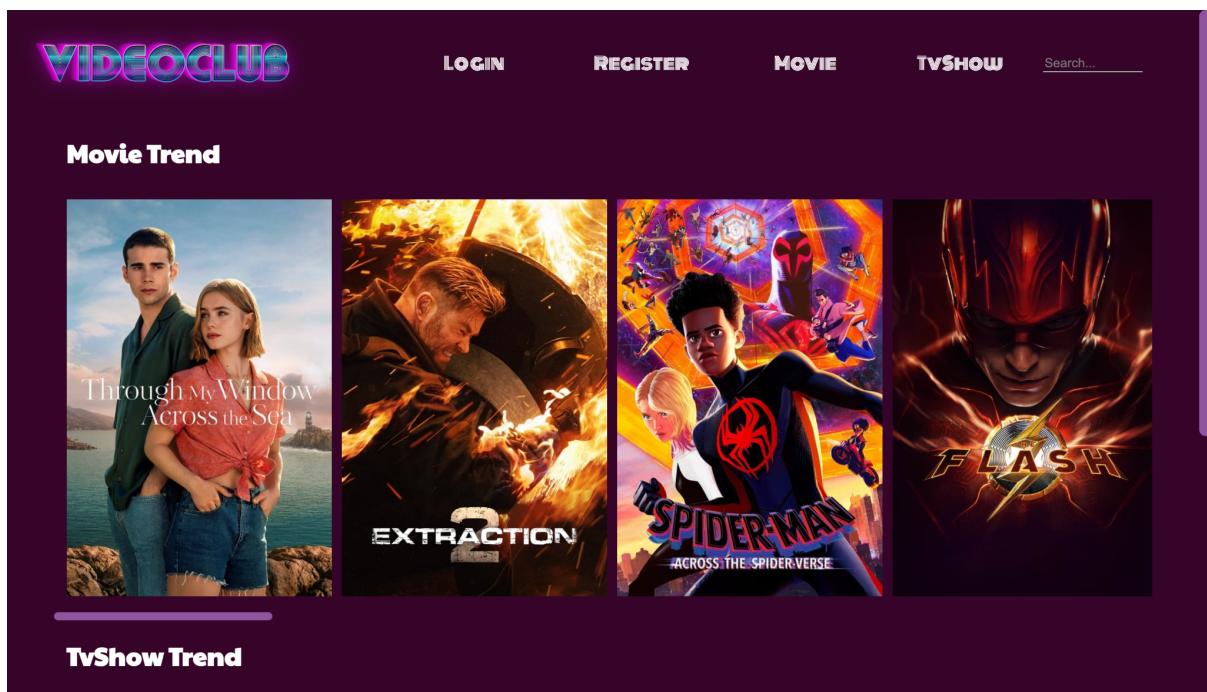


# TITRE : DÉVELOPPEUR WEB ET WEB MOBILE



Dossier de projet : Cinetech

Guangquan Ye

# Table des matières

<b>Compétences du référentiel couvertes par le projet</b>	<b>2</b>
<b>Spécifications fonctionnelles</b>	<b>3</b>
Description du projet	4
Périmètre du projet	4
Arborescence du site	4
Description des fonctionnalités	5
1.Authentification	5
2.Catalogue et filtre	5
3.Description de films et série	5
4.Suggestion de recherche	5
5.Favoris	5
6.Commentaires	5
7.Panel Administratif	5
Spécification technique	6
Choix techniques et environnement de travail	6
Architecture du projet	6
Réalisations	6
Charte graphique	6
Maquette	6
Conception de la base de données	6
Extraits de code significatifs	8
Authentification	9
Catalogue films/séries	12
Favoris	16
Annexe	18
maquette figma	18

## **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

# **Spécifications fonctionnelles**

## **Description du projet**

VideoClub est un site dédié aux amateurs de cinéma, offrant une vaste collection de films et de séries. Il permet aux utilisateurs découvrir les informations des titres variés provenant de l'API de TMDB. Le site propose des fonctionnalités telles que la recherche par genre ou recherche par nom du film. Les utilisateurs peuvent aussi ajouter les films favoris et ajouter des commentaires.

## **Périmètre du projet**

Le site est réalisé en anglais et ce dernier devra être accessible sur différents supports, à savoir mobile, tablette et ordinateur.

