

Dossier de projet

Cinetech

Alban Martinant de Préneuf

Sommaire.....	1
Introduction.....	2
Fonctionnalités.....	5
Réalisations.....	6
- Conception Base de Données.....	6
- Conception graphique.....	9
- Code.....	10
- Architecture.....	10
- API.....	12
- Connexion à la base de données.....	12
- Authentification.....	13
- Favoris.....	17
- Commentaires.....	20
- Panel admin.....	22
- Responsive.....	24
- Menu burger.....	26
Sécurité.....	31
- Injection SQL.....	31
- Faille XSS.....	31
- Mot de passe.....	32
Conclusion.....	32

Introduction

Le but de ce projet est de réaliser une bibliothèque de films et de séries avec des informations les concernant, mettre en place un système de commentaires pour que les utilisateurs puissent donner leur avis, un système de favoris et un système de recherche.

Les technologies utilisées pour le projet sont :

- Figma pour le design.
- Diagrams.net pour modéliser la base de données avec la méthode merise
- HTML/CSS pour la structure et le style.
- JAVASCRIPT pour le comportement des pages.
- PHP associé à la bibliothèque intégrée PDO pour le comportement côté back-end et la communication avec la base de données.
- Composer pour gérer les dépendances
- Autoloader pour pouvoir charger automatiquement les classes grâce à l'utilisation des namespaces.
- Altorouter pour router les requêtes.
- MariaDB pour la base de données.
- Git et GitHub pour gérer le versionning.

Arborescence du site

La page d'accueil présente deux carrousels, un pour les films tendance et un pour les séries tendance.

Une page films qui répertorie tous les films, une page série qui répertorie les séries.

Une page qui donne des informations détaillées sur un film ou une série. Les utilisateurs peuvent ajouter le film/séries à leurs favoris à partir de cette page, il peut aussi commenter l'œuvre ou répondre aux commentaires déjà présents.

Une page favoris pour lister les favoris de l'utilisateur.

Organisation

Pour ce projet, j'ai organisé mon travail avec GitHub en créant une branche par fonctionnalité. J'ai commit régulièrement les avancés du projet et j'ai effectué des pull requests quand la fonctionnalité était opérationnelle.

Dès que j'avais besoin de rechercher une information, j'ai effectué une recherche avec google, en anglais pour avoir plus de résultats de qualité étant donné que la majeur partie du contenu sur internet est dans la langue de Shakespeare. Par exemple, les deux recherches suivantes m'ont permis de trouver un moyen d'envoyer des données avec JS et de les récupérer avec PHP.

Google search results for "how to send data with fetch in javascript".

Environ 54 100 000 résultats (0.36 secondes)

The simplest way to make a request is with the `global fetch()` method. This method takes two parameters - the URL of the resource you want to retrieve and an optional configuration object. This code will send a GET request to the URL specified and then alert the response (in this case, an HTML document). 26 déc. 2022

HubSpot
https://blog.hubspot.com › website › javascript-fetch-api

What Is Fetch API in JavaScript? How to Use It (with Examples)

À propos des extraits optimisés • Commentaires

mozilla.org
https://developer.mozilla.org › en-US › Traduire cette page

Using the Fetch API - MDN Web Docs

7 juil. 2023 — The `Fetch API` provides a `JavaScript` interface for accessing and manipulating parts of the protocol, such as requests and responses.

Supplying request options · Aborting a fetch · Uploading JSON data · Headers

Use `fetch()` to POST JSON-encoded data.

JS

```
async function postJSON(data) {
  try {
    const response = await fetch("https://example.com/profile", {
      method: "POST", // or 'PUT'
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(data),
    });

    const result = await response.json();
    console.log("Success:", result);
  } catch (error) {
    console.error("Error:", error);
  }
}

const data = { username: "example" };
postJSON(data);
```



Google

how to get http request data in php



Vidéos

Livres

Images

Actualités

Maps

Flights

Finance

Environ 215 000 000 résultats (0,48 secondes)



Stack Overflow

<https://stackoverflow.com> › questions · Traduire cette page



How to get body of a POST in php?

20 janv. 2012 · 9 réponses

To access the entity **body** of a POST or PUT **request** (or any other **HTTP** method): \$entityBody = file_get_contents('php://input');

[How to send a GET request from PHP? - Stack Overflow](#) 6 juin 2009

[How to get request content \(body\) in PHP? - Stack Overflow](#) 25 août 2011

[How do I send a POST request with PHP? - Stack Overflow](#) 13 avr. 2011

[how to read an http get request from a php server](#) 8 juil. 2011

[Autres résultats sur stackoverflow.com](#)



To access the entity body of a POST or PUT request (or any other HTTP method):

712

```
$entityBody = file_get_contents('php://input');
```



Also, the `STDIN` constant is an already-open stream to `php://input`, so you can alternatively do:



```
$entityBody = stream_get_contents(STDIN);
```

Compétences du référentiel couvertes par le projet

Le projet couvre les compétences énoncées ci-dessous :

Pour l'activité 1, “Développer la partie front-end d'une application web et web mobile en intégrant les recommandations de sécurité” :

- Maquetter une application
- Réaliser une interface utilisateur web ou mobile statique et adaptable
- Développer une interface utilisateur web dynamique
- Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce

Pour l'activité 2, “Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité.”:

- Créer une base de données
- Développer les composants d'accès aux données
- Développer la partie back-end d'une application web ou web mobile
- Élaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce.

Fonctionnalités

- La page d'accueil affiche deux carrousels qui présentent les films et les séries tendances, le nombre de films et séries affiché dépend de la largeur de la page.
- Les pages films et séries affichent respectivement les films et les séries par groupe de vingt avec une pagination pour appeler les vingts suivants ou précédents.
- La page de détails des films et séries permet aux utilisateurs inscrits et connectés d'ajouter un commentaire sous la fiche détaillée. Il est aussi possible de répondre à un commentaire en particulier.
- Pendant le chargement des pages, il y a une animation qui se joue pour

signifier à l'utilisateur que le chargement est en cours.

- Il y a un menu burger quand l'écran est au format téléphone ou tablette.
- Il y a un système d'inscription/connexion qui permet aux utilisateurs de s'authentifier pour accéder à certaines fonctionnalités.
- Possibilité de se connecter à la base de données.
- L'API The Movie DataBase (TMDB) est la source des données concernant les films et séries.
- Les utilisateurs identifiés ont la possibilité d'ajouter des œuvres à leur favoris, ceux-ci sont ensuite visibles sur une page dédiée.
- Le site est responsive, il utilise les media queries en CSS et du JAVASCRIPT pour adapter le contenu à la taille de l'écran.
- Une page panel administrateur qui permet à l'administrateur de gérer les utilisateurs et modérer les commentaires.
- Une page permet aux utilisateurs de changer leur mot de passe quand ils le souhaitent.

