



Mudeo

Table des matières

Bienvenus	4
Couleurs	5
Logo	6
Fonctionnalités	8
Concurrence	9
Site Map	10
L'application	11
Config.php	11
Helpers	11
Les librairies	12
Le router	12
Le controller	13
Le database	13
Les contrôleurs	16
Controller Medias	16
Construct()	16
Index ()	16
Add()	17
delete (\$id)	18
AddToWatchList(\$id)	19
RemoveFromWatchList (\$id)	19

createArrayListLiked()	19
Controller Search	19
search()	20
viewsearch()	20
Controller Pages	20
Construct	20
Index()	21
show(\$id)	22
Controller Users	23
Construct	23
Index ()	23
register ()	23
login ()	24
member()	24
sendtoken()	25
reset (\$token)	26
createUserToken()	26
Controller Pages Errors	27
error ()	27
Models	27
CategoriesContainers.php	28
Entities.php	28
SQL Database	28
Mises à jour futures	30

Mudeo

Bienvenus

Créée en 2021, l'entreprise MUDEO vous propose une nouvelle façon de partager vos créations multimédias.

En quoi sommes-nous différents ?

Nous vous proposons une interface utilisateur où les contenus sont accessibles facilement et ne se perdent pas dans la masse.

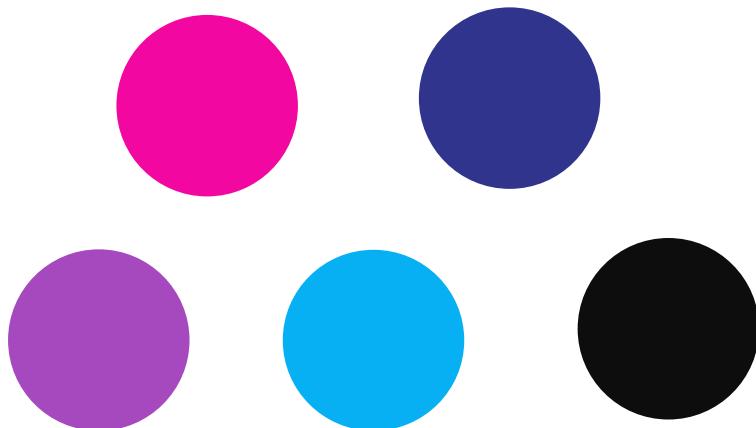
La fonction d'upload est limitée à 5 fichiers par personne. Les contenus chargés sont ainsi mieux sélectionnés. L'application est spécialement conçue pour les écoles de films et audio, mais elle pourra aussi être utilisée par un club des vidéastes amateurs ou une association de musiciens.

Nous voulons ainsi donner une fenêtre sur le monde aux étudiants afin que les recruteurs puissent facilement visionner leurs créations.



Couleurs

On a estimé qu'une seule couleur ne suffit pas. En plus, les dégradés dans la conception des applications resteront tendance en 2022. Les dégradés permettent aux concepteurs de transformer des logos simples en interprétations visuelles plus complexes. Les transitions d'une couleur à l'autre ou les différentes nuances d'une même couleur rendent le logo plus dynamique et hypnotique.



Mudeo

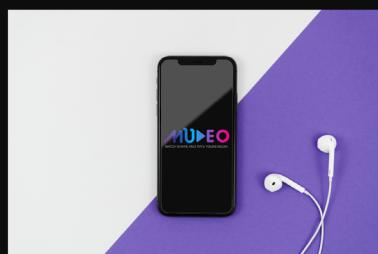
Logo

Dessiné par le designer Emanuele Mastaglia, le logotype veut mettre l'accent sur la possibilité de trouver des morceaux musicaux et des vidéos dans l'application. Au vu du caractère ludique du web site, l'icône s'intègre très bien à la charte graphique de l'entreprise.





