



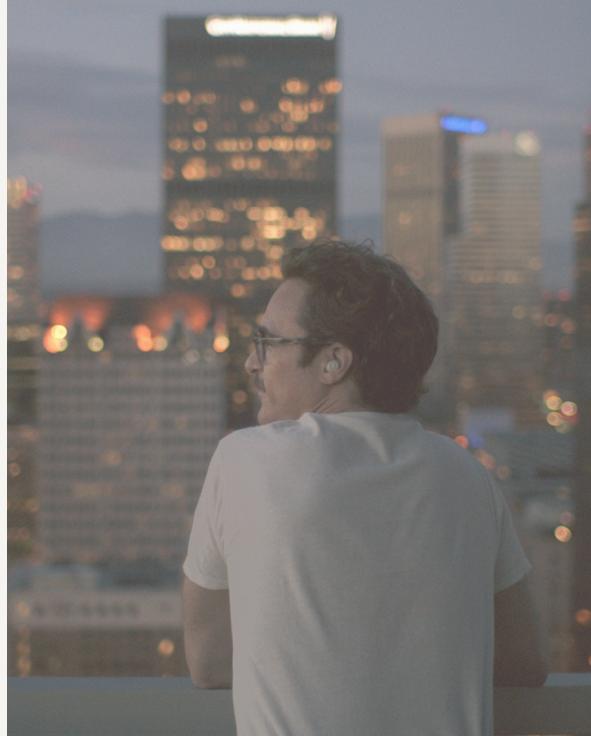
Dossier de Projet

*Titre Professionnel de Développeur Web ↗
et Web Mobile*

ORPHEUS
COLLECTION



"L'accord parfait entre musique et cinéma"



Inès Bakhtaoui

Sommaire

PRÉAMBULE

- 2 - TABLE DES MATIÈRES
- 3 - RÉSUMÉ DU PROJET
- 4 - CAHIER DES CHARGES
- 5-6 - SPÉCIFICATIONS TECHNIQUES
- 7 - COMPÉTENCES DU REFERENTIEL
- 8 - MAQUETTAGE
- 9 - ORGANISATION DU SITE

JMERISE

- 10-11 - MCD ET MLD
- 12-13 - GÉNÉRATION DU CODE SQL
- 13 - BASE DE DONNÉES

MVC, PDO & CRUD

- 14 - MVC
- 14-17 - PDO
- 18-29 - CRUD

CONCLUSION

- 30 - LES FAILLES DE SÉCURITÉ
- 31 - TRADUCTION D'UNE PROBLÉMATIQUE EN ANGLAIS
- 32 - L'AVENIR DU PROJET
- 33 - REMERCIEMENTS



Résumé du Projet



QU'EST CE QUE? ORPHEUS COLLECTION

Dans la lignée des sites tels que "SensCritique" ou encore, "IMDb", "Orpheus Collection" a pour vocation d'être un espace **informatif, collaboratif et interactif**, consacré aux **bandes originales de films**. L'idée principale étant de créer une plateforme d'**échanges, de découvertes, et d'expression**, consacrée à une thématique qui nous parle et nous touche tous, de près ou de loin. Quelles œuvres vous inspirent la **Mélancolie**, ou la **Peur**? Quelles œuvres vous **Bouleversent** et vous donnent l'impression d'être un **Héros**? J'espère arriver à rassembler des internautes et créer des espaces d'échanges sur des œuvres qui nous **fascinent** et nous **passionnent**.

"Orpheus Collection" a émergé d'une vague idée qui était de créer un site sur le cinéma. Puis, il m'est apparu évident de parler d'un sujet qui me passionne depuis des années: "**l'expression des sentiments à travers l'Art**", ou comment, en tant qu'êtres humains, arrivons-nous à nous comprendre les uns et les autres, à nous sentir plus proche et plus en phase avec les autres, grâce à l'Art.

Le **Romantisme**, mouvement culturel du XVIII^e siècle, m'a aussi largement inspiré. Il se caractérise par une volonté de l'artiste d'**explorer toutes les possibilités de l'art** afin d'exprimer ses **états d'âme**.

Les **sentiments, l'évasion et le rêve**...en sont les thèmes privilégiés.

J'ai choisi de traiter ce vaste sujet, en abordant la bande originale de film, car elle est souvent intemporelle et touche à la fois, les adeptes de musiques, et de cinéma. A écouter seule, elle stimule l'imaginaire, et au sein de son environnement cinématographique, elle illustre et accompagne l'imagerie du réalisateur.

Ce projet tend à être développé pour permettre plus de fonctionnalités à l'utilisateur, notamment sous forme d'application mobile, puis dans sa thématique même (se reporter à la page "L'avenir du Projet" p.32).

Cahier des Charges

BESOINS ET FONCTIONNALITÉS

Les utilisateurs doivent pouvoir **s'inscrire**, **voter** pour une ost, **lister** des ost, **écrire des commentaires** sous les articles, écrire des **mini-posts** sur l'ost de leur choix, obtenir des **badges** en fonction de leur progression sur le site.

Un utilisateur **non connecté** peut avoir accès aux **articles**, à l'**index des musiques** et tous les **commentaires** et **mini-posts** des utilisateurs (dans la limite de ceux affichés sur les pages correspondantes). En revanche, il ne peut en écrire, ni voter pour une musique.

Les musiques de films doivent pouvoir apparaître dans un **index**, et être triées par **catégorie**, **album**, **compositeur** et **titre**.

Pour chaque catégorie, une liste s'alimente continuellement des **votes** des utilisateurs. Cette liste affiche dans une **ordre décroissant**, les musiques qui ont reçu le plus de votes.

De plus, une musique est **sélectionnée aléatoirement** et est mise en avant sur la **catégorie** en question. La musique aléatoire change à chaque **rafraîchissement** de la page. Le but étant de faire **découvrir** de nouvelles musiques, dès que cela est possible.

En effet, l'objectif étant de créer un site qui permet à l'utilisateur de : **s'informer** sur le sujet (grâce aux **articles** écrits par l'administrateur, **flux RSS...**), mais aussi, **s'inscrire** pour : **voter** pour sa bande originale préférée d'une catégorie précise, **créer des listes personnalisables** et des listes d'**écoute**, **écouter** et **découvrir** des bandes originales grâce aux **lecteurs multimédias**, écrire des "mini-posts", afin d'exprimer tout ressenti sur une musique, qui seront visibles publiquement sur le site .



Spécifications Techniques

ENVIRONNEMENT DE DÉVELOPPEMENT

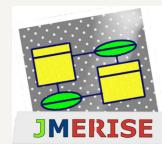


Visual Studio Code

VISUAL STUDIO CODE EST UN ÉDITEUR DE CODE EXTENSIBLE DÉVELOPPÉ PAR MICROSOFT POUR WINDOWS, LINUX ET MACOS. LES FONCTIONNALITÉS INCLUENT LA PRISE EN CHARGE DU DÉBOGAGE, LA MISE ÉVIDENCE DE LA SYNTAXE, LA COMPLÉTION INTELLIGENTE DU CODE, LES SNIPPETS, LA REFACTORISATION DU CODE ET GIT INTÉGRÉ.



C'EST UN SERVICE WEB D'HÉBERGEMENT ET DE GESTION DE DÉVELOPPEMENT DE LOGICIELS, UTILISANT LE LOGICIEL DE GESTION DE VERSIONS GIT.



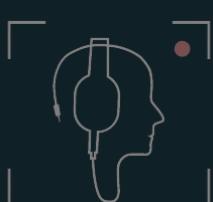
C'EST UNE MÉTHODE D'ANALYSE, DE CONCEPTION ET DE GESTION DE PROJET INFORMATIQUE. IL PERMET DE CRÉER GRAPHIQUEMENT LA CONCEPTION DE LA BASE DE DONNÉES.



C'EST UNE APPLICATION WEB DE GESTION POUR LES SYSTÈMES DE GESTION DE BASE DE DONNÉES MYSQL ET MARIADB, RÉALISÉE PRINCIPALEMENT EN PHP.



CE LANGAGE EST UN PROGRAMME DE BASE EN MATIÈRE DE STRUCTURATION, DE MISE EN RÉSEAU ET DE CONTENU SUR LE WORLD WIDE WEB.





LES FEUILLES DE STYLE EN CASCADE, GÉNÉRALEMENT APPELÉES CSS, FORMENT UN LANGAGE INFORMATIQUE QUI DÉCRIT LA PRÉSENTATION DES DOCUMENTS HTML ET PERMET AINSI DE LEUR DONNER DU STYLE.



JAVASCRIPT EST UN LANGAGE DE PROGRAMMATION DE SCRIPTS PRINCIPALEMENT EMPLOYÉ DANS LES PAGES WEB INTERACTIVES ET EST DONC UNE PARTIE ESSENTIELLE DES APPLICATIONS WEB.