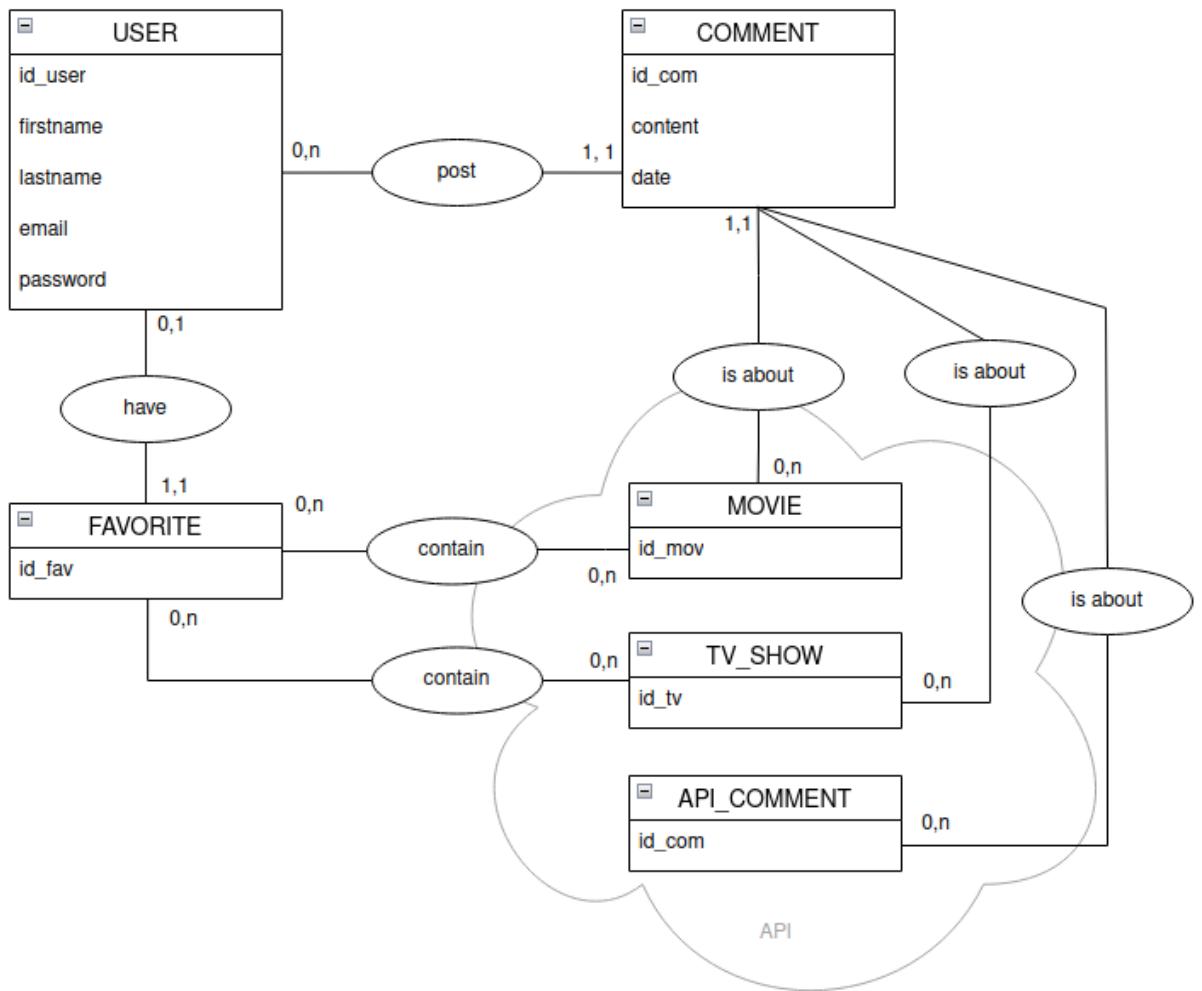
Réalisation

Conception de la base de données

Pour concevoir la base de données, j'ai utilisé la méthode merise, celle-ci se décompose en trois étapes :

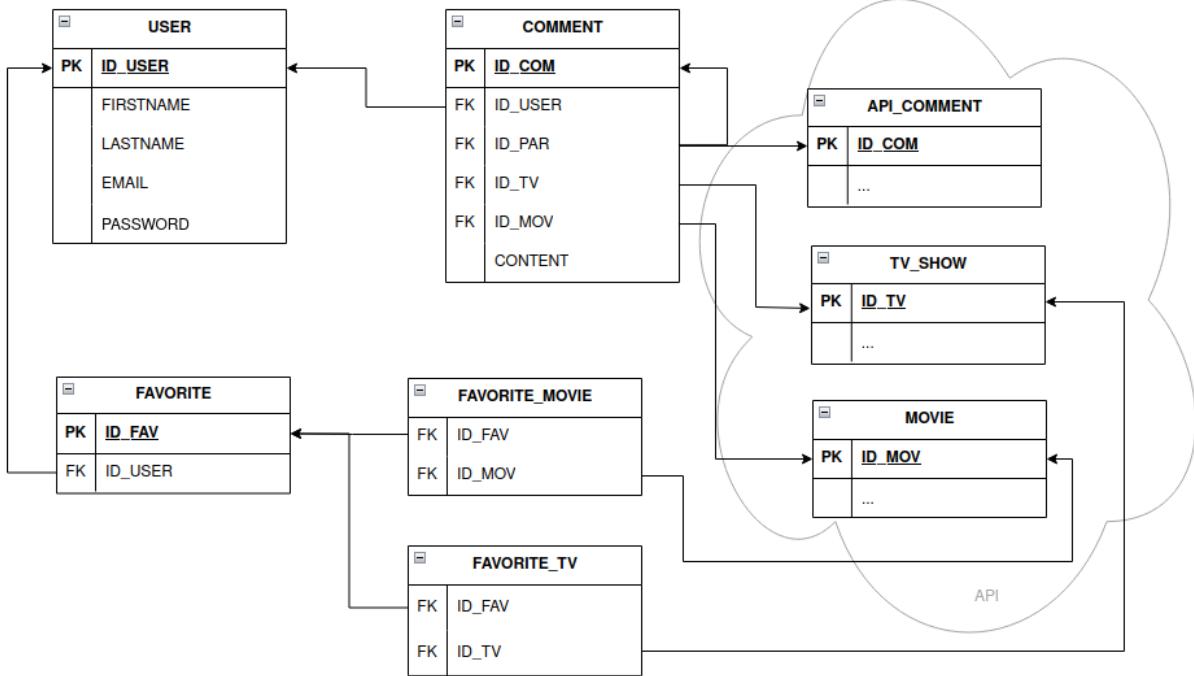
- Le Modèle Conceptuel de Base de Données (MCD) qui permet de représenter les entités et les relations entre elles. Il est réalisé à l'aide de diagrammes entité-association qui identifient les entités, les attributs et les relations entre ces entités grâce aux cardinalités. Le MCD fournit une vision abstraite de la structure globale de la base de données.