## **Arborescence du site**

L'arborescence du site se décline comme suit :

- Page d'accueil
- Page connexion
- Page inscription
- Page tous les films
- Page tous les series
- Page description film ou series
- Page favoris
- Page panel d'administration

## **Description des fonctionnalités**

### **0.Home**

une page d'accueil qui affiche les films/séries du moment.

### **1.Authentification**

Les utilisateurs peuvent s'inscrire rapidement et facilement via un formulaire d'inscription/connexion. Une fois inscrits, les utilisateurs ont accès à leur favoris.

### **2.Catalogue et filtre**

Une page dédiée aux films et une page dédiée aux séries et un filtre permettant de trier les films par genres, facilitant ainsi la recherche des utilisateurs.

### **3.Description de film et série**

L'utilisateur pourra accéder à une page descriptive du film ou série sélectionnée . Cette dernière comprend différents éléments tels que le synopsis, le casting, la date de sortie, etc...

### **4.Suggestions de recherche**

L'utilisateur pourra rechercher directement le film ou la série dans la barre de recherche. La fonction de recherche autocomplétion affiche une liste de suggestions basées sur les mots-clés ou les phrases entrés

### **5.Favoris**

Une fois connecté, l'utilisateur peut ajouter des films ou séries à ses favoris grâce à un bouton.

### **6.Commentaires**

l'utilisateur pourra laisser un commentaire sur les films ou séries sur la page descriptive de ce dernier.

### **7.Panel administrateur**

Un panel d'administration est disponible pour l'administrateur, ainsi il peut gérer les utilisateurs ou les commentaires.

## Spécifications techniques

### Choix techniques et environnement de travail

#### Technologies utilisées pour la partie back-end :

- Le projet sera réalisé avec le langage PHP ( Hypertext Preprocessor)
- Base de données SQL
- Gestionnaire de dépendance: Composer

#### Technologies utilisées pour la partie front-end:

- Le projet sera réalisé avec HTML et CSS.
- Javascript afin de dynamiser le site et d'améliorer l'expérience utilisateur.

#### L'environnement de développement est le suivant:

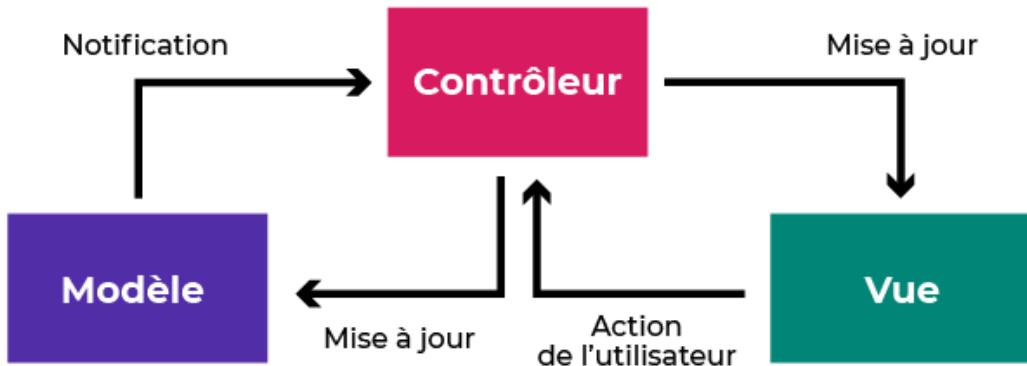
- Editeur de code: Visual Studio Code
- Outil de versioning: GIT, Github
- Maquettage: Figma

## Architecture du projet

L'architecture du site est développée avec le design pattern type MVC.

L'architecture MVC est l'une des architectures les plus utilisées pour les applications Web, elle se compose de 3 modules:

- Modèle: noyau de l'application qui gère les données, permet de récupérer les informations dans la base de données, de les organiser pour qu'elles puissent ensuite être traitées par le contrôleur.
- Vue: composant graphique de l'interface qui permet de présenter les données du modèle à l'utilisateur.
- Contrôleur: composant responsable des prises de décisions, gère la logique du code, il est l'intermédiaire entre le modèle et la vue.