C'EST UNE COLLECTION D'OUTILS UTILES À LA CRÉATION DU DESIGN (GRAPHISME, ANIMATION ET INTERACTIONS AVEC LA PAGE DANS LE NAVIGATEUR, ETC.) DE SITES ET D'APPLICATIONS WEB. C'EST UN ENSEMBLE QUI CONTIENT DES CODES HTML ET CSS, DES FORMULAIRES, BOUTONS, OUTILS DE NAVIGATION ET AUTRES ÉLÉMENTS INTERACTIFS, AINSI QUE DES EXTENSIONS JAVASCRIPT EN OPTION. C'EST L'UN DES PROJETS LES PLUS POPULAIRES SUR LA PLATE-FORME DE GESTION DE DÉVELOPPEMENT GITHUB.



C'EST UN LANGAGE DE PROGRAMMATION PRINCIPALEMENT UTILISÉ POUR PRODUIRE DES PAGES WEB DYNAMIQUES VIA UN SERVEUR HTTP. IL EST CONSIDÉRÉ COMME UNE DES BASES DE LA CRÉATION DE SITES WEB DITS DYNAMIQUES MAIS ÉGALEMENT DES APPLICATIONS WEB.

PDO:

C'EST UNE EXTENSION DÉFINISSANT L'INTERFACE POUR ACCÉDER À UNE BASE DE DONNÉES AVEC PHP. ELLE EST ORIENTÉE OBJET, LA CLASSE S'APPELANT PDO. LE PDO CONSTITUE UNE COUCHE D'ABSTRACTION QUI INTERVIENT ENTRE L'APPLICATION PHP ET UN SYSTÈME DE GESTION DE BASE DE DONNÉES (SGDB) TEL QUE MYSQL, POSTGRESQL OU MARIADB PAR EXEMPLE.



LE TERME MYSQL DÉSIGNE UN SERVEUR DE BASE DE DONNÉES DISTRIBUÉ SOUS LICENCE LIBRE GNU (GENERAL PUBLIC LICENSE). IL EST, LA PLUPART DU TEMPS, INTÉGRÉ DANS LA SUITE DE LOGICIELS LAMP QUI COMPREND UN SYSTÈME D'EXPLOITATION (LINUX), UN SERVEUR WEB (APACHE) ET UN LANGAGE DE SCRIPT (PHP). DANS LA PRATIQUE, LE SERVEUR MYSQL PEUT SE RÉSUMER À UN LIEU DE STOCKAGE ET D'ENREGISTREMENT DES DONNÉES. IL EST ALORS ENSUITE POSSIBLE, VIA UNE REQUÊTE SQL, D'ALLER RÉCUPÉRER DES INFORMATIONS SUR CE SERVEUR



C'EST UNE PLATEFORME DE CONCEPTION GRAPHIQUE. JE L'AI NOTAMMENT UTILISÉ POUR CONCEVOIR LE LOGO DE MON SITE, ET LES PAGES DE CE DOSSIER.

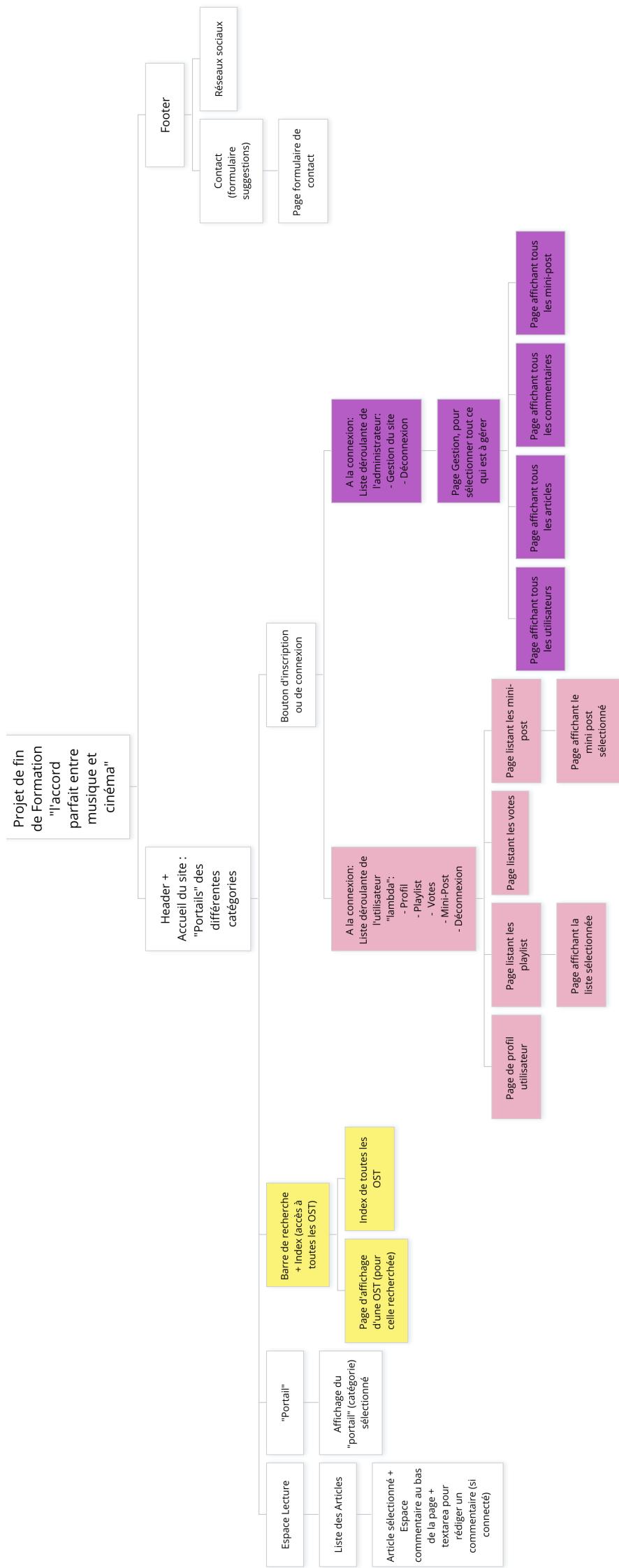
Compétences du Référentiel



REFERENTIEL EMPLOI ACTIVITES COMPETENCES DU TITRE PROFESSIONNEL

N° Fiche AT	Activités types	N° Fiche CP	Compétences professionnelles
1	Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité	1	Maquetter une application
		2	Réaliser une interface utilisateur web statique et adaptable
		3	Développer une interface utilisateur web dynamique
		4	Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce
		5	Créer une base de données
2	Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité	6	Développer les composants d'accès aux données
		7	Développer la partie back-end d'une application web ou web mobile
		8	Elaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce

Maquette



Organisation du Site



PAGE D'ACCUEIL:

- Accès aux pages des différentes catégories
- Accès à la barre de menu où sont l'espace d'inscription et de connexion
- Accès aux articles
- Accès à l'index des musiques

PAGE PROFIL:

- Accès à ses mini-posts, ses playlists, ses articles, ses badges
- Accès à la page d'édition de profil

PAGE MINI-POSTS:

- Accès à un mini-post (pour le modifier ou le supprimer)
- Accès à la page de création d'un mini-post

PAGE MUSIQUE:

- Affichage de la musique sélectionnée
- Lien de redirection pour l'achat de cette musique
- Accès à la catégorie de la musique

PAGE PLAYLISTS:

- Accès à l'affichage d'une playlist (pour la modifier ou la supprimer)
- Accès à la page de création d'une playlist