MCD



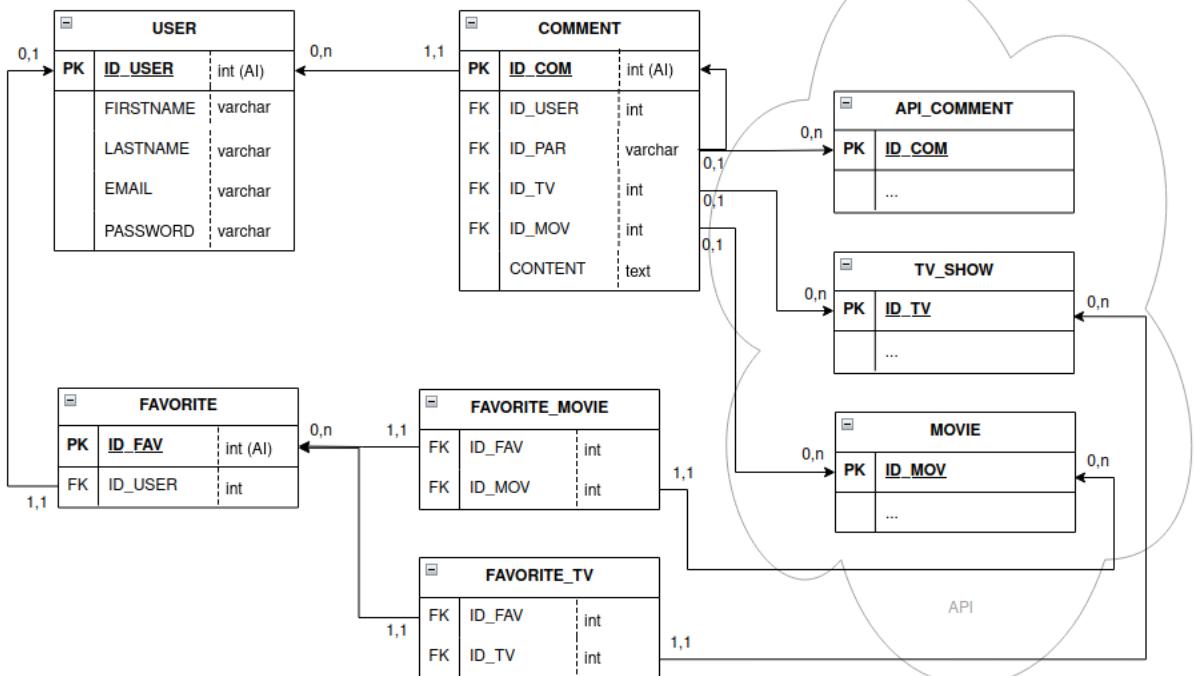
- Le Modèle Logique de Base de Données (MLD) se base sur le MCD et vise à transformer les concepts abstraits en une structure de données concrète. Cette étape permet d'ajouter les clés étrangères la ou elles sont nécessaires pour lier les tables entre elles et introduit si nécessaire des tables de liaisons en cas de relation “many to many”.

MLD



- Enfin, la dernière étape consiste à créer le Modèle Physique de Base de Données (MPD). Le MPD est une représentation concrète du MLD adaptée à un système de gestion de base de données. Il ajoute les types de données.

MPD



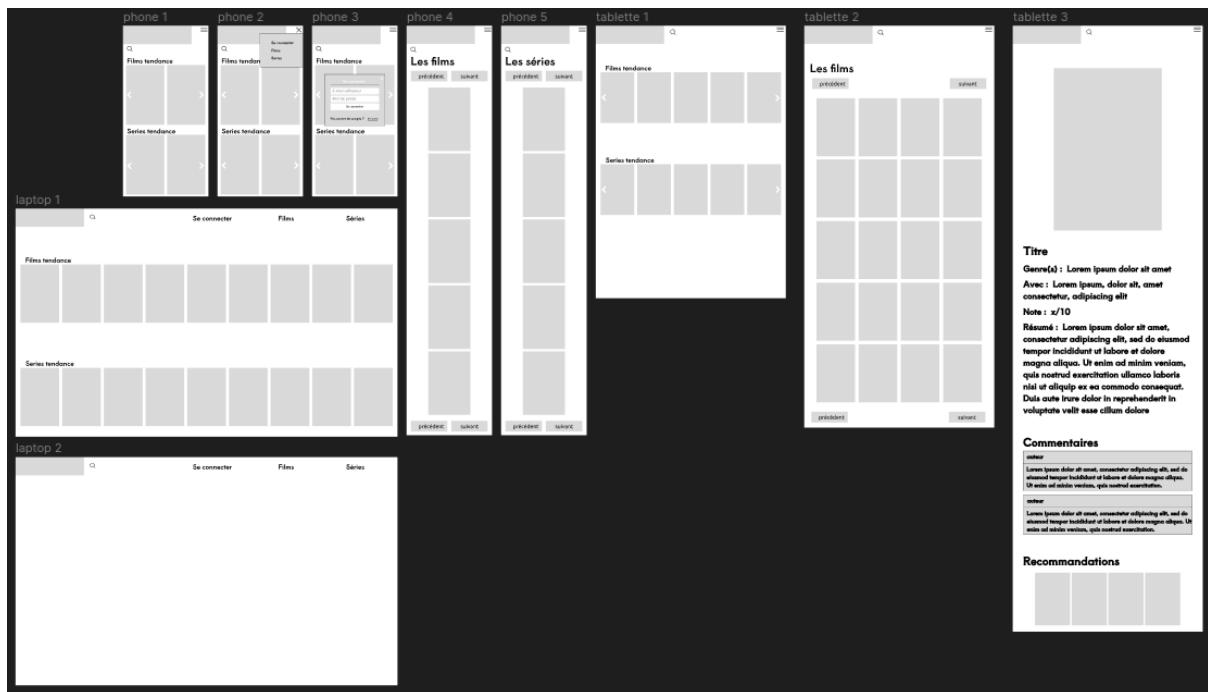
En utilisant la méthode Merise, on passe donc du MCD au MLD, puis au MPD, en

descendant progressivement vers une représentation de plus en plus détaillée et spécifique. Cette approche facilite la compréhension des besoins et permet de structurer efficacement la base de données en utilisant des concepts normalisés et des bonnes pratiques de modélisation.

Conception graphique

Pour concevoir le design du site, j'ai utilisé l'outil de design Figma.

La première étape à été de faire les wireframes, c'est-à-dire une représentation visuelle simple et basique du site. Il s'agit d'un schéma structurel qui met l'accent sur la disposition des éléments sans se concentrer sur les détails visuels ou esthétiques.



La seconde étape est de créer la maquette du site, c'est-à-dire une représentation haute fidélité du design. Elle reprend le squelette des wireframes et y ajoute les détails visuels.