## Réalisations

### 1. Charte graphique

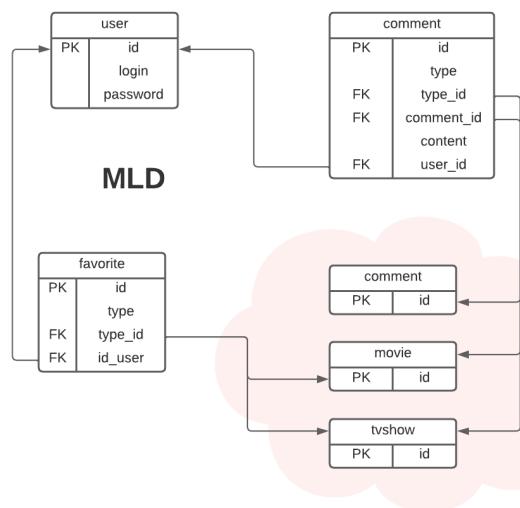
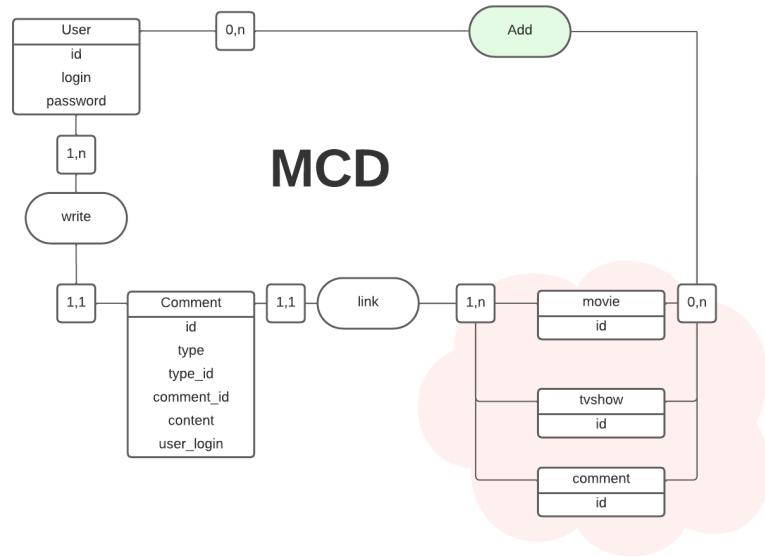
Les polices de caractères sont les suivantes : Roboto et Paytone  
La couleur principale est la suivante : #360429

### 2. Maquette

La maquette est réalisée avec figma, un logiciel gratuit.

### 3. Conception de la base de données

Au regard des fonctionnalités demandées par le projet, on a créé la base de données suivante :



La base de données s'articule autour de la table `user`, qui va permettre d'identifier les utilisateurs. Cette dernière est liée à la table “comment” qui va permettre de relier les commentaires au films/séries concernées.

La table “favorite” stocke les films/séries sélectionnés par l’utilisateur dans L’API.

## Extrait de code significatifs

### home

Concernant la page d'accueil on crée une interface interactive pour parcourir les films/séries tendances grâce à un script Javascript qui récupère les séries en utilisant l'API The Movie Database(TMDB) via une requête HTTP et les affiche dans une page PHP.



```
let options = {
  method: 'GET',
  headers: {
    accept: 'application/json',
    Authorization: 'Bearer
eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkOWRlNjFjMzk3YzEwZTA
4YTI5M2UyOTgyYmYzNzdmNCIsInN1YiI6IjY0NjFlZDgzzGJiYjQ
yMDE1MzA2YmQzMCIisInjb3BlcyI6WyJhcGlfcmVhZCJdLCJ2ZXJ
zaW9uIjoxfQ.Zt5c8JdoJm2dmE-y7Xkt7-
8PoRXsZ8qx60YIx5YyJGU'
  }
};

try {
  let response = await
fetch("https://api.themoviedb.org/3/trending/tv/day?
language=en-US", options);
  let tvshows = await response.json();
```

on crée un objet qui a la variable “**options**” et qui contient les propriétés :

- **“method”**: Une propriété qui spécifie la méthode HTTP utilisée pour la requête. Dans ce cas, la valeur est 'GET', indiquant qu'il s'agit d'une requête de lecture.

- **“headers”**: Une propriété qui spécifie les en-têtes HTTP à inclure dans la requête. Dans cet exemple, il y a deux en-têtes spécifiés :

- **“accept”**: Cet en-tête indique au serveur que le client (dans ce cas, le code JavaScript) accepte la réponse au format JSON.

- **“Authorization”**: Cet en-tête est utilisé pour l'authentification. Il contient un jeton d'accès (access token) qui est utilisé pour vérifier l'identité et les autorisations du client. Dans cet exemple, le jeton d'accès est une chaîne de caractères longue et complexe.