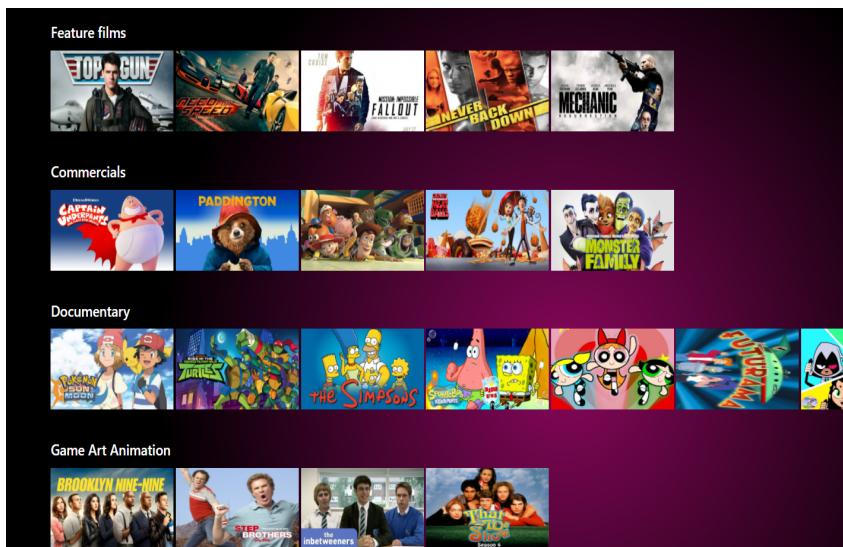
WATCH SHARE AND RATE YOURS MEDIA



LA POLICE « BC ALPHAPIEP » DYNAMIQUE ET LISIBLE
ELLE S'ADAPTE TRÈS BIEN AUX POLICES CHOISIES POUR
L'APPLICATION WEB.

Fonctionnalités

L'application permettra en premier lieu de visionner des files médias (vidéo et audio). Les utilisateurs pourront apprécier ou ne pas apprécier les contenus à l'aide d'un button. Les usagers auront droit d'uploader jusqu'à 5 fichiers. Une fois le fichier téléchargé, celui-ci pourra être modifié (description, cover image, titre et descriptions) ou effacé. Pour pouvoir utiliser l'application, les utilisateurs devront s'enregistrer.