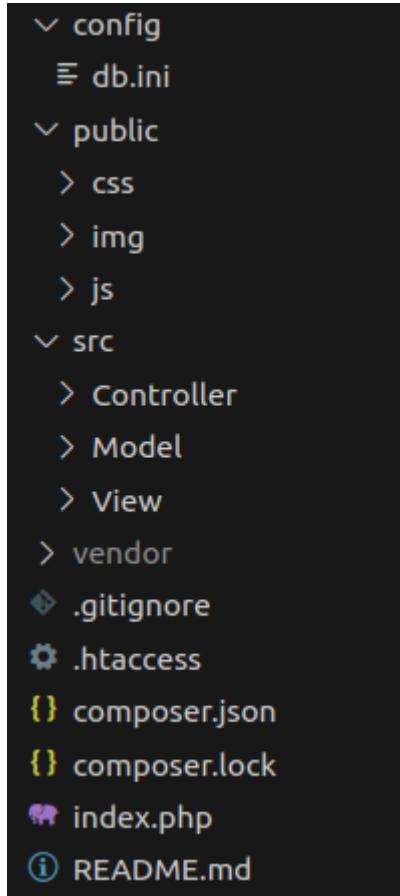


Il est ensuite possible de créer un prototype qui reprend la représentation haute-fidélité, mais y ajoute une dimension dynamique et permet donc de simuler l'expérience utilisateur.

Code

Architecture

Le site adopte l'architecture Modèle View Controller (MVC), voici l'arborescence du système de fichiers :



À la racine du projet se trouve les fichiers suivants :

- **config/** contient le fichier **db.ini** qui contient les paramètres à utiliser pour la connexion à la base de données, ce fichier est ajouté au fichier **.gitignore** pour éviter d'envoyer des informations sensibles sur GitHub.
- **public/** contient trois sous dossiers :
 - **css/** qui contient tous les fichiers css pour le style
 - **img/** qui contient toutes les images
 - **js/** qui contient tous les scripts Javascript
- **src/** contient trois sous dossiers qui contiennent les éléments du modèle MVC :
 - **Model/** contient les classes du modèle, celles-ci s'occupent de faire la liaison avec la base de données.
 - **View/** contient des fichiers PHP qui génèrent le code HTML à afficher.
 - **Controller/** contient les classes chargées du contrôle des données, elles font le lien entre la View et le Model.
- **vendor/** contient les dépendances gérées par Composer, à noter que ce dossier est ajouté à **.gitignore**.
- **.gitignore** contient la liste des fichiers et dossiers à ignorer par git.
- **.htaccess** est un fichier de configuration utilisé par les serveurs Apache, je l'utilise dans ce projet pour faire de la réécriture d'url. toutes les requêtes qui arrivent sur le serveur sont redirigées vers index.php qui s'occupe de les gérer

grâce à altorouter.

- **composer.json** est un fichier de configuration de Composer qui spécifie les dépendances du projet.
- **composer.lock** est un fichier de configuration de Composer qui sert principalement à verrouiller les versions exactes des dépendances installées.
- **index.php** est la page qui gère les requêtes grâce à altorouter.
- **README.md** est un fichier d'informations.

Requêtes à l'API

Pour interroger l'API (Application Programming Interface), j'utilise la fonction `getData()` qui se trouve dans le fichier `modules.js`. La fonction `getData` se sert de la fonction `fetch()` de Javascript pour envoyer des requêtes HTTP vers l'API.

```
import { options } from "./options.js";

export async function getData(url) {
    const response = await fetch(url, options)
    const result = await response.json();
    return result
}
```

Les options utilisées pour la connexion qui contient la clé de connexion à l'API se trouvent dans un fichier à part ce qui permet de l'ajouter à `.gitignore` et évite à la clé de se retrouver sur github.

La fonction `getData` est ensuite importé dans les fichiers JS qui en ont besoin

```
import { getData, loader } from "./modules/module.js";
```

Comme la fonction `getData` est asynchrone, elle renvoie une promesse, j'utilise donc le mot clé “`await`” pour récupérer la résolution de la promesse (à noter que je pourrais aussi utiliser la méthode `.then()`). Voici un exemple :

```
async function displayMovie() {
    const detailsDiv = document.getElementById('details_div')
    const movie = await getData("https://api.themoviedb.org/3/" + typeItemSing + "/" + idItem + "?language=fr-FR");
    const credits = await getData("https://api.themoviedb.org/3/" + typeItemSing + "/" + idItem + "/credits?language=fr-FR");
```

Connexion à la base de données

La connexion à la base de données se fait grâce à la classe `DbConnection` qui se trouve dans le `model`, sa méthode de classe `getDb()` renvoie une instance de `PDO` qui

représente la connexion à la base de données. Les paramètres utilisés pour la connexion se trouvent dans un fichier séparé, config/db.ini.

```
<?php

namespace App\Model;

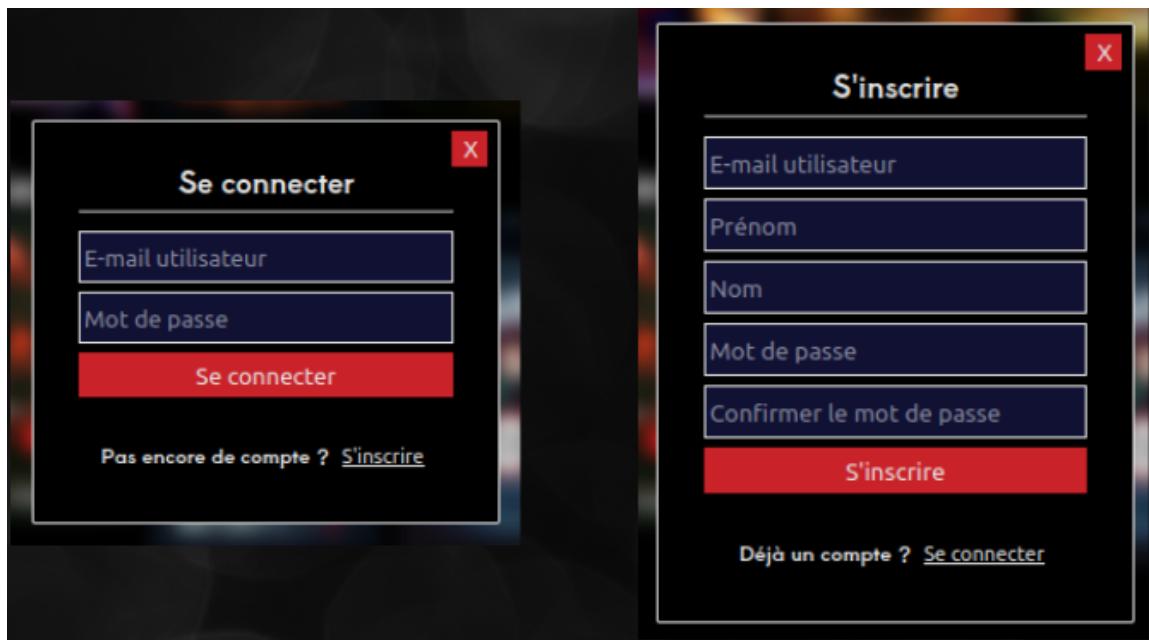
class DbConnection
{

    private static ?\PDO $_db = null;

    public static function getDb()
    {
        if (!self::$_db) {
            try {
                // get database infos from ini file in config folder
                $db_config = parse_ini_file('config' . DIRECTORY_SEPARATOR . 'db.ini');
                // define PDO dsn with retrieved data
                self::$_db = new \PDO($db_config['type'] . ':dbname=' . $db_config['name']
                    . ';host=' . $db_config['host']
                    . ';charset=' . $db_config['charset'], $db_config['user'], $db_config['password']);
                // prevent emulation of prepared requests
                self::$_db->setAttribute(\PDO::ATTR_EMULATE_PREPARES, false);
            } catch (\PDOException $e) {
                header("HTTP/1.1 403 Acces refused to the database");
                die();
            }
        }
        return self::$_db;
    }
}
```