Ensuite, on utilise ces options lors de l'appel à la fonction “**fetch()**” pour effectuer une requête GET à l'URL "<https://api.themoviedb.org/3/trending/tv/day?language=en-US>"

La réponse de la requête est stockée dans la variable “**response**”, puis la fonction “**response.json()**” est appelée pour extraire les données JSON de la réponse. Ces données JSON contiennent les informations sur les séries tendances, qui sont ensuite stockées dans la variable “**tvshows**”.

## Alto-router

Pour le projet on crée un routeur via Alto-router sur Composer qui va me permettre de faciliter la gestion des routes et des URL.

```
$router = new AltoRouter();

$router->setBasePath('/cinetech');

$router->map('GET', '/', function () {
    require_once "home.php";
});

$router->map('GET', '/movie', function () {

    $movie = new MovieController();
    $movie->getMovie();

});
```

## Authentification

Pour l'authentification je crée d'abord une page HTML qui va contenir le formulaire inscription/connexion qu'on appelle via un bouton écouté par un script JS

```
● ● ●  
  
<div class="login-box">  
  <h2>Login</h2>  
  <p id="logMsg"></p>  
  <form action="login" method="POST"  
        class="logForm">  
    <div class="user-box">  
      <input type="text" name="logLogin"  
            required="">  
      <label>Login</label>  
    </div>  
    <div class="user-box">  
      <input type="password" name="logPwd"  
            required="">  
      <label>Password</label>  
    </div>  
    <button type="submit" name="logBtn"  
          class="authBtn">Login</button>  
  </form>  
</div>
```

```
● ● ●  
  
<div class="login-box">  
  <h2>Register</h2>  
  <p id="regMsg"></p>  
  <form action="register" method="POST"  
        class="regForm">  
    <div class="user-box">  
      <input type="text" name="regLogin"  
            required="">  
      <label>Login</label>  
    </div>  
    <div class="user-box">  
      <input type="password" name="regPwd"  
            required="">  
      <label>Password</label>  
    </div>  
    <div class="user-box">  
      <input type="password" name="regPwdConf"  
            required="">  
      <label>Confirm</label>  
    </div>  
    <button type="submit" name="regBtn"  
          class="authBtn">Register</button>  
  </form>  
</div>
```

```
● ● ●  
  
const regDisplayBtn =  
document.querySelector("#regDisplayBtn");  
const loginDisplayBtn =  
document.querySelector("#loginDisplayBtn");
```

Lorsque l'utilisateur clique sur un bouton, la fonction asynchrone "**regDisplayBtn**" est déclenchée. Cette fonction récupère le formulaire d'inscription à partir de l'URL **/cinetech/register** en utilisant la méthode HTTP GET. Le formulaire est alors affiché à l'utilisateur, qui peut le remplir avec ses informations d'inscription.

Une fois que l'utilisateur a rempli le formulaire et appuie sur le bouton **"Register"**, la fonction est exécutée. Cette fonction envoie les valeurs du formulaire vers le routeur en utilisant l'URL **/cinetech/register** et la méthode HTTP POST.

```
● ● ●  
  
form.addEventListener("submit", async (e) => {  
  e.preventDefault();  
  
  const formData = new FormData(form);  
  try {  
    const response = await fetch(form.action, {  
      method: form.method,  
      body: formData,  
    });
```

Le routeur appelle la fonction “**createUsers()**” de la classe “**UserController**”

```
● ○ ●

$router->map('GET', '/register', function(){
    $user = new UserController();
    $user->regFormDisplay();
});

$router->map('POST', '/register', function(){
    if(isset($_POST)){
        $user = new UserController();
        $user->createUsers($_POST["regLogin"], $_POST["regPwd"], $_POST["regPwdConf"]);
    }
});
```

```
● ○ ●

public function createUsers($login, $password, $passwordConf)
{
    if($password == $passwordConf){

        if($this->user->verifyExist($login)){
            echo "Login already exist";
        }else{
            $specialLogin = htmlspecialchars($login);
            $specialPwd = htmlspecialchars($password);
            $hashedPwd = password_hash($specialPwd, PASSWORD_DEFAULT);
            $this->user->insert($specialLogin, $hashedPwd);

            echo "Successfully Submit";
        }
    }else{
        echo "Pwd and confirm do not match";
    }
}
```

la fonction “**createUsers()**” avec trois paramètres : “**\$login**”, “**\$password**” et “**\$passwordConf**”. Ces paramètres représentent le nom d'utilisateur, le mot de passe et la confirmation du mot de passe saisis par l'utilisateur.

