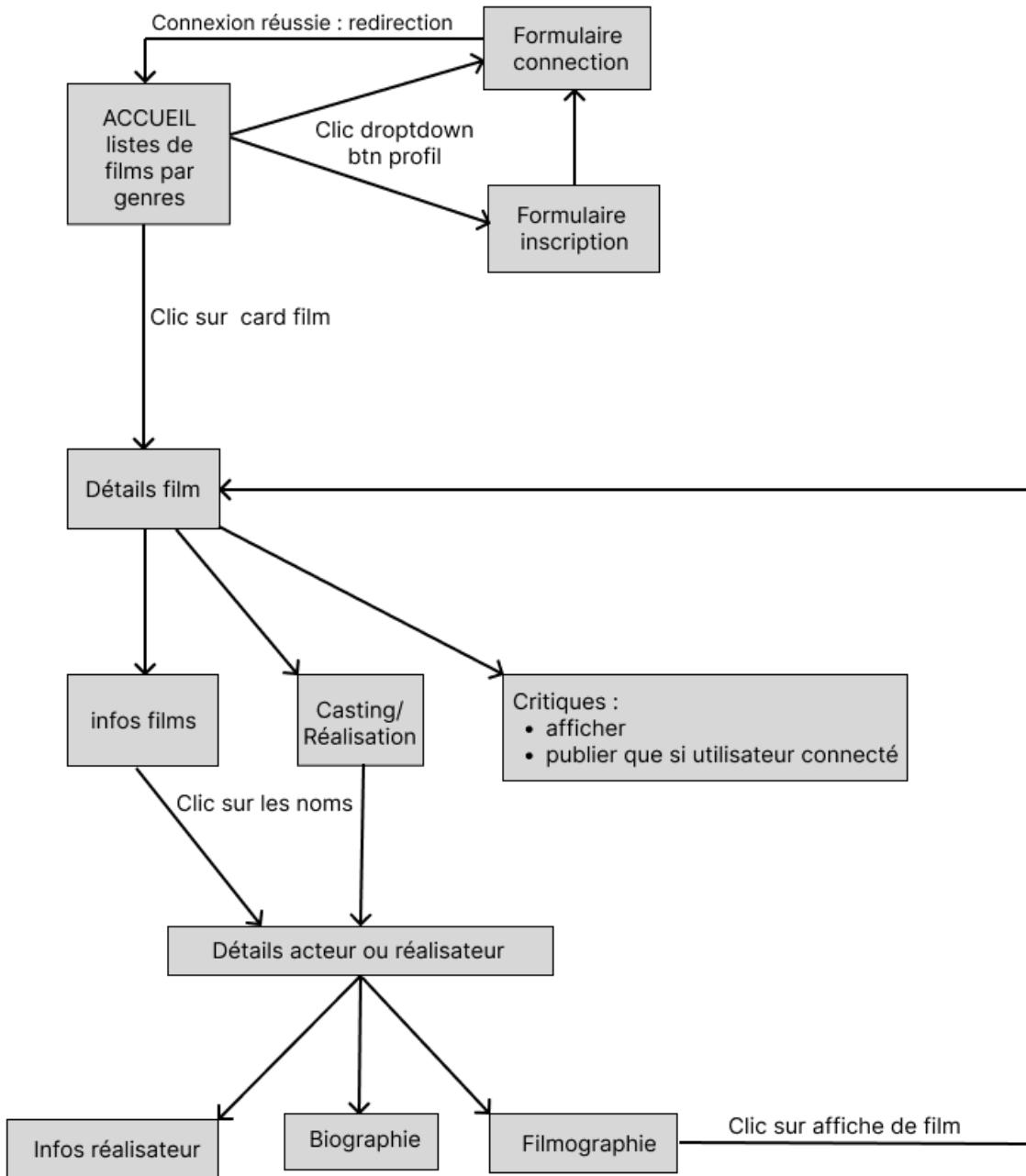
Alors que la Terre dépérît, un ancien astronaute part en mission pour trouver une nouvelle planète habitable pour l'humanité.