Authentification

Les utilisateurs doivent s'authentifier pour avoir accès à certaines fonctionnalités, tout d'abord l'utilisateur doit s'inscrire, ce qui déclenche son enregistrement en base de données. En cliquant sur “se connecter” le formulaire de connexion s'affiche. La première fois, il faut cliquer sur s'inscrire pour faire apparaître le formulaire d'inscription.



Les données sont ensuite envoyées à la route /register qui instancie un nouveau controller et appelle sa méthode register() en lui passant en paramètre les données entrées dans le formulaire.

```
$router->map('POST', '/register', function () {
    $authController = new AuthController();
    $authController->register($_POST['firstname'], $_POST['lastname'], $_POST['email'], $_POST['password'],
    $_POST['password2']);
});
```

La méthode register() vérifie que tous les champs ont bien été remplis, applique la méthode htmlspecialchars() à tous les arguments ce qui convertit les caractères spéciaux en entités HTML (par exemple "<" devient "<") ce qui évite que du code soit enregistré en BDD et donc prévient la faille XSS; ensuite elle vérifie que les deux mots de passe sont identiques, que l'email est valide et que personne n'est déjà inscrit avec cet email. Si l'une de ces conditions n'est pas remplie, elle renvoie un en-tête HTTP avec le code d'erreur 400 et le message d'erreur approprié. Ce message est ensuite affiché dans le formulaire avec Javascript pour en informer l'utilisateur. Si il n'y a pas d'erreur, elle hash le password, instancie un nouveau model et appelle la méthode register du model en lui passant les paramètres.

```

namespace App\Controller;

use App\Model\UserModel;

class AuthController
{

    public function register($firstname, $lastname, $email, $password, $password2)
    {
        $args = func_get_args();
        foreach ($args as &$arg) {
            if (empty($arg)) {
                header("HTTP/1.1 400 Empty field");
                die();
            }
            $arg = htmlspecialchars($arg);
        }
        if ($password !== $password2) {
            header("HTTP/1.1 400 Passwords don't match");
            die();
        }
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            header("HTTP/1.1 400 Invalid email");
            die();
        }
        $userModel = new UserModel();
        if ($userModel->getUser($email)) {
            header("HTTP/1.1 400 Email already exists");
            die();
        }
        $password = password_hash($password, PASSWORD_DEFAULT);
        $userModel->register($firstname, $lastname, $email, $password);
    }
}

```

```

function submitSignInForm(e) {
    e.preventDefault();
    const data = new FormData(e.target);
    fetch('/cinetech/register', {
        method: 'POST',
        body: data
    })
    .then(response => {
        if (response.status === 200) {
            document.querySelector('#form_connection').remove();
            displayLogInForm();
        } else {
            console.log(response.statusText);
            const error = document.getElementById('error');
            error.innerHTML = response.statusText;
        }
    })
}

```

La méthode register() du model s'occupe d'enregistrer l'utilisateur en base de données.

```

public function register($firstname, $lastname, $email, $password)
{
    $db = DbConnection::getDb();
    $sql_request = ("INSERT INTO user (firstname, lastname, email, password)
        VALUES (:firstname, :lastname, :email, :password)"
    );
    $statement = $db->prepare($sql_request);
    $statement->execute([
        ':firstname' => $firstname,
        ':lastname' => $lastname,
        ':email' => $email,
        ':password' => $password
    ]);
    // create favorite list
    $userId = $db->lastInsertId();
    $this->createFavoriteList($userId);
}

```

Le formulaire de login, quant à lui, envoie les données à la route /login qui instancie un nouveau controller et appelle sa méthode login() en lui passant les données postées.

```

$router->map('POST', '/login', function () {
    $authController = new AuthController();
    $authController->login($_POST['email'], $_POST['password']);
});

```

La méthode login() vérifie la validité des données comme précédemment et si tout est ok, elle ajoute l'utilisateur dans la super globale \$_SESSION.

```

public function login($email, $password)
{
    $args = func_get_args();
    foreach ($args as &$arg) {
        if (empty($arg)) {
            header("HTTP/1.1 400 Empty field");
            die();
        }
        $arg = htmlspecialchars($arg);
    }
    $userModel = new UserModel();
    $user = $userModel->getUser($email);
    if ($user && password_verify($password, $user['password'])) {
        $_SESSION['user'] = $user;
        header("HTTP/1.1 200 OK");
        die();
    } else {
        header("HTTP/1.1 400 Invalid credentials");
        die();
    }
}

```

Favoris

Sur la page détaillant les films et séries, un utilisateur connecté à la possibilité d'ajouter le films ou la séries à ses favoris. Le bouton apparaît seulement si l'utilisateur est connecté.

```
<?php if (isset($_SESSION['user'])) : ?>
|   <div id="movie_btns">
|       <button id="favorite_btn" class="red small_btn"></button>
```

La fonction activateFavorite() sélectionne ensuite le bouton, si il n'existe pas, elle ne fait rien. S'il existe, il récupère la liste des favoris de l'utilisateur connecté et vérifie si le film ou la série y est déjà. Si c'est le cas le bouton sera pour le retirer des favoris sinon il sera pour l'ajouter aux favoris.

```
async function activateFavorite() {
    const favoriteBtn = document.getElementById('favorite_btn');
    if (!favoriteBtn) return;

    const response = await fetch('/cinetech/favoriteslist')
    const userFavorites = await response.json();
    if (userFavorites[typeItemPlur].includes(parseInt(idItem))) {
        favoriteBtn.append('Retirer des favoris')
        activateRemoveFavorite(favoriteBtn);
    } else {
        favoriteBtn.append('Ajouter aux favoris')
        activateAddToFavorite(favoriteBtn);
    }
}
```