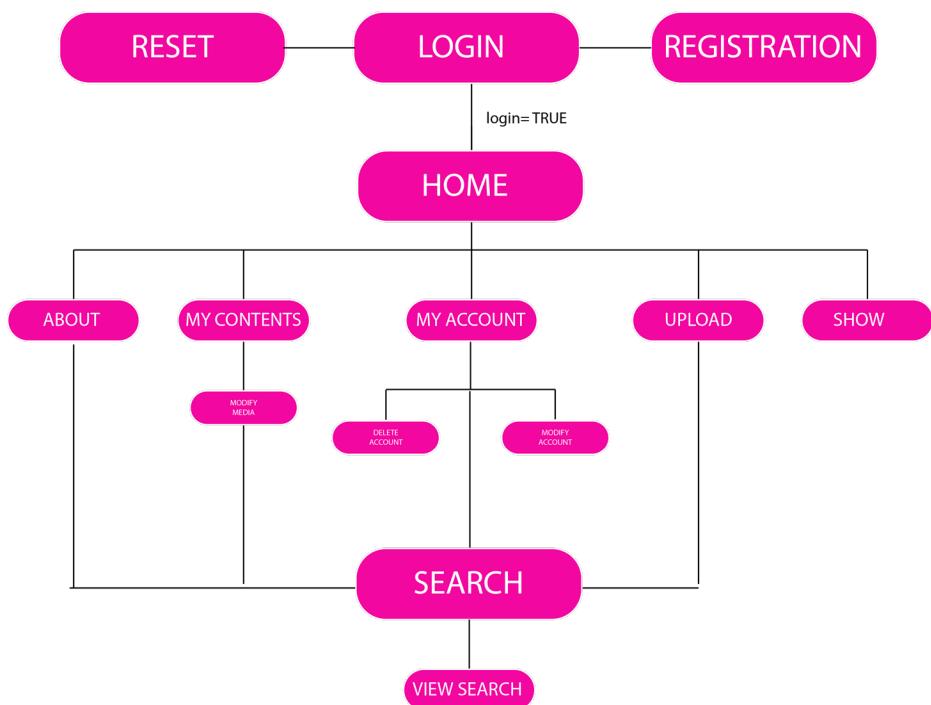
Concurrence

La concurrence existante est variée: Youtube, vimeo, Netflix, Disney+. Toutes ces applications comptent un grand nombre d'utilisateurs (200 millions pour Netflix). Par contre, les grandes plateformes de streaming (Netflix et Disney+) n'offrent pas la possibilité aux créateurs émergents d'avoir une place. Les plateformes comme YouTube et vimeo sont basés seulement sur l'upload. Tout le monde peut uploader, mais c'est très difficile d'avoir de la visibilité. L'interface graphique de YouTube est assez datée et de plus, si un recruter veut visionner le contenu, il doit aussi regarder la publicité. Ce que » Mudeo » propose est donc une plateforme avec du contenu réfléchi et accessible aux créateurs émergents dans une veste graphique fonctionnelle et moderne.



Mudeo

Site Map



L'application

L'application a été développée avec le design pattern MVC. L'architecture créée ainsi permet une meilleure maintenance du code.

Config.php

Le fichier config.php permet différents réglages. Il définit aussi les constantes pour les routes :

- pour l'application
- pour l'URL
- pour l'UPLOAD des fichiers

```
//DB params
define ('<>DB_HOST', «localhost»);
define ('<>DB_USER', «root»);
define ('<>DB_PASS», «»);
define ('DB_NAME', 'shareposts');
//App Root
define ('APPROOT', dirname(dirname(__FILE__)));
//Upload Root
define('UPLROOT', dirname(dirname(__FILE__))));
//URL Root Mac
```

Helpers

Le fichier se trouve dans app/helpers/session_helper. Il permet notamment de créer un message flash pour un retour utilisateur.

Il vérifie si un utilisateur est connecté ou pas afin d'activer ou désactiver les différents liens du site et bloquer l'accès aux utilisateurs que ne sont pas enregistrés.

```
function isLoggedIn(){
    if (! $_SESSION[ «logged»]) {
        return false;
    }else{
        return true;
    }
}
```

Le fichier vérifie aussi si un contenu multimédia a été aimé ou ajouté à la « watch list » de l'utilisateur.

```
function isAdded($id){  
    if (in_array($id, $_SESSION['array_watch'])) {  
        return true;  
    }else{  
        return false;  
    }  
  
}  
  
function isLiked($id){  
    if (in_array($id, $_SESSION['array_like'])) {  
        return true;  
    }else{  
        return false;  
    }  
}
```

Les librairies

Le router

Le router est défini dans la classe Core.php. Le fichier bootstrap.php qui est « require_once » dans l'index.php du dossier public permet l'inclusion automatique de toutes les autres classes dont le router. Le router récupère l'URL depuis la variable. `$_GET['URL']`

L'URL est sécurisée avec `FILTER_SANITIZE_URL`

L'URL est ensuite décomposée en/contrôleur/method/params, le controller par défaut est *Pages*, la méthode par défaut est « *index* et les paramètres restent vident sauf s'ils sont demandés dans la méthode. Après avoir vérifié le paramètre provenant de l'URL un call back s'effectue sur celui-ci avec :

```
call_user_func_array([$this->currentController, $this->currentMethod], $this->params);
```

Le controller

Le contrôleur père, qui sera étendu à tous les autres controller se trouve en app/librairies/Controller.php. Il contient deux fonctions publiques. La première permet de faire appel à un modèle avec la simple fonction:

```
$this→entitiesModel = $this→model('Entities');
```

Il est structuré ainsi:

```
public function model($model){  
    //Require Model  
    require_once '../app/models/' . $model . '.php';  
    //Instaciated model  
    return new $model();  
}
```

La deuxième fonction est celle qui permet de faire appel au view et transmettre des données:

```
$this→view(<>pages/index>, $ data);
```

Il est structuré ainsi:

```
//Render view  
public function view($view, $data=[]){  
    //check for view  
    if(file_exists('../app/views/' . $view . '.php')){  
        require_once '../app/views/' . $view . '.php';  
    }else{  
        //view not exist  
        http_response_code('404');  
        require_once '../app/views/pages/404.php';  
    }  
}
```

Le database

Le database père se trouve en app/librairies/Database.php et il gère toutes les requêtes SQL. Il est possible dans chaque modèle de créer un