On va d'abord convertir les paramètres “**\$login**”, “**\$password**” et “**\$passwordConf**” en caractères spéciaux éventuels avec la fonction “**htmlspecialchars**” en entité HTML. Ce qui aide à prévenir les attaques XSS.

on établit ensuite deux conditions :

- la première “**if(\$password == \$passwordConf)**” vérifie si le mot de passe correspond à la confirmation du mot de passe.

- la deuxième “**if(\$this->user->verifyExist(\$login))**” vérifie si le nom d’utilisateur existe déjà dans la base de données appelant la méthode “**verifyExist()**” sur l’objet “**\$this->user**”. Si le login existe déjà, le message “Login already exists” est affiché.

Si le mot de passe correspond à la confirmation du mot de passe et que le login n’existe pas, on hash le mot de passe avec la fonction “**password\_hash()**” qui est ensuite stocké dans la variable “**\$hashedPwd**”.

Et on insère le “**\$specialLogin**” et “**\$hashedPwd**” dans la méthode “**insert()**” de la classe “**UserModel**”.



```

public function insert($login, $password){

    $insert = "INSERT INTO user (login, password) VALUES (:login, :password)";
    $prepare = DbConnexion::getDb()->prepare($insert);
    $prepare->execute([
        "login" => $login,
        "password" => $password,
    ]);
}

```

Une fois dans le “**Model**” on prépare la requête SQL d’insertion qui insère les valeurs dans la table “**user**”. Les paramètres nommés sont utilisés dans la requête pour améliorer la lisibilité et la maintenabilité du code. De plus, l’utilisation de requêtes préparées avec des paramètres nommés aide à prévenir les attaques d’injection SQL en séparant clairement les valeurs des parties de la requête, ce qui permet au système de gérer correctement l’encodage et l’échappement des données pour renforcer la sécurité.

## Catalogue et filtre

Pour la page du catalogue, on crée deux pages PHP distinctes : “**movie.php**” pour les films et “**tvShow.php**” pour les séries. Dans chacune de ces pages, on intègre un script JavaScript qui utilise l’API TMDB pour rechercher et afficher les films ou séries correspondants.

Pour la page “**tvShow.php**” par exemple, on crée deux constantes :

- “**const currentPage = 1**” : Cette constante maintient le numéro de page actuel pour la pagination des résultats de séries.

- “**const tvshowsPerPage = 20**” : Cette constante définit le nombre de séries à afficher par page.

et une fonction “**getGenre()**” qui récupère les genres de séries à partir de l’API TMDB. Elle effectue une requête GET vers l’URL [“https://api.themoviedb.org/3/genre/tv/list?language=en”](https://api.themoviedb.org/3/genre/tv/list?language=en) et récupère les données JSON correspondantes.

```
● ● ●

async function getGenres() {
  let options = {
    method: 'GET',
    headers: {
      accept: 'application/json',
      Authorization: 'Bearer eyJhbGciO...'
    }
};

try {
  let response = await fetch("https://api.themoviedb.org/3/genre/tv/list?
language=en", options);
  let data = await response.json();
  let genres = data.genres;
```

A partir des données JSON la fonction crée des cases à cocher pour chaque genre de série. Lorsqu’une case à cocher est cochée ou décochée, les fonctions “**check(value)**” ou “**uncheck(value)**” sont appelées respectivement.

```
● ● ●

const checkboxDiv = document.querySelector("#checkboxDiv");

for (let genre of genres) {
  const onecheckbox = document.createElement("div");
  onecheckbox.classList.add("onecheckbox");

  const checkbox = document.createElement('input');
  checkbox.type = 'checkbox';
  checkbox.value = genre.id;

  checkbox.addEventListener('change', function () {
    if (checkbox.checked) {
      check(checkbox.value);
    } else {
      uncheck(checkbox.value);
    }
  });
}
```

Les fonctions “**check(value)**” et “**uncheck(value)**” sont des fonctions de rappel pour les cases à cocher. Elles ajoutent ou suppriment la valeur du genre sélectionné en appelant respectivement “**composeArray(value)**” ou “**decomposeArray(value)**”.