Pour récupérer la liste des favoris, j'envoie une requête en GET à la route /favoritesList qui instancie un nouveau controller et appelle sa méthode getFavorites().

```
$router->map('GET', '/favoriteslist', function () {
    $userController = new UserController();
    $userController->getFavorites();
});
```

getFavorites() vérifie s'il y a un utilisateur connecté, instancie un nouveau model et appelle ses méthodes getFavoritesMovies() et getFavoritesTvs() en leur passant l'id de l'utilisateur connecté. Elle affiche ensuite le résultat sous forme de json grâce à json_encode().

```

public function getFavorites()
{
    if (isset($_SESSION['user'])) {
        $userModel = new UserModel();
        $favoritesMovies = $userModel->getFavoritesMovies($_SESSION['user']['id_user']);
        $favoritesTvs = $userModel->getFavoritesTvs($_SESSION['user']['id_user']);
        $favorites = [
            'movies' => $favoritesMovies,
            'tvs' => $favoritesTvs
        ];
        echo json_encode($favorites);
    }
}

```

Les méthodes getFavoritesMovies() et getFavoritesTvs() du modèle se connectent à la base de données et font une requête qui récupère la liste des favoris de l'utilisateur et la retourne.

```

public function getFavoritesMovies($id)
{
    $db = DbConnection::getDb();
    $sql_request = ("SELECT id_mov FROM favorite_movie
                    INNER JOIN favorite ON favorite_movie.id_fav = favorite.id_fav
                    WHERE favorite.id_user = :id"
    );
    $statement = $db->prepare($sql_request);
    $statement->execute([
        ':id' => $id
    ]);
    $favorites = $statement->fetchAll(\PDO ::FETCH_COLUMN);
    return $favorites;
}

```

La fonction activateAddToFavorite() ajoute un écouteur d'événement au bouton grâce à la fonction méthode addEventListener(), celle-ci prend en paramètre le type d'événement à écouter et la fonction callback à appeler au déclenchement de l'événement.

```

function activateAddToFavorite(favoriteBtn) {
    favoriteBtn?.addEventListener('click', () => {
        fetch('/cinetech/favorites/add' + typeItemSing + '/' + idItem)
            .then(response => {
                if (response.ok) {
                    window.location.reload();
                } else {
                    console.error(response.status);
                }
            })
    })
}

```

La fonction passée en callback envoie une requête à /cinetech/favorites/addmovie(ou addtv)/id_du_film et recharge la page.

```

$router->map('GET', '/favorites/addmovie/[i:id]', function ($id) {
    $userController = new UserController();
    $userController->addFavoriteMovie($id);
});

```

Le controller appelle ensuite addFavoriteMovie() du Model.

```

public function addFavoriteMovie($id)
{
    if (isset($_SESSION['user'])) {
        $userModel = new UserModel();
        $favId = $userModel->getFavoriteId($_SESSION['user']['id_user']);
        if ($favId) {
            $userModel->addFavoriteMovie($favId, $id);
        } else {
            header("HTTP/1.1 400 No favorite list");
            die();
        }
    }
}

```

addFavoriteMovie() enregistre l'id du film dans la liste des favoris de l'utilisateur.

```

public function addFavoriteMovie($favId, $movieId) {
    $db = DbConnection::getDb();
    $sql_request = ("INSERT INTO favorite_movie (id_fav, id_mov)
    |   VALUES (:id_fav, :id_mov)"
    );
    $statement = $db->prepare($sql_request);
    $statement->execute([
        ':id_fav' => $favId,
        ':id_mov' => $movieId
    ]);
}

```

Commentaires

L'ajout de commentaires est accessible aux utilisateurs enregistrés. Le contenu du commentaire est envoyé à la route /cinetech/movies(ou tvs)/addcomment/\${id} grâce à la fonction fetch(), je lui précise que la requête sera faite avec la méthode POST et passe le contenu dans le body de la requête.

```

function activateSendComment() {
    activateAddComment();
    const addComment = document.getElementById('add_comment');
    const commentContent = document.getElementById('comment_content');
    addComment?.addEventListener('click', async () => {
        const comment = commentContent.value;
        const response = await fetch('/cinetech/' + typeItemPlur + '/addcomment/' + idItem, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ comment })
        })
        if (response.ok) {
            window.location.reload();
        } else {
            console.error(response.status);
        }
    })
}

```

Le contenu est ensuite récupéré avec php://input est un flux en lecture seule qui permet de lire des données brutes depuis le corps de la requête.

file_get_contents() lit ce contenu brut et le renvoie sous forme de chaîne de caractères.

json_decode() est utilisée pour décoder une chaîne JSON en une structure de données PHP. Le deuxième argument true indique que nous souhaitons obtenir un tableau associatif plutôt qu'un objet PHP lors du décodage JSON.

['comment'] permet donc de récupérer ce qui se trouve à l'index 'comment' du

tableau retourné par json_decode().

```
$router->map('POST', '/movies/addcomment/[i:id]', function ($id) {
    $comment = json_decode(file_get_contents('php://input'), true)['comment'];
    $commentController = new CommentController();
    $commentController->addMovieComment($id, $comment);
});
```

Je fais ensuite appel à la méthode addMovieComment() qui vérifie si un utilisateur est connecté, convertit les caractères spéciaux en entité html avec htmlspecialchars() pour contrer la faille XSS, instancie un objet CommentModel() et appelle la méthode addMovieComment() en lui passant l'id du film concerné, le commentaire et l'id de l'utilisateur connecté.

Si aucun utilisateur n'est connecté, je fait appel à la fonction header() pour renvoyer un message d'erreur avec le code d'erreur http 400.

```
public function addMovieComment($idMovie, $comment)
{
    if (isset($_SESSION['user'])) {
        $comment = htmlspecialchars($comment);
        $userModel = new CommentModel();
        $userModel->addMovieComment($idMovie, $comment, $_SESSION['user']['id_user']);
    } else {
        header("HTTP/1.1 400 No user connected");
    }
}
```

La méthode addMovieComment() enregistre ensuite le commentaire en base de données.

```
public function addMovieComment($idMovie, $content, $idUser)
{
    $db = DbConnection::getDb();
    $sql_request = ("INSERT INTO comment (id_mov, id_user, content)
                    VALUES (:id_mov, :id_user, :content)");
    $statement = $db->prepare($sql_request);
    $statement->execute([
        ':id_mov' => $idMovie,
        ':id_user' => $idUser,
        ':content' => $content
    ]);
}
```

Panel admin

L'utilisateur ayant l'id 1 est l'administrateur du site, il est le seul à avoir accès au panel admin. Sur la page admin.php, le code suivant redirige tous les utilisateurs dont l'id est différent de 1 sur la page d'accueil du site.

```
<?php
if (!isset($_SESSION['user']) || $_SESSION['user']['id_user'] !== 1) {
    header('Location: /cinetech');
}
?>
```

À travers le panel admin, l'administrateur peut modifier et supprimer les infos des utilisateurs et supprimer les commentaires.