Mudeo

nouveau database en utilisant toutes les méthodes propres à celui-ci.

```
$this->db = new Database;

<?php

class Database{
    private $host= DB_HOST;
    private $user= DB_USER;
    private $pass= DB_PASS;
    private $dbname= DB_NAME;

    private $dbh;
    private $stmt;
    private $error;

    public function __construct()
    {
        //Set DSN
        $dsn= 'mysql:host=' . $this->host . ';dbname=' . $this->dbname;
        $options= array(
            PDO::ATTR_PERSISTENT=> true,
            PDO::ATTR_ERRMODE=> PDO::ERRMODE_EXCEPTION,
        );

        //Create PDO INSTANCE
        try{
            $this->dbh = new PDO($dsn,$this->user,$this->pass,$options);
        }catch (PDOException $e){
            $this->error = $e->getMessage();
            echo $this->error;
        }
    }

    //Prepare statement with query
    public function query($sql){
        $this->stmt= $this->dbh->prepare($sql);
    }

    //Bind values
    public function bind($param,$value, $type = null){
        if(is_null($type)){
            switch (true){
                case is_int($value):
                    $type = PDO::PARAM_INT;
                    break;
                case is_bool($value):
                    $type = PDO::PARAM_BOOL;
                    break;
                case is_null($value):
                    $type = PDO::PARAM_NULL;
                    break;
                default:
                    $type = PDO::PARAM_STR;
            }
        }
        $this->stmt->bindValue($param,$value,$type);
    }

    //Execute the prepare statement
    public function execute(){
        return $this->stmt->execute();
    }
}
```

```
//Get result as array of objects
public function resultSet(){
    $this->execute();
    return $this->stmt->fetchAll(PDO::FETCH_OBJ);
}

//Single Row
public function single(){
    $this->execute();
    return $this->stmt->fetch(PDO::FETCH_OBJ);
}

//Single Row
public function singleAss(){
    $this->execute();
    return $this->stmt->fetch(PDO::FETCH_ASSOC);
}

//Get row count
public function rowCount(){
    return $this->stmt->rowCount();
}
```

Les contrôleurs

Tous les contrôleurs sont des extensions du fichier qui se trouve dans app/libraries/Controller.php, comme susmentionné. Quatre contrôleurs ont été définis: *Medias.php*, *PageError.php*, *Pages.php* et *Users.php*.

Controller Medias

Le controller média contient différentes méthodes. Nous allons souligner dans cette section les fonctionnalités principales de chaque méthode.

Construct()

Le *construct()* vérifie si l'utilisateur est connecté ou pas. Dans le cas où il n'est pas connecté, il sera automatiquement redirigé vers la page de login. Ensuite un appel au modèle entities est effectué. Les médias qui ont été aimés ou ajoutés à la watch list, sont récupérés automatiquement.

Avec les fonctions suivantes:

```
$this->entitiesModel = $this->model('Entities');
$this->createArrayListLiked();
$this->createArrayList();
```

Index ()

Les fichiers uploadés et la «watch list» (s'il y en a) sont récupérés et

transmis dans la view (médias/index) via la variable \$data.

Add()

La méthode *add()* permet d'uploader jusqu'à 5 fichiers sur l'application web. La fonction contrôle d'abord si les champs du formulaire sont remplis correctement. La taille maximale pour l'image de couverture est de 3 Mb, si l'image est trop grande elle sera redimensionnée à l'aide des fonctions natives de PHP. Les fichiers multimédias ont une taille maximale de 1gb (la taille reste à définir à la suite d'une étude auprès des utilisateurs), si aucune erreur n'est détectée, c'est la librairie FFMPEG qui lance son script.