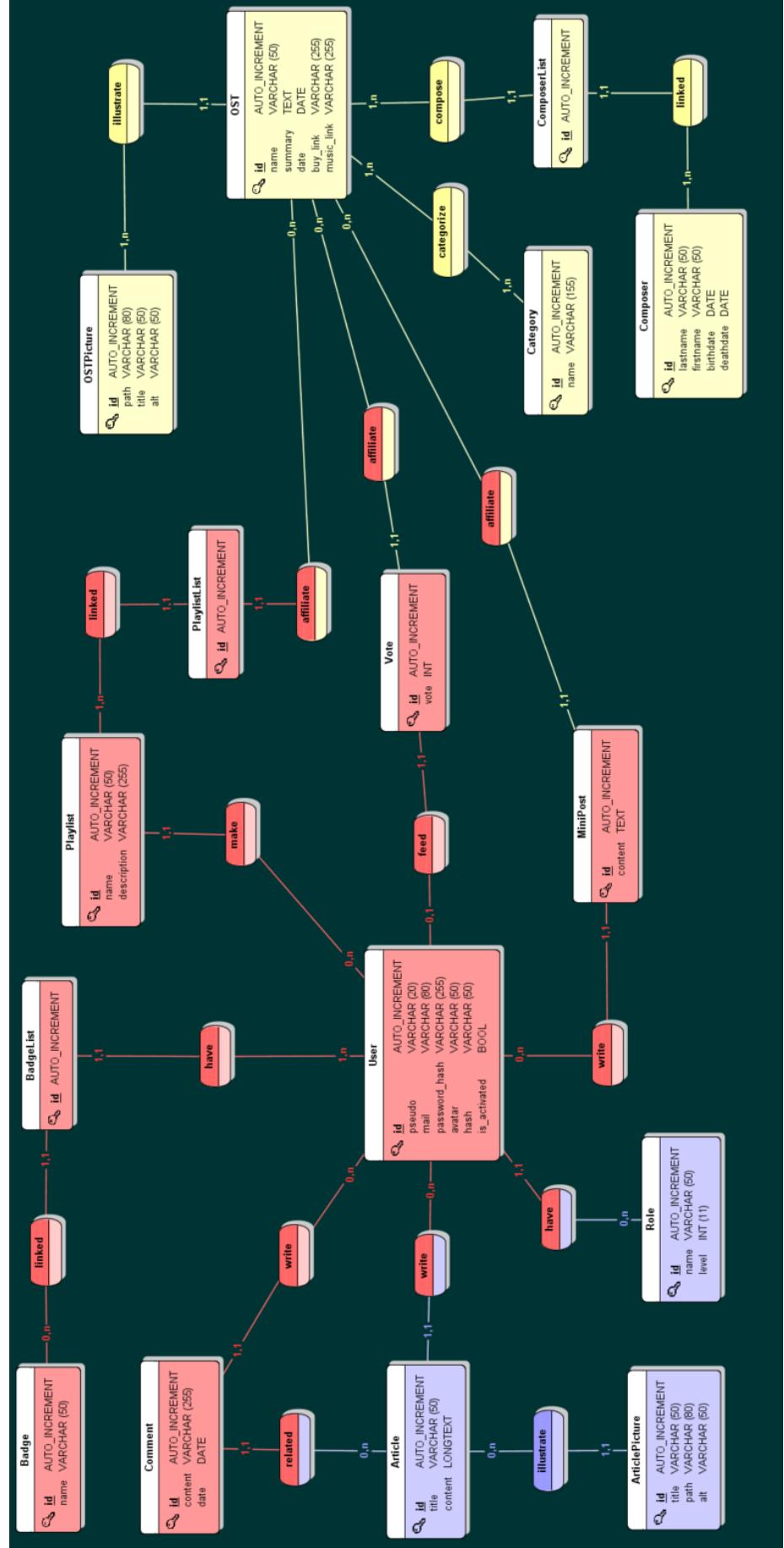
PAGE ARTICLES:

- Accès à l'affichage d'un article
- Accès aux commentaires de l'article sélectionné

PAGE CATÉGORIE:

- Affichage d'une musique aléatoire
- Affichage de mini-posts
- Affichage des votes

MCD & MLD



LÉGENDE

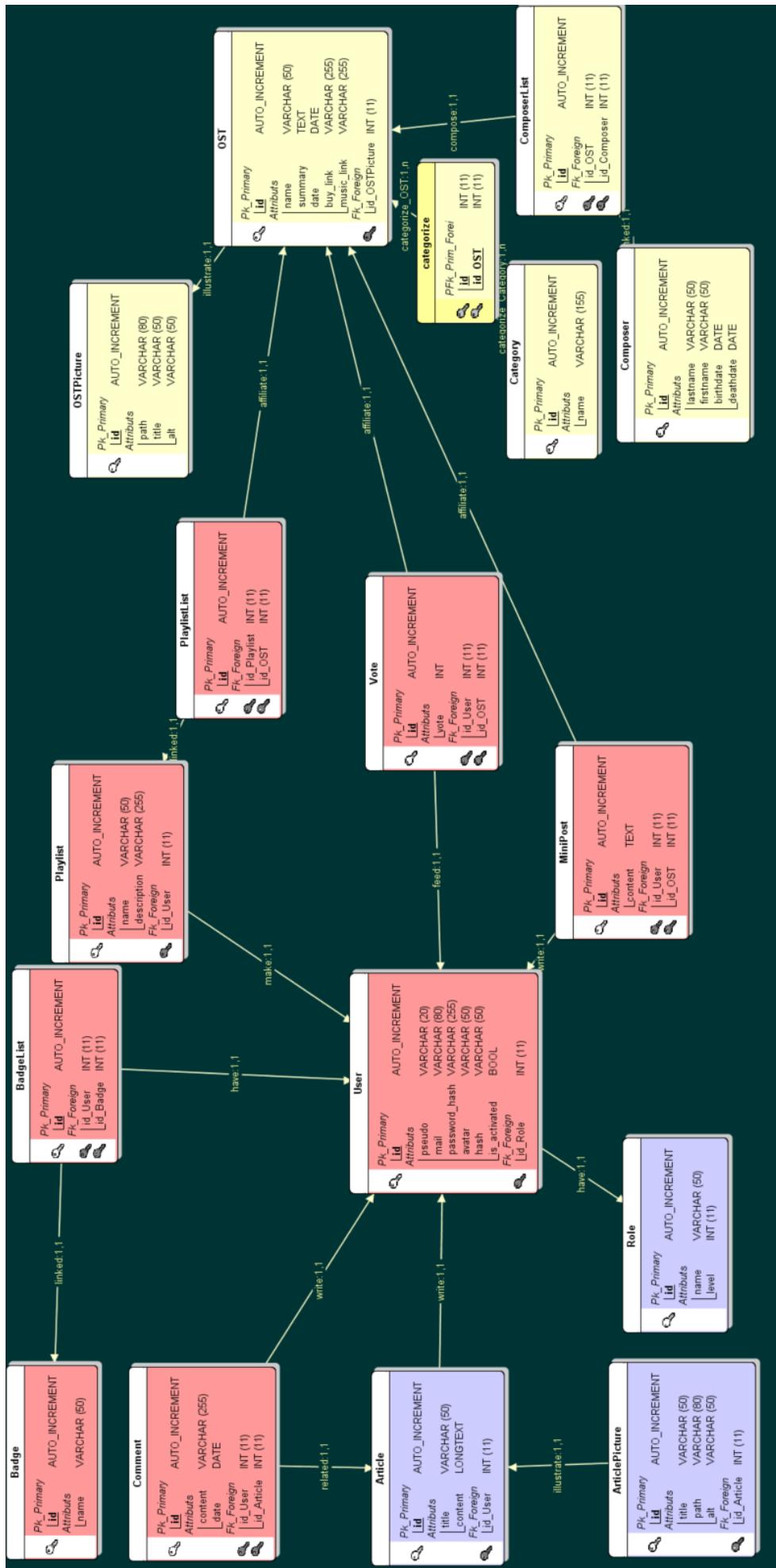
UTILISATEUR

UTILISATEUR/ADMINISTRATEUR

MUSIQUES

LE MCD (MODÈLE CONCEPTUEL DES DONNÉES) EST UNE PRÉSENTATION GRAPHIQUE QUI PERMET FACILEMENT ET SIMPLEMENT DE COMPRENDRE COMMENT LES DIFFÉRENTS ÉLÉMENTS SONT LIÉS ENTRE EUX À L'AIDE DE DIAGRAMMES CODIFIÉS.





LE MLD OU MODÈLE LOGIQUE DES DONNÉES EST
SIMPLEMENT LA PRÉSENTATION TEXTUELLE DU MCD. IL
S'AGIT JUSTE DE LA PRÉSENTATION EN LIGNE DU SCHÉMA
PRÉSENTANT LA STRUCTURE DE LA BASE DE DONNÉES.