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 dark-themed web application interface. At the top left is the logo "MovieLovers". On the right, there's a user dropdown menu showing "superbob (admin)" and a sun icon. Below the header are navigation links: "Films", "Acteurs", and "Réalisateur". The main content area is titled "Les films".  
**Science-fiction**  
A grid of movie cards for the genre:

- 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

  
**Action**  
A horizontal row of movie cards for the genre:

- Terminator 2: Judgment Day
- Star Wars: Episode V - The Empire Strikes Back
- Die Hard
- Gladiator

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 :

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

        <?php
        // Parcours dimension 2 : les films du genre courant
        //
        ?>
        <!-- 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;">
                <i class="bi bi-chevron-left"></i>
            </button>

            <?php
            // Liste des films en scroll horizontal
            foreach ($films as $film) { ?>
                <a href="index.php?controller=Film&action=details&id_film=<?= htmlspecialchars($film->id_film, ENT_QUOTES, "UTF-8") ?>" class="film-item">
                    <div class="flex-shrink-0" style="width: 150px;">
                        <object data="img/img_films/<?= htmlspecialchars($film->picture, ENT_QUOTES, "UTF-8") ?>" class="img-fluid rounded shadow-sm mb-1">
                            
                        </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;">
                <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 left is the "MovieLovers" logo with a small icon. On the right is a user menu showing "superbob (admin)" with a dropdown arrow and a moon icon. Below the header is a navigation bar with "Films", "Acteurs", and "Réalisateur" dropdown menus. The main title "Interstellar" is centered above a row of buttons: "Accueil" (grey), "Casting" (light blue), "Critiques" (white), "Modifier" (black), and "Supprimer" (red). To the left is a movie poster for "Interstellar" showing a宇航员 in space. To the right of the poster are movie details: "Sorti en 2014 | 2 h 49 min | Drame, Science-fiction", "Réalisé par Christopher Nolan", and "Avec Matthew McConaughey, Anne Hathaway, Jessica Chastain". Below this is a rating box: "Spectateurs" (grey), "4.6" (yellow), and "11 critiques" (grey). The "Synopsis" section below the rating contains 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  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="pill-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="pill-reviews" aria-selected="false">Critiques</button>
                <!-- BOUTON MODIFIER ET BOUTON SUPPRIMER -->
                <?php if (isset($_SESSION["user"]) && $_SESSION["user"]["type"] === "admin") { ?>
                    <a class="ms-2 p-2 darkBtm btnWithBorders"
                        | href="index.php?controller=Film&action=updateForm&id_film=<?= htmlspecialchars($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
                } ?>
            </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">
                <!-- 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($film["details"]->release_year, ENT_QUOTES, "UTF-8") . " </b> | <?=
                            | htmlspecialchars($film["details"]->duration, ENT_QUOTES, "UTF-8") . " | ">
                        <?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>

```

La partie casting dans les détails d'un film avec les boutons spécifiques aux administrateurs :

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

The screenshot shows the 'Casting' section for the movie 'Interstellar'. At the top, there's a navigation bar with 'Films', 'Acteurs', and 'Réalisateurs'. Below it, the movie title 'Interstellar' is displayed. A button bar includes 'Accueil', 'Casting' (which is highlighted in blue), 'Critiques', 'Modifier', and 'Supprimer'. The main content area is titled 'Réalisateur(s)' and shows a photo and name for 'Christopher Nolan'. There's also a button labeled 'Retirer du film'.

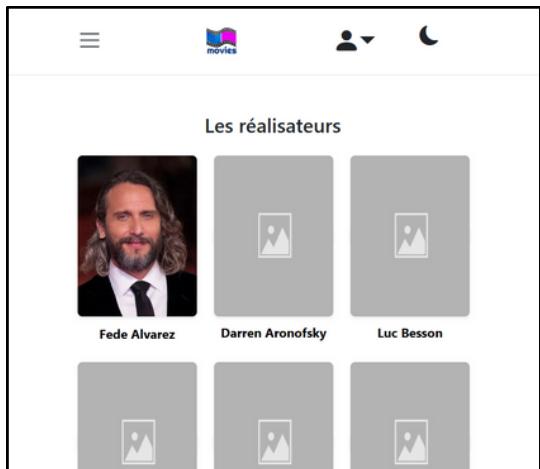
La page qui affiche la liste de tous les réalisateurs :

The screenshot shows a grid of director profiles under the heading 'Les réalisateurs'. The first profile, 'Fede Alvarez', has a photo and name. The other profiles are shown as placeholder cards with icons and names: 'Darren Aronofsky', 'Luc Besson', 'Neill Blomkamp', 'Danny Boyle', and 'Park Chan-wook'.

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 mobile de cette page avec la barre de navigation adaptée web mobile :



Pour l'adaption web mobile, j'utilise les classes col-6 col-sm-4 col-md-3 col-lg-2 (2 colonnes en mobile device, 3 colonnes en small device, 4 colonnes 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 <img> à l'intérieur de la balise <object>.

## Réalisations front-end significatives : parties dynamiques

Chargement des scripts JS : J'utilise un système générique pour charger dynamiquement les scripts nécessaires à chaque vue du MVC.

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 .

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

Pour cela, je place également du côté du formulaire HTML une structure qui permettra 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).

Dans le script JS, je définis des constantes pour chacun de ces critères et dont les valeurs sont des RegEx définissant 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.

J'empêche premièrement la soumission du formulaire avec "e.preventDefault()". Une variable "isValid" est définie avec comme valeur le boolean true pour pouvoir ensuite la passer à false dans les boucles if ( ) qui vérifient les champs du formulaire, en cas de condition non remplie.

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 la valeur 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 les chaînes de caractères à vérifier et en incrémentant la variable "checkCount" définie par défaut à 0 (car par défaut, 0 sur 5 critère de validation du mot de passe vérifié).

J'utilise la barre de progression de bootstrap :

toujours dans la fonction validateAll( ), avec une boucle switch( ) et la variable checkCount passée en argument, je modifie la couleur de la barre de progression (si checkCount est inférieure à 3 couleur=orange, si checkCount=5, couleur orange et couleur verte sinon) 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 multipliée par 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;
        }
    }