FFMPEG

La librairie FFMPEG a été choisie, car elle est très souple et permet de choisir parmi de nombreuses options. Dans notre cas elle comprime et formate le fichier vidéo en mp4 avec une taille de 1920x1080. Elle lance aussi un deuxième script qui permet d'extraire un preview vidéo de 15 secondes pour l'afficher en homepage. Lors de l'ajout d'un fichier audio un fichier.mp 4 de la vague sonore est générée.

```
//compress and resize video
$cmd = "$ ff -i $file_temp -s 1920:1080 $final_directory";
//create preview
$cmd_prew = "$ ff -i $file_temp -vcodec copy -ss 00:00:05 -t 00:00:05 $final_directory_preview";
system($cmd);
system($cmd_prew);

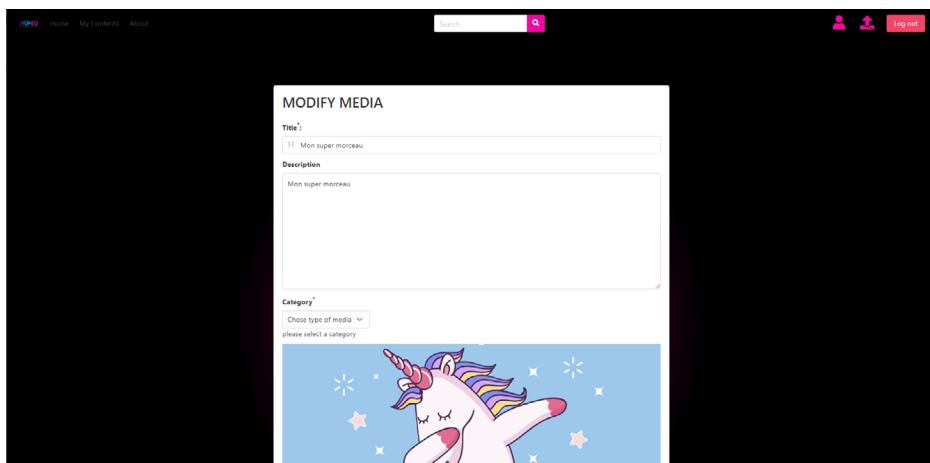
//creat waveform video
$filter = "[0:a]showwaves=colors=blueviolet:s=1280x720:mode=cline, format=yuv420p [v]";
$map = "[v]";
$cmd = "$ff -i $file_temp -filter_complex $filter $map -map 0:a -c:v libx264 -c:a copy
$final_directory";
system($cmd);
```

delete (\$id)

La méthode `delete($id)` permet d'effacer les médias uploadés par l'utilisateur. Un paramètre \$id est passé. Si la méthode est en POST et si le média appartient bien au propriétaire du compte, elle efface le fichier sur le serveur et aussi dans la base de données.

modify (\$id)

La méthode `modify($id)` permet de modifier les médias uploadés par l'utilisateur. Un paramètre \$id est passé. Si la méthode est en POST et si le média appartient bien au propriétaire du compte, une page avec un formulaire de modification s'ouvrira. Il n'est cependant pas possible de modifier le fichier vidéo, la meilleure méthode pour le faire consiste à effacer le média et commencer un nouvel upload avec la méthode `add()`.



AddToWatchList(\$id)

La méthode `addToWatchList($id)` permet d'ajouter des fichiers à sa watch list. Un paramètre `$id` est passé. Elle s'active avec une requête en POST avec la méthode en fetch API. Si le média est déjà dans la watch list il ne sera pas possible de l'ajouter à double

RemoveFromWatchList (\$id).

La méthode `removefromWatchList($id)` permet de supprimer des fichiers de sa watch list. Un paramètre `$id` est passé.

createArrayListLiked()

Elle ajoute dans un array les médias qui ont été aimés par l'utilisateur. Cette fonction protégée se trouve également dans `Pages.php`

createArrayList()

Elle ajoute dans un array les médias qui ont été ajoutés par l'utilisateur. Cette fonction protégée se trouve également dans `Pages.php` (car c'est aussi dans le home qu'on peut ajouter de contenu)

Controller Search

Le controller `search` contient différentes méthodes. Nous allons souligner dans cette section les fonctionnalités principales de chaque méthode.

Mudeo

search()

Elle permet de chercher avec une requête SQL un terme donné. Si aucun terme de recherche n'est saisi, la recherche ne se lance pas. Cette méthode fonctionne en Fetch API.

viewsearch()

Elle affiche le résultat de recherche depuis la méthode search ()

Controller Pages

Le contrôleur pages contient différentes méthodes. Nous allons souligner dans cette section les fonctionnalités principales de chaque méthode.