Génération du Code SQL



VOICI UN APERÇU DU CODE SQL GÉNÉRÉ PAR JMERISE

```

#-----#
# Script MySQL.
#-----#
#-----#
# Table: Role
#-----#
CREATE TABLE Role(
    id Int Auto_increment NOT NULL ,
    name Varchar (50) NOT NULL ,
    level Int NOT NULL ,
    CONSTRAINT Role_PK PRIMARY KEY (id)
)ENGINE=InnoDB;

#-----#
# Table: User
#-----#
CREATE TABLE User(
    id Int Auto_increment NOT NULL ,
    pseudo Varchar (20) NOT NULL ,
    mail Varchar (80) NOT NULL ,
    password_hash Varchar (255) NOT NULL ,
    avatar Varchar (50) ,
    hash Varchar (50) ,
    is_activated Bool NOT NULL ,
    id_Role Int NOT NULL ,
    CONSTRAINT User_PK PRIMARY KEY (id)
)ENGINE=InnoDB;

#-----#
# Table: Role
#-----#
CREATE TABLE Role(
    id Int Auto_increment NOT NULL ,
    name Varchar (50) NOT NULL ,
    CONSTRAINT Role_PK PRIMARY KEY (id)
)ENGINE=InnoDB;

#-----#
# Table: Playlist
#-----#
CREATE TABLE Playlist(
    id Int Auto_increment NOT NULL ,
    name Varchar (50) NOT NULL ,
    description Varchar (255) ,
    id_User Int NOT NULL ,
    CONSTRAINT Playlist_PK PRIMARY KEY (id)
    CONSTRAINT Playlist_User_FK FOREIGN KEY (id_User) REFERENCES User(id)
)ENGINE=InnoDB;

#-----#
# Table: Article
#-----#
CREATE TABLE Article(
    id Int Auto_increment NOT NULL ,
    title Varchar (50) NOT NULL ,
    content Longtext NOT NULL ,
    id_User Int NOT NULL ,
    CONSTRAINT Article_Pk PRIMARY KEY (id)
)ENGINE=InnoDB;

#-----#
# Table: Comment
#-----#
CREATE TABLE Comment(
    id Int Auto_increment NOT NULL ,
    content Varchar (255) NOT NULL ,
    date Date NOT NULL ,
    id_User Int NOT NULL ,
    id_Article Int NOT NULL ,
    CONSTRAINT Comment_Pk PRIMARY KEY (id)
)ENGINE=InnoDB;

#-----#
# Table: Composer
#-----#
CREATE TABLE Composer(
    id Int Auto_increment NOT NULL ,
    lastname Varchar (50) NOT NULL ,
    firstname Varchar (50) NOT NULL ,
    birthdate Date NOT NULL ,
    deathdate Date ,
    CONSTRAINT Composer_pk PRIMARY KEY (id)
)ENGINE=InnoDB;

#-----#
# Table: Category
#-----#
CREATE TABLE Category(
    id Int Auto_increment NOT NULL ,
    name Varchar (155) NOT NULL ,
    CONSTRAINT Category_pk PRIMARY KEY (id)
)ENGINE=InnoDB;

#-----#
# Table: Badge
#-----#
CREATE TABLE Badge(
    id Int Auto_increment NOT NULL ,
    name Varchar (50) NOT NULL ,
    CONSTRAINT Badge_pk PRIMARY KEY (id)
)ENGINE=InnoDB;

#-----#
# Table: OSTpicture
#-----#
CREATE TABLE OSTpicture(
    id Int Auto_increment NOT NULL ,
    path Varchar (80) NOT NULL ,
    title Varchar (50) NOT NULL ,
    alt Varchar (50) NOT NULL ,
    CONSTRAINT OSTpicture_pk PRIMARY KEY (id)
)ENGINE=InnoDB;

```

Base de Données



VOICI UN APERÇU, DEPUIS PHPMYADMIN, DE LA BASE DE DONNÉES, PRÉCÉDEMMENT VUE. ELLE CONTIENT DONC 17 TABLES.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
article	Parcourir	3	InnoDB	utf8mb4_general_ci	32,0 kio	-
articlepicture	Parcourir	4	InnoDB	utf8mb4_general_ci	32,0 kio	-
badge	Parcourir	0	InnoDB	utf8mb4_general_ci	16,0 kio	-
badgelist	Parcourir	0	InnoDB	utf8mb4_general_ci	48,0 kio	-
categorizedby	Parcourir	40	InnoDB	utf8mb4_general_ci	32,0 kio	-
category	Parcourir	12	InnoDB	utf8mb4_general_ci	16,0 kio	-
comment	Parcourir	2	InnoDB	utf8mb4_general_ci	48,0 kio	-
composer	Parcourir	62	InnoDB	utf8mb4_general_ci	16,0 kio	-
composertlist	Parcourir	38	InnoDB	utf8mb4_general_ci	48,0 kio	-
minipost	Parcourir	1	InnoDB	utf8mb4_general_ci	48,0 kio	-
ost	Parcourir	41	InnoDB	utf8mb4_general_ci	32,0 kio	-
ostpicture	Parcourir	41	InnoDB	utf8mb4_general_ci	16,0 kio	-
playlist	Parcourir	0	InnoDB	utf8mb4_general_ci	32,0 kio	-
playlistlist	Parcourir	0	InnoDB	utf8mb4_general_ci	48,0 kio	-
role	Parcourir	3	InnoDB	utf8mb4_general_ci	16,0 kio	-
user	Parcourir	6	InnoDB	utf8mb4_general_ci	32,0 kio	-
vote	Parcourir	0	InnoDB	utf8mb4_general_ci	48,0 kio	-
17 tables		253	MyISAM	utf8mb4_general_ci	560,0 kio	0 o
Avec la sélection :						
<input type="checkbox"/> Tout cocher						
<input type="checkbox"/> Imprimer	Dictionnaire de données					
<input type="checkbox"/> Console de requêtes SQL						

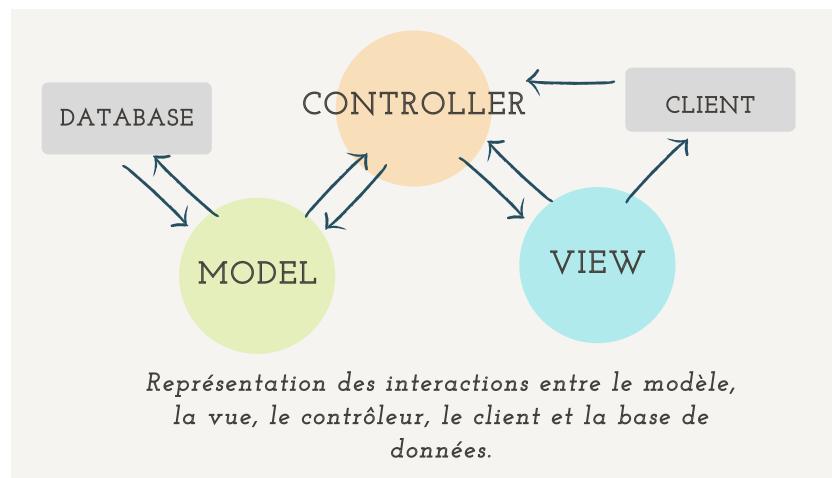
MVC, PDO & CRUD

Pour réaliser mon site, j'ai utilisé l'architecture **MVC**.

M pour "**Modèle**" : il s'agit des données et de l'état d'une application web, représentées par un ensemble de classes, qui interagissent avec la base de données.

V pour "**Vue**" : il s'agit de la présentation de l'interface graphique

C pour "**Contrôleur**" : il fait le lien entre le modèle et la vue, et représente particulièrement la logique concernant les actions effectuées par l'utilisateur.



PDO

Pour accéder à ma base de données, et communiquer avec cette dernière, j'ai utilisé l'extension de PHP, "**PDO**" (PHP Date Objects), qui est une interface orientée objet.

Pour me connecter à ma base de données, j'ai créé un fichier **mainModel.php** contenant:

- la classe **mainModel**
- les attributs **\$pdo** et **\$table** qui seront disponible dans la classe, et les **enfants** de la classe
- la méthode magique **__construct()** contient la nouvelle instance de PDO, le **try** (information de connexion) et le **catch** (messages d'erreur).

```

<?php
/**
 * Modèle principal
 */
class MainModel
{
    //Les attributs $pdo et $table seront disponibles dans les enfants
    public $pdo = null;
    public $table = null;

    public function __construct()
    {
        try {
            // On se connecte à MySQL pour faire le lien avec la BDD
            $this->pdo = new PDO('mysql:host=' . DB_HOST . ';dbname=' . DB_NAME . ';charset=utf8', DB_USER, DB_PASSWORD);
            // En cas d'erreur, on affiche un message et on arrête tout
        } catch (Exception $e) {
            die('Erreur : ' . $e->getMessage());
        }
    }
}

```



Ici, j'ai mis en place un "getter" et un "setter". Ces deux méthodes magiques permettent d'accéder aux attributs de la classe (getter), et de modifier ses attributs (setter).

```
/*
 * Getter permettant d'avoir accès à tous les attributs de la classe
 *
 * @param string $property
 * @return mixed
 */
public function __get($property)
{
    if (property_exists($this, $property)) {
        if ($property != 'pdo') {
            return $this->$property;
        } else {
            return false;
        }
    } else {
        return false;
    }
}

/*
 * Setter permettant de modifier les attributs de la classe
 *
 * @param string $property
 * @param mixed $value
 * @return boolean
 */
public function __set($property, $value)
{
    if (property_exists($this, $property)) {
        if ($property != 'pdo') {
            $this->$property = $value;
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
}
```