The screenshot shows the Cinetech administration interface. At the top, there's a navigation bar with the Cinetech logo, a search icon, and a menu icon. Below it, a large title "Administration" is displayed. The first section is titled "Utilisateurs" (Users) and contains a table with five rows of user data:

Nom	Prénom	Email	Modifier	Supprimer
admin	admin	admin@cinetech.com	Modifier	Supprimer
Victor	Bellic	vicbell@aol.qs	Modifier	Supprimer
James	deamon	demo@demo.fr	Modifier	Supprimer
John	Doe	johndoe@wanadoo.fr	Modifier	Supprimer

The second section is titled "Commentaires" (Comments) and contains a table with ten rows of comment data:

Utilisateur	Commentaire	Supprimer
admin admin	test	Supprimer
Utilisateur supprimé	super film	Supprimer
Victor Bellic	I love this movie !	Supprimer
Victor Bellic	Mario is the best plumber !	Supprimer
Victor Bellic	The first one was better...	Supprimer
John Doe	C'est pas mal	Supprimer
John Doe	I'm agree with you bro	Supprimer
admin admin	great job	Supprimer
James deamon	Mon film préféré	Supprimer

Il y a des écouteurs d'événements sur les boutons. Le clique sur le bouton "Modifier" envoie une requête en POST avec la fonction fetch en envoyant dans le corps de la requête les données du formulaire.

```

button.addEventListener('click', async (e) => {
    if (confirm('Voulez-vous vraiment modifier cet utilisateur ?')) {
        const inputFirstname = e.target.parentNode.parentNode.children[0].children[0].value;
        const inputLastname = e.target.parentNode.parentNode.children[1].children[0].value;
        const inputEmail = e.target.parentNode.parentNode.children[2].children[0].value;

        if (user.firstname !== inputFirstname
            || user.lastname !== inputLastname
            || user.email !== inputEmail) {
            const res = await fetch('/cinetech/admin/users/modify/' + user.id_user, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({
                    firstname: inputFirstname,
                    lastname: inputLastname,
                    email: inputEmail
                })
            });
            if (res.ok) {
                e.target.parentNode.parentNode.children[0].children[0].value = inputFirstname;
                e.target.parentNode.parentNode.children[1].children[0].value = inputLastname;
                e.target.parentNode.parentNode.children[2].children[0].value = inputEmail;
            }
        }
    }
});

```

Le routeur récupère les données, instancie un AdminController() et appelle la méthode modifyUser() de celui-ci.

```

$router->map('POST', '/admin/users/modify/[i:id]', function ($id) {
    $user = json_decode(file_get_contents('php://input'), true);
    $adminController = new AdminController();
    $adminController->modifyUser($id, $user);
});

```

À noter que pour des raisons de sécurité, la classe AdminController() n'est pas instanciable si l'utilisateur courant n'est pas l'administrateur.

```

class AdminController
{
    public function __construct()
    {
        if (!isset($_SESSION['user'])) {
            header("HTTP/1.1 400 No user connected");
            die();
        }

        if ($_SESSION['user']['id_user'] !== 1) {
            header("HTTP/1.1 403 Forbidden");
            die();
        }
    }
}

```

modifyUser() instancie ensuite un UserModel() et appelle sa méthode modifyUser().

```
public function modifyUser($id, $user)
{
    $userModel = new UserModel();
    $userModel->modifyUser($id, $user);
}
```

La méthode modifyUser() modifie l'entrée de la base de données correspondante avec une requête UPDATE.

```
public function modifyUser($id, $user)
{
    $db = DbConnection::getDb();
    $sql_request = ("UPDATE user
                    SET firstname = :firstname, lastname = :lastname, email = :email
                    WHERE id_user = :id");
    $statement = $db->prepare($sql_request);
    $statement->execute([
        ':id' => $id,
        ':firstname' => $user['firstname'],
        ':lastname' => $user['lastname'],
        ':email' => $user['email']
    ]);
}
```

Les boutons supprimer fonctionnent à peu près de la même manière, une requête est envoyée à la bonne route qui appelle un controller qui lui-même appelle le modèle qui envoie la requête pour supprimer.

Responsive

Le responsive du site se fait grâce aux medias queries de CSS ainsi que JAVASCRIPT.

Sur la home page, la fonction setSlideToDisplay() adapte le nombre d'image affiché en fonction de la taille de la fenêtre.

```
function setSlideToDisplay(imageSize) {
    windowHeight = window.innerWidth;
    if (windowWidth > 9 * imageSize + 10) {
        slideToDisplay = 10;
    } else if (windowWidth > 8 * imageSize + 10) {
        slideToDisplay = 9;
    } else if (windowWidth > 7 * imageSize + 10) {
        slideToDisplay = 8;
    } else if (windowWidth > 6 * imageSize + 10) {
        slideToDisplay = 7;
    } else if (windowWidth > 5 * imageSize + 10) {
        slideToDisplay = 6;
    } else if (windowWidth > 4 * imageSize + 10) {
        slideToDisplay = 5;
    } else if (windowWidth > 3 * imageSize + 10) {
        slideToDisplay = 4;
    } else if (windowWidth > 2 * imageSize + 10) {
        slideToDisplay = 3;
    } else {
        slideToDisplay = 2;
    }
}
```

Les medias queries définissent des règles à appliquer pour différents formats d'écran.

Quand la largeur fait moins de 700px, les medias queries n'entre pas en compte

```
/* media queries */

@media screen and (max-width: 1200px) and (min-width: 700px) {
>   div#search_div { ...
|   }
| }

@media screen and (min-width: 1200px) {

  div.movie_details,
>  div.tv_details { ...
| }

>  div#menu { ...
| }

>  div#details_div { ...
| }

>  div#details_div>div { ...
| }

>  ul#list-menu { ...
| }

>  div#burger_div { ...
| }
}

@media screen and (min-width: 1500px) {
>   div#search_div { ...
|   }
| }
```

Menu Burger

L'objectif principal d'un menu burger est de fournir une expérience utilisateur optimale sur les appareils mobiles ou les écrans plus petits où l'espace est limité. Au lieu d'afficher tous les éléments du menu de navigation encombré sur l'écran, le menu burger permet de les regrouper et de les cacher initialement.

```

<div id="menu">
  <nav>
    <a id="link" href="">
      <div id="burger_div">
        <span id="burger"></span>
      </div>
    </a>
    <ul id="list-menu">
      <?php if (isset($_SESSION['user'])) : ?>
        <li><a href="/cinetech/favorites" id="logout">Mes favoris</a></li>
        <li><a href="/cinetech/logout" id="logout">Se déconnecter</a></li>
      <?php else : ?>
        <li><a href="#" id="auth">Se connecter</a></li>
      <?php endif; ?>
        <li><a href="/cinetech/movies" id="movies">Films</a></li>
        <li><a href="/cinetech/tvs" id="tvs">Series</a></li>
      </ul>
    </nav>
  </div>