```

- Utilisation d'une fenêtre modale pour l'ajout d'acteurs ou réalisateurs à un film par l'administrateur**

Pour ajouter du dynamisme dans l'ajout de réalisateurs ou d'acteurs par l'administrateur, j'utilise une fenêtre modale masquée par défaut et qui s'ouvre lors du clic par l'admin 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 nous donne une information passée dans l'attribut "id" du lien cliqué, que l'on peut exploiter dans le fichier JS pour déterminer si l'utilisateur veut ajouter un acteur ou un réalisateur, et ainsi modifider le contenu de la modale.

Le contenu de la modale est donc modifié dynamiquement pour afficher un formulaire, contentant une barre de recherche d'acteurs ou réalisateurs, traitée en asynchrone avec la méthode AJAX, pour ensuite les ajouter au film de manière asynchrone également.

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

- le mot clé *async* avant une fonction fait en sorte que la fonction retourne une promesse.
- Le mot clé *await*, qui peut seulement être utilisé dans une fonction *async*, met en pause l'exécution de la fonction et attend que la promesse soit résolue avant de continuer l'exécution.

On va récupérer depuis le Back-End dans la variable *data* un tableau d'objets acteurs ou réalisateurs ; 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 donc dans le DOM le contenu de la modal pour y afficher une liste de personnes ( l'élément <li> prendra pour id l'id de "person").