Cette classe a été créée dans le but de gérer tous les **formulaires** de mon site. L'idée générale étant de **regrouper** au sein d'**un même objet**, des **méthodes** qui seront appelés plusieurs fois.

```
<?php
/**
 * Objet permettant de gerer les formulaires
 */
class Form
{
    public $error = [];
    const PATTERN = '/^[_a-zA-Z0-9-]{3,20}$/'; //on contrôle que le pseudo contient 3 à 20 caractères

    /**
     * Vérification que le champ n'est pas vide et qu'il existe
     * @param string $fieldName
     * @param string $fieldValue
     * @return boolean
     */
    public function isNotEmpty($fieldName, $fieldValue)
    {
        if (!empty($fieldValue)) {
            return true;
        } else {
            $this->error[$fieldName] = 'Le champ ' . $fieldName . ' est vide';
        }
    }

    /**
     * Vérification que le champ pseudo correspond à la regex
     * @param string $fieldName
     * @param string $fieldValue
     * @return boolean
     */
    public function isValidFormat($fieldName, $fieldValue, $format = SELF::PATTERN)
    {
        if (preg_match($format, $fieldValue)) {
            return true;
        } else {
            $this->error[$fieldName] = 'Le champ ' . $fieldName . ' n\'est pas au bon format. Il doit contenir au moins 3 caractères, sans aucun espace et aucun caractères spéciaux';
        }
    }

    /**
     * Vérification de l'unicité d'un champ
     * @param string $fieldName
     * @param string $fieldValue
     * @param string $className
     * @return mixed
     */
    public function isUnique($fieldName, $fieldValue, $className)
    {
        $class = new $className();
        if (method_exists($class, 'isUnique')) {
            if ($class->isUnique($fieldName, $fieldValue)) {
                return true;
            } else {
                $this->error[$fieldName] = 'Le champ ' . $fieldName . ' est déjà utilisé';
                return false;
            }
        } else {
            return null;
        }
    }
}
```

```

/**
 * Vérification de l'existence d'une valeur
 *
 * @param [type] $fieldName
 * @param [type] $FieldValue
 * @param [type] $className
 * @return void
 */
public function doesExist($fieldName, $FieldValue, $className)
{
    $class = new $className();
    if (method_exists($class, 'doesExist')) {
        if ($class->doesExist($fieldName, $FieldValue)) {
            return true;
        } else {
            $this->error[$fieldName] = 'La valeur ' . $fieldName . ' n\'existe pas';
            return false;
        }
    } else {
        return null;
    }
}

/**
 * Vérification de la taille d'un champ
 * @param string $fieldName
 * @param string $FieldValue
 * @param int $min
 * @param int $max
 * @return boolean
 */
public function isValidLength($fieldName, $FieldValue, $min, $max)
{
    $length = strlen($FieldValue);
    if ($length >= $min && $length <= $max) {
        return true;
    } else {
        $this->error[$fieldName] = 'La taille du champ ' . $fieldName . ' doit être comprise entre ' . $min . ' et ' . $max;
        return false;
    }
}
/**
 * Vérification de la validité d'un email
 *
 * @param string $fieldName
 * @param string $FieldValue
 * @return boolean
 */
public function isValidEmail($fieldName, $FieldValue)
{
    if (filter_var($FieldValue, FILTER_VALIDATE_EMAIL)) {
        return true;
    } else {
        $this->error[$fieldName] = 'Le champ ' . $fieldName . ' n\'est pas une adresse mail valide';
        return false;
    }
}

/**
 * Vérification de la validité du formulaire
 *
 * @return boolean
 */
public function isValid()
{
    return empty($this->error);
}

```

Je vais maintenant vous présenter les **4 opérations** du CRUD:

- **Create**: créer
- **Read**: lire
- **Update**: modifier
- **Delete**: supprimer

Ces 4 opérations seront illustrées au sein d'un exemple concret: une **fonctionnalité utilisateur**, qui est le **Mini-Post**.

Avant toute chose, voici une rapide explication de ce qu'est le mini-post. Il s'agit d'un texte, ayant pour vocation d'être une **analyse/critique** ou tout autre **ressenti** sur une musique en particulier. La musique est recherchée via une barre de recherche. Ce texte apparaît publiquement sur les pages des catégories et est disponible pour l'auteur du mini-post, dans un menu en particulier.

CREATE

→ MODEL minipost.php

Ici, je crée la méthode **createMiniPost()**. J'utilise la commande SQL 'INSERT INTO', en indiquant le nom de la table ('**minipost**'), ainsi que le nom des **champs** dans lesquels je souhaite que les **valeurs** soient insérées. Puis les valeurs dans la commande 'VALUES', sous forme de **marqueurs nominatifs**. Il est important de construire cette méthode en utilisant une **requête préparée** et d'utiliser des marqueurs nominatifs, pour éviter les **injections SQL**. A cette requête préparée est associé le '**bindValue**', qui permet d'associer un paramètre à une valeur, puis le type de donnée.

Dans l'ordre nous avons donc:

le paramètre: '**:content**'/ la valeur: **\$this->content** / le type: **PDO::PARAM_STR**

```
* Methode permettant de créer un mini-post
*
* @return boolean
*/
public function createMiniPost()
{
    $pdostatement = $this->pdo->prepare(
        'INSERT INTO `minipost`(`content`, `id_User`, `id_OST`)
        VALUES(:content, :id_User, :id_OST)'
    );
    $pdostatement->bindValue(':content', $this->content, PDO::PARAM_STR);
    $pdostatement->bindValue(':id_User', $this->id_User, PDO::PARAM_STR);
    $pdostatement->bindValue(':id_OST', $this->id_OST, PDO::PARAM_STR);
    $pdostatement->execute();
    return $this->pdo->lastInsertId();
}
```

CREATE

→ CONTROLLER miniPostCreationCtrl.php

Pour faire le lien entre le modèle et la vue, on passe par le **contrôleur**.

Ici, nous faisons tous les **contrôles** nécessaires en appelant les méthodes vues dans la classe '**Form**'.

Pour cette exemple, nous vérifions que le contenu du mini-post, ainsi que la barre de recherche de musiques, ne sont pas vides ('**isNotEmpty()**').

Puis grâce au **setter** mis en place dans le **mainModel**, nous modifions les attributs de la classe.

Projet_fin_formation > controllers > 📄 miniPostCreationCtrl.php > ...

```
1  <?php
2  // Je charge ici les modèles et classes
3  require_once 'config.php';
4  require_once 'models/mainModel.php';
5  require_once 'models/miniPost.php';
6  require_once 'models/ost.php';
7  require_once 'classes/form.php';
8
9  $minipost = new MiniPost();
10 $minipostForm = new Form();
11 $message = '';
12
13 /**
14 * Vérifications du formulaire d'écriture de minipost
15 */
16 if (isset($_POST['submitMiniPost'])) {
17     $miniPostContent = '';
18     $resultSearch = '';
19     //Je récupère les données du formulaire
20     if (isset($_POST['miniPostContent'])) {
21         $miniPostContent = ($_POST['miniPostContent']);
22     }
23     if (isset($_POST['resultSearch'])) {
24         $resultSearch = htmlspecialchars($_POST['resultSearch']);
25     }
26     //Je vérifie le content du minipost
27     $minipostForm->isNotEmpty('miniPostContent', $miniPostContent);
28     $minipostForm->isNotEmpty('resultSearch', $resultSearch);
29     //Si il n'y a pas d'erreur sur le formulaire...
30     if ($minipostForm->isValid()) {
31         $minipost->__set('content', $miniPostContent);
32         $minipost->__set('id_OST', $resultSearch);
33         $minipost->__set('id_User', $_SESSION['user'][id']);
34         $minipost->createMiniPost();
35         echo $message = 'Mini-post bien publié!';
36         // redirection au bout de 3s sur la page des minipost
37         header('Refresh:3; url=../miniPost.php?' . $_GET['minipostID']);
38     } else {
39         echo $message = 'Une erreur a été identifiée.';
40     }
41 }
```

CREATE

→ VIEW miniPostCreation.php

Du point de vue **utilisateur**, un **formulaire** comportant une **barre de recherche** (crée en **AJAX**), qui génère un affichage automatique dans un '**select**', au moment où une lettre est tapée. Pour ce faire, j'utilise l'évènement du **DOM** (Document Object Model) **Javascript 'onkeyup'**.

L'affichage se fera dans un '**select**' qui lui même est présent dans une '**div**'.

En dessous se trouve la **zone d'écriture**, construite grâce à l'éditeur de HTML "**TinyMCE**".