Les fonctions “**composeArray(value)**” et “**decomposeArray(value)**” sont utilisées pour ajouter ou supprimer des valeurs au tableau “**genreArray[ ]**” qui stocke les genres sélectionnés.

```
const genreArray = [];

function composeArray(value) {
    genreArray.push(value);
    currentPage = 1;
    updateGenresString();
}

function decomposeArray(value) {
    const index = genreArray.indexOf(value);
    if (index !== -1) {
        genreArray.splice(index, 1);
        currentPage = 1;
        updateGenresString();
    }
}
```

On convertit ensuite le tableau “**genreArray[ ]**” en une chaîne de caractères où les valeurs sont séparées par des virgules avec la fonction “**arrayToString(array)**”.

La fonction “**updateGenresString()**” est appelée chaque fois qu'un genre est ajouté ou supprimé. Elle utilise “**arrayToString(genreArray)**” pour obtenir une chaîne de caractères représentant les genres sélectionnés, puis appelle “**getTVShowByGenre(genresString, currentPage, tvshowsPerPage)**” pour récupérer et afficher les séries correspondantes.



```
function updateGenresString() {
    const genresString = arrayToString(genreArray);
    getTVShowByGenre(genresString, currentPage, tvshowsPerPage);
}
```

La fonction “**getTVShowByGenre(genresString, page, perPage)**” récupère les séries correspondant aux genres sélectionnés. Elle effectue une requête GET vers l'URL "<https://api.themoviedb.org/3/discover/tv>" avec les paramètres “**with\_genres**”, “**page**” et “**per\_page**” et affiche les résultats dans la section avec l'ID “tvshowDisplay” de la page PHP.



```
tvshowDisplay.innerHTML = '';
for (let i = 0; i < data.results.length; i++) {
    if (i < perPage) {
        let tvshow = data.results[i];
        tvshowDisplay.innerHTML += `
            <div class="tvshowDiv">
                <a href="/cinetech/tv/${tvshow.id}"></a>
            </div>
        `;
    }
}
```

Pour la pagination, on a créé deux boutons, “**Previous**” et “**Next**”. Lorsque l'utilisateur clique sur le bouton “**Previous**”, la fonction “**previousPage()**” est déclenchée pour mettre à jour la valeur de la page courante “**currentPage**” et afficher les séries de la page précédente. De même, lorsque l'utilisateur clique sur le bouton “**Next**”, la fonction “**nextPage()**” est déclenchée pour mettre à jour la valeur de la page courante et afficher les séries de la page suivante en utilisant la fonction “**updateGenresString()**”.

```
function previousPage() {
    if (currentPage > 1) {
        currentPage--;
        updateGenresString();
    }
}

function nextPage() {
    currentPage++;
    updateGenresString();
}
```

## Favoris

Pour l'ajout en favoris, on récupère le chemin d'accès actuel de la page à l'aide de “`window.location.pathname`” et le stocke dans la variable “**favUri**”.

Puis on divise le “**favUri**” en segments en utilisant le caractère “/” comme séparateur et stocke les segments dans la variable “**favParts**”.

On extrait ensuite les deux derniers segments en utilisant les index -1 et -2 de “favParts”. Ces segments sont respectivement stockés dans les variables “favTypeId” et “favTypeName”.

```
const favUri = window.location.pathname;  
const favParts = favUri.split("/");  
const favTypeId = favParts[favParts.length - 1];  
const favTypeName = favParts[favParts.length - 2];
```

Lorsque le bouton “**addFavoriteBtn**” est cliqué on active la fonction “**addFavorite**” avec les paramètres “**favTypeIid**” et “**favTypeName**”. À l’intérieur de la fonction “**addFavorite**”, un objet “**FormData**” est créé et les valeurs “ok”, “**favTypeIid**” et “**favTypeName**” sont ajoutées en tant que paires clé-valeur. On envoi ensuite “**FormData**” vers le routeur en utilisant l’URL “/cinetech/favorite” et la méthode HTTP POST.

```
let formData = new FormData();
formData.append("favorite", 'ok');
formData.append("typeId", favTypeId);
formData.append("typeName", favTypeName);

addFavoriteBtn.disabled = true;

fetch('/cinetech/favorite', {
  method: 'POST',
  body: formData,
})
```