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

Pour l'ajout au film, j'ai utilisé un à nouveau l'API Fetch, en passant dans l'URL l'entité ("Actor" ou "Director") pour appeler la bonne méthode de controller, et l'id de l'acteur ou réalisateur récupéré grâce à e.target ; l'id ayant été récupérée plus haut dans le code lors de la recherche en direct.

```
// 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(), "");

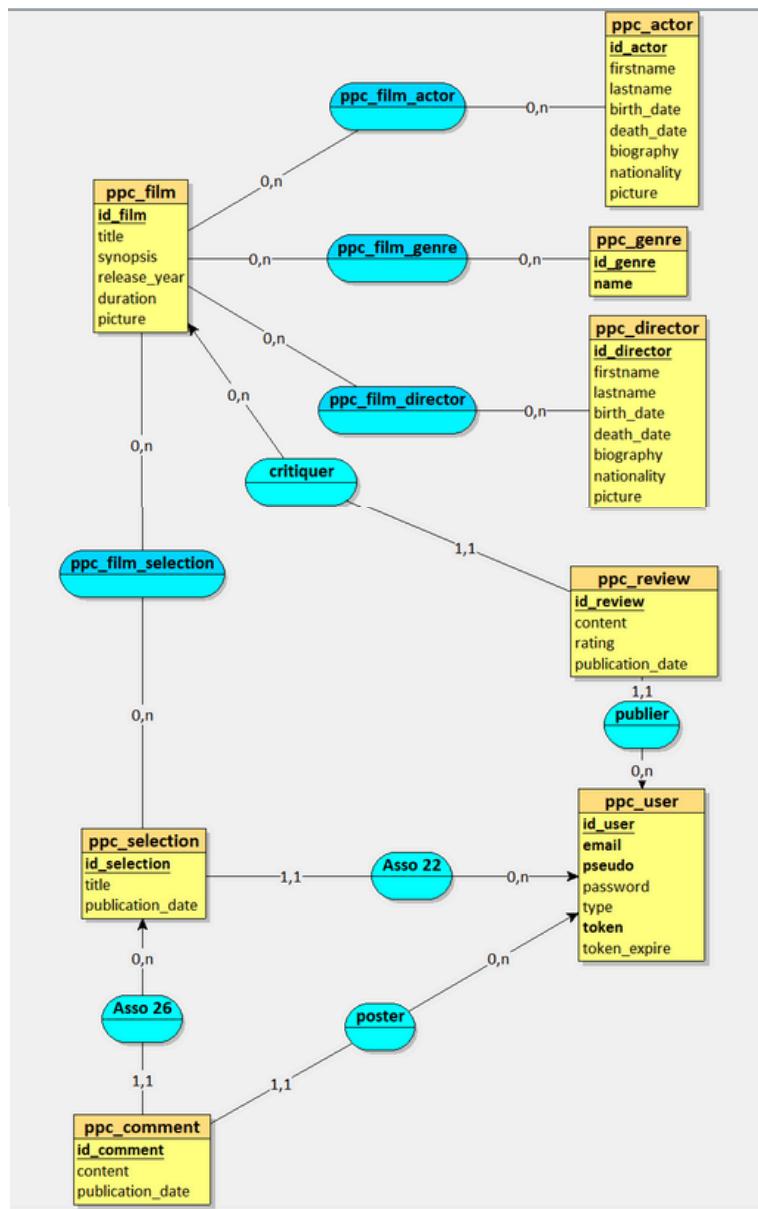
    fetch(`index.php?controller=Film&action=add${entity}ToFilm&id_film=${id}&id_${entity.toLowerCase()}=${id}`, {
      method: "GET"
    })
      .then(response => response.json())
      .then(data => {
        if (data.success === true) {
          document.querySelector(`#${entity.toLowerCase() + id}Li`).remove();
          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

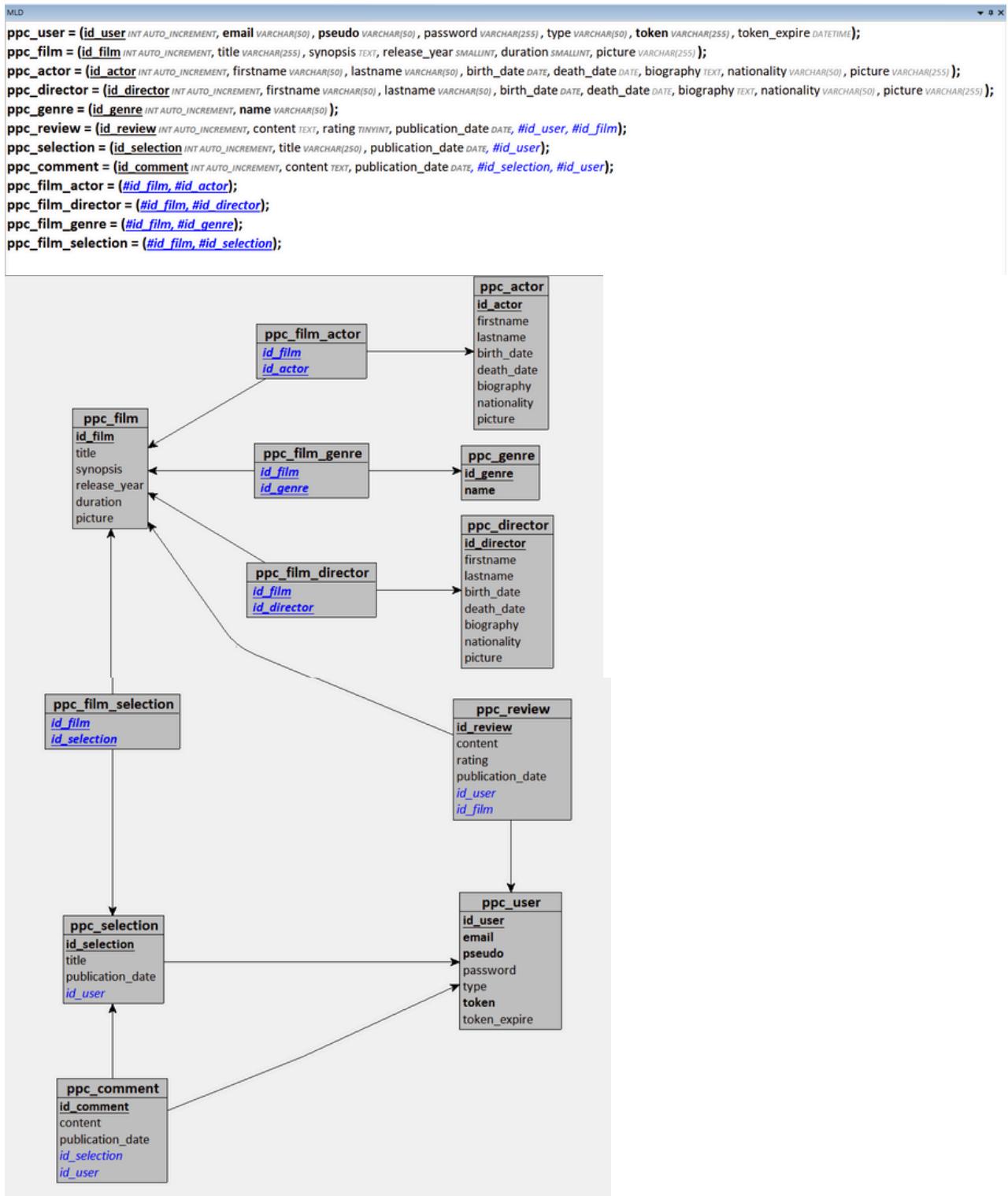
## Présentation de la base de données

Note : Pour la phase de conception de la base de données, j'ai commencé par définir les règles de gestion et un dictionnaire de données, pour lesquels il y a des captures d'écran en annexe du projet. Les MLD et MPD ont été générés automatiquement avec Looping après avoir réalisé le MCD.

- 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 pu créer la base de données avec phpMyAdmin, application Web de gestion pour les systèmes de gestion de base de données MySQL et MariaDB.  
Le système de gestion de ma base de données est MySQL.

## Présentation de composants d'accès aux données

Pour accéder / se connecter à ma base de données depuis l'application, j'ai défini une class abstraite DbConnect dans un fichier DbConnect.php (situé dans *app/core/*) dans laquelle je définit 2 attributs : \$connection et \$request ; et un constructeur qui fait appel à PDO (PHP Data Objects) : une extension définissant l'interface pour accéder à une base de données avec PHP. Elle est orientée objet, la classe s'appelant PDO.