Et enfin, un **bouton de validation** de formulaire, pour finaliser **la création** du mini-post.

```
<div class="container p-5 my-5 bg-dark">
<form method="POST">
  <div class="row mb-3">
    <div class="col-6">
      <input type="search" name="search" id="search" class="form-control mx-auto" placeholder="Search OST..." onkeyup="searchOst(this.value)">
    </div>
    <!-- affichage du résultat de la recherche-->
    <div class="col-6">
      <select class="form-select" id="resultSearch" name="resultSearch">
        </select>
    </div>
  </div>
  <div class="col-6 my-5">
    <textarea name="miniPostContent" id="miniPostContent"></textarea>
  </div>
  <!-- bouton de validation -->
  <button type="submit" name="submitMiniPost" class="btn btn-outline-light mt-2">Publier</button>
  <p class="text-lead"><?= $message ?></p>
</form>
</div>
<div class="d-flex justify-content-end m-5">
  <a class="btn btn-outline-secondary bi bi-pencil fs-5" href="miniPostList.php">
    Mes Mini-Post
  </a>
</div>
```

CREATE

→ VIEW

BARRE DE RECHERCHE

Cette barre de recherche a été créé en **AJAX**, afin d'afficher automatiquement le résultat. AJAX utilise l'objet **XMLHttpRequest**. J'utilise cet objet pour échanger des données avec le serveur, sans avoir à recharger la page. Pour l'affichage, j'utilise des **méthodes** du DOM, qui sont '**innerHTML**', '**createElement**', '**appendChild**', '**innerText**' et '**getElementById**'.

```
/**  
 * Barre de recherche Ajax  
 */  
function clearElement(elementId) {  
    document.getElementById(elementId).innerHTML = "";  
}  
  
function searchOst(searchContent) {  
    let xhr  
    xhr = new XMLHttpRequest()  
    xhr.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200) { //4: request finished and response is ready, 200: "OK"  
            //On commence par vider le contenu de la div parent  
            clearElement("resultSearch")  
            let selectParent = document.getElementById("resultSearch")  
            let element = JSON.parse(this.response)  
            //faire une boucle  
            for (let i = 0; i < element.length; i++) {  
                let optionChild = document.createElement('option')  
                optionChild.value = element[i].id  
                optionChild.innerText = element[i].ostName  
                selectParent.appendChild(optionChild)  
            }  
            console.log(this)  
        } else {  
            document.getElementById('resultSearch').innerHTML = '<option>OST non trouvée.</option>'  
        }  
    }  
    xhr.open('GET', '../controllers/ajaxCtrl.php?search=' + encodeURIComponent(searchContent), true)  
    xhr.send()  
}
```

Puis nous faisons appel à la méthode **searchOst()** qui contient les informations nécessaires à l'affichage de la musique.

```
<?php  
require_once '../config.php';  
require_once '../models/mainModel.php';  
require_once '../models/ost.php';  
$ost = new ost;  
  
/**  
 * Barre de recherche AJAX  
 */  
$searchOst = '';  
if (!empty($_GET['search'])) {  
    $searchOst = htmlspecialchars($_GET['search']);  
    $searchOst = trim($_GET['search']);  
    echo json_encode($ost->searchOst($searchOst));  
}
```

READ

→ MODEL minipost.php

Ici, je crée la méthode `getMiniPostList()` qui permet de **lister tous les mini-post** d'un utilisateur. Je mets en paramètre la variable `$idUser`, contenant la valeur de l'ID de l'utilisateur. Cette variable sera appelé dans le contrôleur.

Je crée des **jointures** avec les tables '`ostpicture`' et '`ost`' qui contiennent des informations relatives aux musiques. J'y sélectionne ce qui m'intéresse pour mon affichage (à savoir, l'image de l'album, ainsi que le titre de la musique).

Enfin, je retourne un tableau contenant toutes les lignes du jeu d'enregistrement (`fetchAll`).

```
/*
 * Méthode pour lister les mini-posts d'un utilisateur
 *
 * @return string
 */
public function getMiniPostList($idUser)
{
    // On récupère le contenu qui nous intéresse, de la table minipost
    $pdoStatement = $this->pdo->prepare(
        'SELECT `minipost`.`id`, `content`, `id_User`, `id_OST`, `path`, `alt`, `title`, `ost`.`name` AS `ostName`
        FROM `minipost`
        LEFT JOIN `ost`
        ON `minipost`.`id_OST` = `ost`.`id`
        LEFT JOIN `ostpicture` AS `op`
        ON `op`.`id` = `ost`.`id_OSTPicture`
        WHERE `id_User` = :idUser
        ORDER BY `id`'
    );
    $pdoStatement->bindValue(':idUser', $idUser, PDO::PARAM_STR);
    $pdoStatement->execute();
    // On retourne un tableau contenant toutes les lignes du jeu d'enregistrements. Le tableau représente chaque ligne comme soit un tableau de valeurs des colonnes,
    // FETCH_OBJ retourne un objet anonyme avec les noms de propriétés qui correspondent aux noms des colonnes retournés dans le jeu de résultats
    return $pdoStatement->fetchAll(PDO::FETCH_OBJ);
}
```

Ici, je crée la méthode `getMiniPostInfo()` qui permet de récupérer les informations d'**UN** mini-post. Cette fois, je retourne **une ligne** grâce au `fetch`.

```
/*
 * Méthode pour récupérer les informations d'un mini-post
 *
 * @return string
 */
public function getMiniPostInfo()
{
    $pdoStatement = $this->pdo->prepare(
        'SELECT `minipost`.`id`, `content`, `id_User`, `pseudo`, `op`.`id`, `path`, `alt`, `id_OST`, `title`, `ost`.`name` AS `ostName`
        FROM `minipost`
        LEFT JOIN `user`
        ON `minipost`.`id_User` = `user`.`id`
        LEFT JOIN `ost`
        ON `minipost`.`id_OST` = `ost`.`id`
        LEFT JOIN `ostpicture` AS `op`
        ON `op`.`id` = `ost`.`id_OSTPicture`
        WHERE `minipost`.`id` = :id'
    );
    $pdoStatement->bindValue(':id', $this->id, PDO::PARAM_INT);
    $pdoStatement->execute();
    // On retourne une ligne depuis un jeu de résultats associé à l'objet
    return $pdoStatement->fetch(PDO::FETCH_OBJ);
}
```

READ

→ CONTROLLER minipostCtrl.php

En passant la méthode `getMiniPostList()` à la variable `$minipost`, je mets en paramètre la **'superglobale'** `$_SESSION` qui contient l'**ID de l'utilisateur**. Cela me permet d'associer la liste des mini-posts, à l'utilisateur connecté et possédant lesdits mini-posts. Les informations relatives au mini-post sont ensuite attribuées à la variable `$minipostList`, qui nous servira à créer notre affichage dans la vue.

```
<?php
// On charge le fichier du modèle.
require_once 'models/mainModel.php';
require_once 'models/minipost.php';
require_once 'models/ost.php';

$minipost = new MiniPost();
$minipostList = $minipost->getMiniPostList($_SESSION['user']['id']);
```

Pour récupérer les informations d'un seul mini-post, j'**hydrate** la variable `$minipost` de l'attribut `id`, dans laquelle je stocke l'**ID** du minipost.

```
<?php
// On charge le fichier du modèle.
require_once 'models/mainModel.php';
require_once 'models/minipost.php';
require_once 'classes/form.php';

$minipostForm = new Form();
$minipost = new MiniPost();
/**
 * Récupération des informations du mini-post + pochette de l'OST et son nom
 */
$minipost->id = $_GET['minipostID'];
// on stocke dans une variable la fonction qui va appeler toutes les informations de l'article
$minipostInfo = $minipost->getMiniPostInfo();
```