```

Le clique sur le menu burger ajoute la classe “open” à la liste grâce à JS, le menu est ensuite affiché et les 3 petits traits horizontaux sont transformés en croix grâce aux règles CSS qui cible la classe “open”.

```

// listen for click on burger
link.addEventListener('click', (e) => {
  e.preventDefault();
  burger.classList.toggle('open');
  ul.classList.toggle('open');
  ul.focus();
  ul.addEventListener('blur', () => {
    burger.classList.remove('open');
    ul.classList.remove('open');
  });
})

```

```
#burger.open {
    background: transparent;
}

#burger.open::before {
    transform: rotate(45deg);
    top: 0;
}

#burger.open::after {
    transform: rotate(-45deg);
    top: 0;
}

#burger_div {
    float: right;
    height: 30px;
}

ul#list-menu {
    position: absolute;
    padding: 5%;
    right: -10%;
    top: 35px;
    width: 0px;
    background-color: □black;
    color: ■#e9e9e9;
    transition: all 0.5s ease-in-out;
    overflow: hidden;
    white-space: nowrap;
}

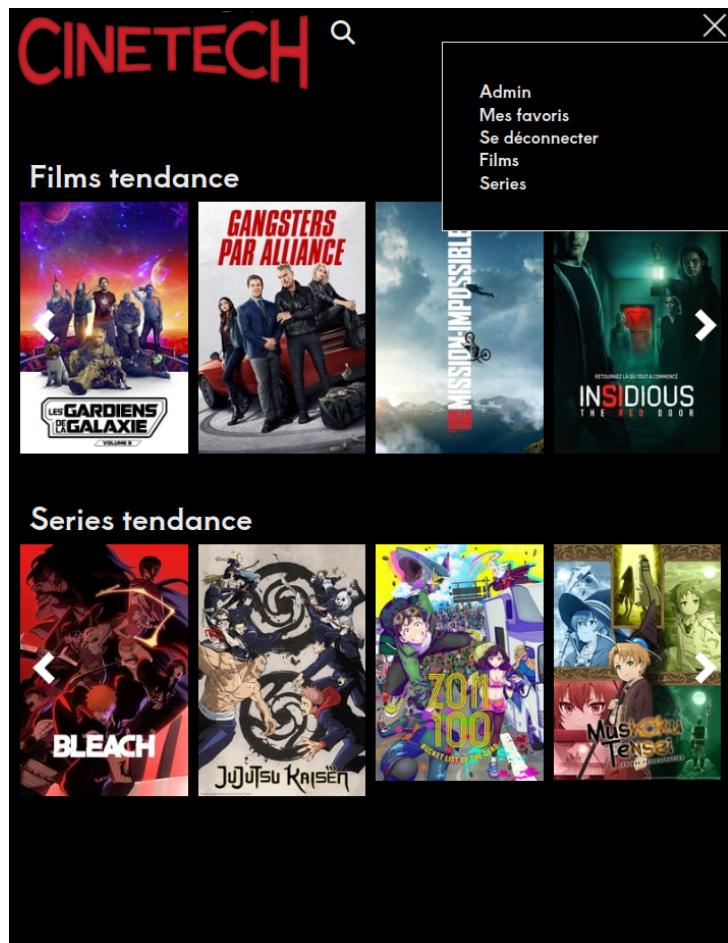
ul#list-menu.open {
    width: 40%;
    z-index: 9999;
    transition: all 0.5s ease-in-out;
    border: 1px solid ■#e9e9e9;
    line-height: 1.4em;
}
```

Films tendance



Séries tendance





Lorsque l'écran fait moins de 1200px de large, la liste des éléments du menu s'affiche horizontalement en haut du header au-delà de 1200px.

```

ul#list-menu {
    position: static;
    display: flex;
    flex-direction: row;
    justify-content: space-around;
    background-color: transparent;
    color: #e9e9e9;
    font-size: 1.5em;
    width: 900px;
    padding: 5px 0 0 0;
}

div#burger_div {
    display: none;
}

```

Films tendance



Séries tendance



Sécurité

Injection SQL

L'idée principale derrière l'injection SQL est d'insérer du code SQL malveillant dans une requête SQL existante, afin d'induire le serveur de base de données à exécuter des commandes non souhaitées ou non autorisées.

J'ai utilisé la méthode `prepare()` de la classe PDO pour prévenir la faille, grâce à `prepare()`, la requête SQL et les valeurs des paramètres sont envoyées au serveur de base de données séparément. Le serveur de base de données traite la requête et les valeurs des paramètres indépendamment, évitant ainsi l'interprétation incorrecte des valeurs en tant que commandes SQL.

Cela signifie que les valeurs des paramètres sont traitées comme des données brutes et non comme du code SQL potentiellement malveillant. Les caractères spéciaux et les instructions SQL intégrées aux valeurs des paramètres sont correctement échappés ou encodés, garantissant ainsi que la requête SQL ne sera pas altérée par des valeurs malveillantes.

Faille XSS

Une faille XSS consiste à injecter du code directement interprétable par le navigateur Web, comme, par exemple, du JavaScript ou du HTML. Cette attaque ne vise pas directement le site comme le ferait une injection SQL mais concerne plutôt la partie

client. Les possibilités sont nombreuses : redirection vers un autre site, vol de cookies, modification du code HTML de la page, exécution d'exploits contre le navigateur : en bref, tout ce que ces langages de script permettent de faire.

Pour se protéger contre la faille XSS, j'ai utilisé la fonction `htmlSpecialChars()` qui convertit les caractères spéciaux en entités HTML, ce qui empêche que les entrées utilisateur soient interprétées comme du code.

Changer le mot de passe

Il est recommandé aux utilisateurs de changer régulièrement de mot de passe car utiliser le même mot de passe pendant une longue période augmente les chances qu'il soit compromis. Utiliser des mots de passe forts et uniques est tout aussi important que le fait de les changer régulièrement.

Les utilisateurs peuvent changer de mot de passe sur la page "mon profile" :

The screenshot shows the Cinetech website's user profile page. At the top, there is a red logo with the word 'CINETECH' and a magnifying glass icon. On the right side, there is a three-line menu icon. Below the logo, the page title 'Mon Profil' is displayed in large white font. Underneath the title, the section 'Mes informations' is shown, listing the user's name, first name, and email address. At the bottom of the page, there is a form titled 'Changer mon mot de passe' with four input fields: 'Ancien mot de passe', 'Nouveau mot de passe', 'Confirmation du nouveau mot de passe', and a 'Modifier' button.

Conclusion

Ce projet a permis de voir le développement de A à Z d'un site web connecté à une API et une base de données.

Nous avons vu comment

- maquetter le site
- concevoir une base de données
- mettre en place une architecture MVC

- mettre en place un aultoloader
- mettre en place un router
- faire des appel à une API
- faire des requêtes à une base de données
- gérer le responsive
- contrer les principales failles de sécurité connues