Dans les class Models de mon MVC ou sont situées, les méthodes (les composants d'accès aux données qui contiennent les requêtes SQL), je peux utiliser les attributs \$request et \$connection de la class DbConnect pour exécuter les requêtes avec PDO.

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.

Dans les class Models du MVC, j'importe les class Entities nécessaires, afin d'utiliser les *getters* et les *setters* pour hydrater les entités selon les besoins des différents méthodes.

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* et qui a été préalablement hydraté 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.
- 

Pour pouvoir ajouter un film, et associer des genres de film au film créé, j'utilise la fonction *lastInsertId()* pour récupérer l'id du film nouvellement créé (le dernier id inséré en base de données, dans la table).

J'utilise ensuite *addGenreToFilm()* , une méthode située dans le même fichier, qui insère dans la table de jointure entre les tables *Film* et *Genre* les associations mentionnées ci-dessus avec une boucle *foreach* pour parcourir le tableau d' id des genres.

- **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'ai utilisé des INNER JOIN pour récupérer en plus des données de la table film pour récupérer tous les films associés à un genre

J'ai utilisé 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.

J'ai également choisi de calculer la moyenne des notes laissées à un film dans cette requête afin de recevoir un tableau de films dans la vue ayant déjà une note moyenne attribuée , et donc éviter de recalculer cette valeur dans la vue afin d'en aléger le code.

## **Présentation de composants métier**