READ

→ VIEW miniPostList.php

Pour l'affichage de tous les mini-posts, je fais une boucle **foreach** où je parcours le tableau présent dans la variable **\$minipostList**. Je mets mes données en forme grâce à des classes **Bootstrap** (card, d-flex etc...).

```
projet_fin_formation > miniPostList.php > ...
1  <?php
2  include 'parts/header.php';
3  require_once 'controllers/minipostListCtrl.php';
4  ?>
5  <div class="container p-2">
6      <div class="containerRise">
7          |   <h3 class="rise-text">Mes Mini-Posts</h3>
8      </div>
9      <?php
10     // On récupère nos variables de session
11     if (isset($_SESSION['user'])['isConnected'] && $_SESSION['user']['isConnected']) {
12     ?>
13         <span class="d-flex justify-content-end my-5">
14             |   <a href="miniPostCreation.php" class="btn btn-outline-secondary bi bi-pencil-square">Créer un mini-post</a>
15         </span>
16         <div class="row row-cols-sm-1 row-cols-2 row-cols-md-4 mt-5">
17             <?php
18             foreach ($minipostList as $value) {
19             ?>
20                 <div class="card text-center p-2 m-2" style="max-width: 300px;">
21                     |   <a href="miniPost.php?minipostID=<?= $value->id ?>" class="list-group-item list-group-item-action" aria-current="true" style="background: #05171e">
22                         |       alt ?>" title="<?= $value->title ?>" style="max-width: 300px;">
23                         |       <div class="card-body lead text-center text-light">
24                             |           <p><?= $value->postName ?></p>
25                         |       </div>
26                     </a>
27                 </div>
28             <?php
29             }
30             ?>
31         </div>
32     <?php
33     } else {
34     ?>
35         <p class="lead text-center text-danger">Il n'y a rien ici!</p>
36     <?php
37     }
38     ?>
39     </div>
40     <?php include 'parts/footer.php'; ?>
```

miniPost.php

Pour l'affichage d'un seul mini-post, j'utilise la variable **\$minipostInfo**, précédemment vue dans le contrôleur. Je l'hydrate avec les **attributs** demandés.

```
<?php
$controllers = 'controllers/minipostCtrl.php';
include 'parts/header.php';
?>
<div class="container-fluid p-5">
    <div class="card mb-3 bg-dark text-white p-2 mx-auto mb-5" style="width: 940px;">
        <div class="row">
            <div class="col-md-7">
                |   title ?>" alt="<?= $minipostInfo->alt ?>">
            </div>
            <div class="col-md-5">
                <div class="card-body">
                    |   <button type="button" class="offset-11 btn btn-sm btn-outline-secondary bi bi-pencil" data-bs-toggle="modal" data-bs-target="#updateElement"></button>
                    |   <h3 class="card-title"><?= $minipostInfo->postName ?></h3>
                    |   <p class="card-text"><?= $minipostInfo->content ?></p>
                </div>
            </div>
        </div>
    </div>
    <div class="col-auto d-flex justify-content-between">
        |   <a href="miniPostCreation.php" class="btn btn-outline-secondary bi bi-pencil-square"> Créer un mini-post</a>
        |   <a href="miniPostList.php" class="btn btn-outline-secondary bi bi-list"> Revenir à la liste des minipost</a>
        |   <button type="button" class="btn btn-outline-danger bi bi-x-circle" data-bs-toggle="modal" data-bs-target="#deleteElement"> Supprimer</button>
    </div>
</div>
```

UPDATE

→ MODEL minipost.php

Ici, je créé la méthode 'public' `updateMiniPost()` qui permet de modifier un mini-post, en utilisant la commande SQL 'UPDATE', puis 'SET' et enfin 'WHERE'.

```
/*
 * Méthode pour modifier un minipost
 *
 * @return void
 */
public function updateMiniPost()
{
    $pdoStatement = $this->pdo->prepare(
        'UPDATE `minipost`
        SET `content` = :content
        WHERE `id` = :id'
    );
    $pdoStatement->bindValue(':content', $this->content, PDO::PARAM_STR);
    $pdoStatement->bindValue(':id', $this->id, PDO::PARAM_INT);
    return $pdoStatement->execute();
}
```

UPDATE

→ CONTROLLER miniPostCtrl.php

Pour la **modification** du mini-post, je vérifie le formulaire '`updateMiniPost`', et plus particulièrement le '`textarea`' où sera écrit le contenu ('`updateContent`'). Si n'y a pas d'erreur, j'utilise de nouveau le **setter** pour mettre à jour l'**attribut**, et enfin je rafraîchis la page automatiquement avec la fonction `header('Refresh:0')`.

```
/*
 * Modification du minipost
 */
if (isset($_POST['updateMiniPost'])) {
    //Je récupère les données du formulaire
    if (isset($_POST['updateContent'])) {
        $updateContent = $_POST['updateContent'];
        //Je vérifie le contenu du minipost
        $minipostForm->isNotEmpty('content', $updateContent);
        //Si il n'y a pas d'erreur sur le formulaire...
        if ($minipostForm->isValid()) {
            $minipost->__set('content', $updateContent);
            $minipost->updateMiniPost();
            header('Refresh:0');
        }
    }
}
```

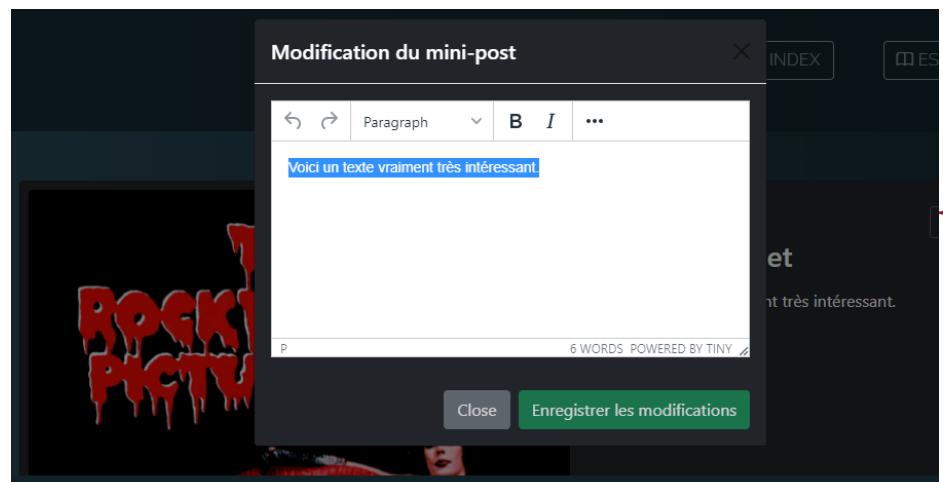
UPDATE

→ VIEW miniPost.php

La modification du mini-post dans ma vue, se fait dans une **fenêtre modale** créée avec **Bootstrap**. La modale est déclenchée au clic sur un bouton comportant une icône 

La modale contient une zone de texte ('**textarea**'), dont le contenu est le texte du mini-post (avant modification).

```
<!-- Modal Modification-->
<div class="modal fade" id="updateElement" tabindex="-1" aria-labelledby="updateElementLabel" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content bg-dark text-white">
      <div class="modal-header">
        <h5 class="modal-title" id="updateElementLabel">Modification du mini-post</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <form method="POST">
        <div class="modal-body">
          <textarea name="updateContent"><?= $minipostInfo->content ?></textarea>
        </div>
        <div class="modal-footer bg-dark text-white">
          <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button>
          <button type="submit" name="updateMiniPost" class="btn btn-success">Enregistrer les modifications</button>
        </div>
      </form>
    </div>
  </div>
</div>
```



DELETE

→ MODELE minipost.php

Pour la **suppression** du mini-post, je crée la méthode '`deleteMiniPost()`', qui récupère simplement l'**ID** du mini-post choisi.

```
/**  
 * Méthode pour supprimer un minipost  
 *  
 * @return void  
 */  
public function deleteMiniPost()  
{  
    $pdoStatement = $this->pdo->prepare(  
        'DELETE FROM `minipost`  
        WHERE `id` = :id'  
    );  
    $pdoStatement->bindValue(':id', $this->id, PDO::PARAM_INT);  
    $pdoStatement->execute();  
}
```

DELETE

→ CONTROLLER miniPostCtrl.php

Au niveau du **contrôleur**, je vérifie que l'**ID** du mini-post a bien été récupéré dans l'**URL**.

```
/**  
 * Suppression du minipost  
 */  
if (isset($_POST['delete'])) {  
    // on vérifie que l'ID du minipost a bien été récupéré dans l'URL  
    if (isset($_GET['minipostID'])) {  
        $delete = new MiniPost();  
        $delete->id = htmlspecialchars($_GET['minipostID']);  
        $deleteMinipost = $delete->deleteMiniPost();  
        // si tout est ok, on redirige vers la page de la liste des miniposts  
        header('Location: miniPostList.php');  
    }  
}
```