Le routeur va ensuite appeler la fonction “**addFavorite()**” de Class “**FavoriteController**”.

```
$router->map('POST', '/favorite', function(){
  if(isset($_POST["favorite"])){
    $fav = new FavoriteController();
    $fav->addFavorite($_POST["typeName"],
$_POST["typeId"], $_SESSION["user"]["id"]);
  }
}, 'addFav');
```

Le Controller envoie ensuite les valeurs au Model “**FavoriteModel**”.

```
public function addFavorite($type, $typeId, $userId)
{
  $this->fav->insert($type, $typeId, $userId);
}
```

La fonction “**insert**” du “**FavoriteModel**” effectue une requête SQL d’insertion dans la table “**favorite**” avec les valeurs fournies.

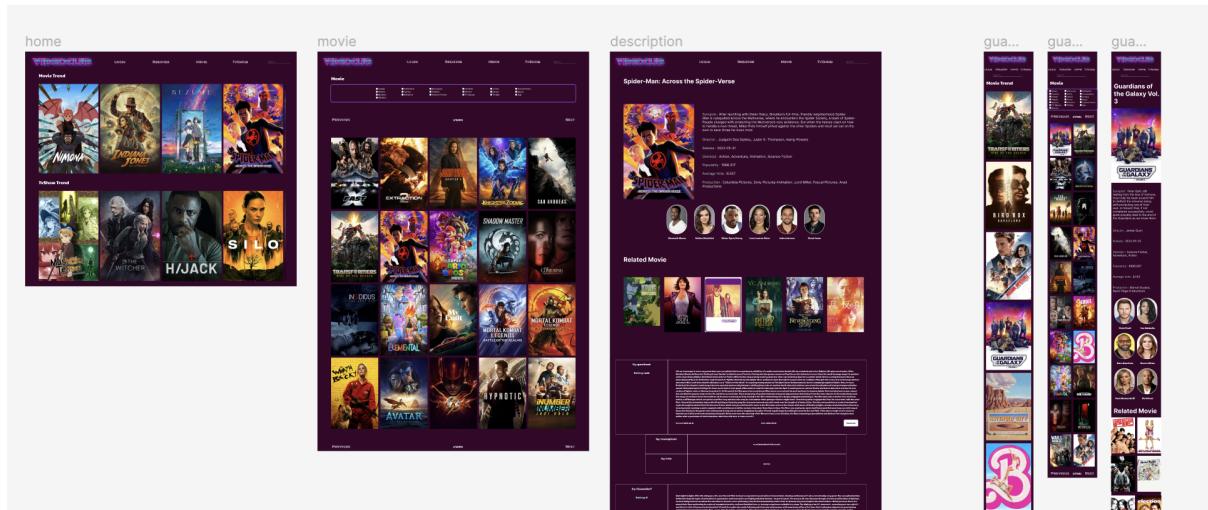


```
public function insert($type, $typeId, $userId){

    $insert = "INSERT INTO favorite (type, id_type, id_user) VALUES (:type,
:id_type, :id_user)";
    $prepare = DbConnexion::getDb()->prepare($insert);
    $prepare->execute([
        "type" => $type,
        "id_type" => $typeId,
        "id_user" => $userId
    ]);
}
```

## Annexe

### figma



figma de la boutique en ligne

The image displays a grid of 12 mobile device screenshots, each showing a different page from a coffee shop website. The pages are arranged in three rows:

- Row 1:**
  - Index:** A page with a large red rectangular placeholder.
  - Shop:** A page featuring a purple and red color scheme.
  - About:** A page with a light gray background.
  - Contact:** A page titled "Formulaire de contact" with fields for name, email, phone, and message.
- Row 2:**
  - Idea 2:** A page titled "START YOUR DAY WITH OUR COFFEE" featuring a photo of a coffee cup and a landscape.
  - Index ...** A page showing a grid of coffee products.
  - Shop m...:** A page titled "COLD BREW" showing a product image.
  - Article ...:** A page showing a grid of coffee products.
- Row 3:**
  - Connection:** A login page with fields for email and password.
  - Register:** A registration page with fields for name, email, phone, and password.
  - panier ...:** A shopping cart page showing a grid of coffee products.
  - Shop:** A page titled "Our Beans" showing a grid of coffee products.
  - Register:** A registration page with fields for name, email, phone, and password.
  - Product:** A product detail page showing a coffee bag and its details.