DELETE

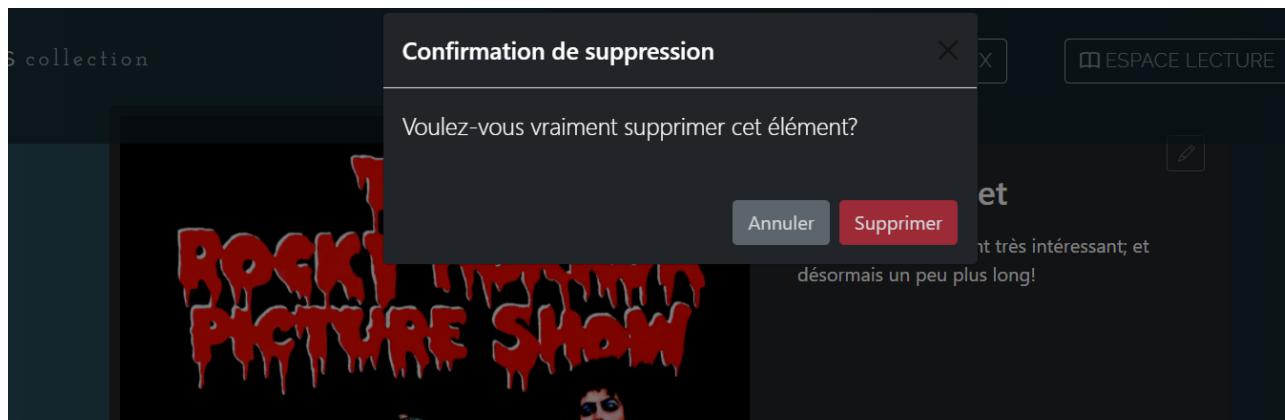
→ VIEW miniPost.php

Tout comme pour la modification, la suppression fait appel à une fenêtre modale pour confirmer l'action. Cela évite de supprimer malencontreusement un élément.

```
<!-- Modal Suppression-->


<div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="deleteElementLabel">Confirmation de suppression</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <div class="modal-body">
        <p class="lead">Voulez-vous vraiment supprimer cet élément?</p>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Annuler</button>
        <form method="POST">
          <button type="submit" name="delete" class="btn btn-danger">Supprimer</button>
        </form>
      </div>
    </div>
  </div>


```



Les failles de sécurité

PROTECTION CONTRE LES FAILLES XSS

Cette faille consiste à **injecter un script arbitraire** dans une **page** pour provoquer une action bien définie. Il en existe **deux sortes**:

XSS Permanent

C'est lorsque le script est stocké sur le **serveur externe** (base de données). Il est donc récupéré et exécuté à tous moments sur le site par n'importe quel utilisateur.

XSS Non Permanent

Le script est souvent intégré à une **URL** et est exécuté sans être stocké sur un serveur.

Pour m'en prémunir, j'utilise la fonction php **htmlspecialchars()** qui désactive les balises HTML. Il existe également la fonction **htmlentities()** qui filtre toutes les entités html.

PROTECTION CONTRE LES INJECTIONS SQL

L'injection SQL directe est une technique où un pirate **modifie une requête SQL existante** pour afficher des **données cachées**, ou pour **écraser des valeurs** importantes, ou encore **exécuter des commandes** dangereuses pour la base. Cela se fait lorsque l'application prend les **données envoyées par l'internaute**, et l'utilise directement pour construire une requête SQL. Une des plus connues: '**OR 1 = 1 #**'

Pour m'en prémunir, je ne stocke pas les mots de passe **en clair** dans ma base de données, ils sont "**hashé**" grâce à la fonction php: **password_hash()**. J'utilise également des **requêtes préparées**.

L'exécution d'une requête préparée se déroule en deux étapes : la **préparation** et l'**exécution**. Lors de la préparation, un **modèle de requête** est envoyé au serveur de base de données. Le serveur effectue une **vérification de la syntaxe**, et **initialise les ressources internes** du serveur pour une utilisation ultérieure.

AUTRES ATTAQUES

Concernant les attaques **CSRF**, j'ai mis en place un système de **confirmation de suppression** par fenêtre modale. Cela empêche l'utilisateur victime d'une attaque de ce type, de supprimer ses propres données sans le vouloir.

Pour les attaques **BruteForce**, il est prévu que j'intègre des **captcha**, ainsi qu'un **compteur d'essais** pour complexifier le mot de passe d'un utilisateur.



Traduction d'une Problématique



Durant l'élaboration de mon projet, j'ai été confronté à une problématique vis à vis de mes fenêtres modales de connexion et d'inscription. En effet, ces dernières se fermaient après soumission du formulaire, même lorsqu'une erreur était présente. J'ai donc recherché une aide sur StackOverflow. Voici la version originale + la traduction:

The screenshot shows a Stack Overflow page with the question "How to get Twitter Bootstrap modals to stay open on form submit?". The question was asked 9 years, 1 month ago and has been viewed 22k times. The user describes trying to figure out if there's a simple way to keep the modal open after a form submit. They mention that if the form is successful or there are errors, their PHP code closes the modal immediately. They prefer the user to see the message and close it themselves. A comment follows, offering to post code if helpful.

Comment faire en sorte que les modales de Bootstrap Twitter restent ouvertes après soumission d'un formulaire?

"J'essaie de savoir s'il y a une manière relativement simple (n'étant pas très doué avec JQuery) de garder la modale ouverte après la soumission d'un formulaire (la modale contient le formulaire).

Si le formulaire est valide, ou s'il y a une erreur, mon PHP les fera apparaître, mais la modale se ferme dès que le bouton de soumission est déclenché. Si je recharge le formulaire, je peux voir les messages de succès ou d'erreur, donc tout cela fonctionne bien, mais je préférerais que l'utilisateur voit le message, puis clique ensuite pour fermer la fenêtre. Je peux poster mon code si cela aide. Merci."

"Quand votre formulaire est soumis, la page se recharge, même si l'action (attribut) du formulaire est la même que la page (l'action laissée vide renvoie à la même page également). Je pense que vous voulez ré-ouvrir la modale, une fois que la page charge à nouveau. J'imagine que vous utilisez method="post", donc votre code devrait être quelque chose comme cela: "

The screenshot shows a response to the question. It explains that when a form is submitted, the page reloads, even if the action is the same page (empty action means the same page). The user wants to re-open the modal after a reload. The answer provides sample code using jQuery to handle the modal's visibility based on the form submission status.

```
<html>
    <head>
        <!-- stuff -->
        <script type="text/javascript">

<?php if(isset($_POST['submit_button'])) { ?> /* Your (php) way of checking that th

        $(function() {
            $('#myModal').modal('show'); // Show the modal
        });

<?php } ?>                                /* /form has been submitted */

        </script>
    </head>
    <body>
        <!-- etc -->
    </body>
</html>
```

AMÉLIORER

En premier lieu, je souhaite améliorer la version actuelle de mon site qui n'est que la **version 1**. J'aimerais rajouter les **fonctionnalités** que je n'ai pas pu planter, comme notamment l'ajout de **badges**, la création de **playlists** et la possibilité de **voter**.

J'aimerais améliorer le **dashboard de l'Administrateur**, et également mettre en place le rôle de **Modérateur**, et laisser la possibilité aux utilisateurs de pouvoir écrire des **articles** (fonctionnalité qui est pour le moment réservée à l'administrateur). J'ai également prévu et commence à rédiger d'autres articles. En voici les titres de certains d'entre eux:

"Les duos iconiques", "La place des femmes dans la musique de film", "Panorama des supports insolites" ... etc.

Concernant l'**aspect visuel**, j'aimerais apporter des modifications et améliorations; afin de rendre mon site le plus attrayant possible.

ÉTENDRE

Pour le moment "Orpheus Collection" reste un site relativement "de niche". J'aimerais étendre le concept à d'autres domaines, comme les **Beaux Arts**, la **Photographie**, le **Jeu Vidéo**, la **Littérature**... Beaucoup de domaines sont exploitables, et beaucoup d'oeuvres procurent un **engouement** et des **émotions** très importantes pour beaucoup d'entre nous.

Toutes ces œuvres **passionnantes** n'attendent qu'à être (re)découvertes par des utilisateurs **passionnés**.

Par ailleurs, ayant mis en place des liens de redirection pour acheter les divers albums de musiques de films, il m'apparaît intéressant d'envisager, à l'avenir, des **partenariats (via des liens sponsorisés)** avec les sites commerçants que je mets en avant sur mon site.

ÉVOLUER

Ayant pour projet de poursuivre mes études après cet examen, et plus particulièrement, de préparer un diplôme en tant que **Conceptrice et Développeuse d'Application**; j'envisage de faire évoluer "Orpheus Collection" en une application mobile. Cela me semble particulièrement important de nos jours d'envisager le "**multiplateforme**".



Remerciements

Je souhaite remercier en premier lieu mes proches qui m'auront soutenu et inspiré tout le long de ce projet. Je remercie mes camarades de formation, la *Pl de Versailles*, pour leurs rires, leurs blagues et leur aide... et sans qui toutes ces lignes de code n'auraient été rien de plus qu'un dialecte incompréhensible et..."capillotracté".

Enfin, je remercie l'école *La Manu*, et toute son équipe pédagogique pour m'avoir laissé ma chance et pour m'avoir permis de concrétiser un projet dont je suis fière aujourd'hui.

Inès Bakhtaoui