Construct

Le *construct()* vérifie si l'utilisateur est connecté ou pas. Dans le cas où il n'est pas connecté, il sera automatiquement redirigé vers la page de login. Ensuite un appel aux modèles "entities" aux "categoriesentities" est effectué. Les médias qui ont été aimés ou ajoutés à la watch list sont récupérés automatiquement avec les fonctions :

```
$this->entitiesModel = $this->model('Entities');
$this->categoriesContainers = $this->model('categoriesContainers');
$this->createArrayListLiked();
$this->createArrayList();
```

Index()

La méthode *index()* permet de récupères le donnés relatives aux médias qui se trouvent dans la base de données. Ces données sont ensuite transmises dans la *view (pages/index)* via la variable *\$data*. Les fichiers sont également divisés par catégories en faisant appel au modèle *categoriesContainer* et aux modèles *entities* avec la fonction *getCategories()* (*app/models/categoriesContainer*):

```
public function getCategories(){
    $this->db->query("SELECT categories.id,
                        categories.name AS mainCategories,
                        entities.categoryId
                     FROM categories
                     INNER JOIN entities
                     ON categories.id = entities.categoryId
                GROUP BY categories.name, categories.id, entities.categoryId");
    $results=$this->db->resultSet();
    return $results;
}
```

La fonction *getRandomEntity()* permet de récupérer avec la méthode *random* un média parmi ceux qui sont présents dans la BDD et l'afficher dans la section *hero* de la page *home*.

```
public function getRandomEntity(){
    $this->db->query("SELECT * FROM entities ORDER BY RAND() LIMIT 1");
    $results=$this->db->single();
    return $results;
}
```

Mudeo

show(\$id)

La méthode `show($id)` permet d'afficher le média sélectionné par l'utilisateur. Un paramètre \$id est passé. Si l'id n'existe pas, l'utilisateur sera renvoie sur la homepage. Cette fonction permet aussi d'augmenter le nombre de view unique de chaque vidéo.

```
if ($this->entitiesModel->preventDoubleInViewsCount($id)) {
    $this->entitiesModel->countUniqueViews($id);
    $this->entitiesModel->countViews($id);
}
```

La fonction dans le modèle “entities”(app/models/entities.php) se présente comme suit:

```
public function countViews($id){
    $this->db->query("UPDATE entities SET view_count = view_count + 1 WHERE id= :id");
    $this->db->bind(':id', $id);
    $this->db->execute();
}

public function countUniqueViews($id){
    $this->db->query("INSERT INTO count_views (user_id,entity_id) VALUES (:user_id,
:entity_id)");
    $this->db->bind(':entity_id', $id);
    $this->db->bind(':user_id', $_SESSION[ 'user_id']);
    $this->db->execute();
}

public function preventDoubleInViewsCount($id)
{
    $this->db->query('SELECT * FROM count_views WHERE user_id = :user_id AND entity_id=
:entity_id');
    $this->db->bind(':entity_id',$id);
    $this->db->bind(':user_id', $_SESSION[ 'user_id']);
```

Une fois que la vidéo est terminée, le système nous propose une nouvelle vidéo à visionner avec une redirection automatique. En effet, après 5 secondes la vidéo est récupérée avec la fonction `getRandomEntity()`.

about ()

La méthode `about()` permet d'afficher la page `about.php`.

Controller Users

Le contrôleur `Users.php` contient différentes méthodes. Il gère notamment l'enregistrement et le login des utilisateurs.

Construct

Le `construct()` vérifie si un cookie de connexion existe. Ensuite un appel au modèle `entities` aux `users` est effectué. Avec les fonctions :

```
$this->userModel = $this->model('User');  
$this->autoLog();
```

Index ()

L'index nous renvoie directement à la méthode `login()`. En effet, c'est depuis la méthode `login()` que l'utilisateur est redirigé sur la page `home`, s'il est connecté.

register ()

La méthode `register()` permet d'enregistrer les utilisateurs dans la BDD. Elle vérifie notamment si tous les champs du formulaire sont bien remplis. Elle vérifie aussi si l'utilisateur n'est pas déjà présent dans la

Mudeo

BDD avec la fonction *findUsersByEmail(\$email)* (app/models/users.php).

login()

La méthode *login()* vérifie si les valeurs introduites par l'utilisateur sont valides. Elle fait aussi appel à la fonction *createUserSession()* qui crée la session pour l'utilisateur.

```
private function createUserSession($ user)
{
    $_SESSION['Logged'] = true;
    $_SESSION[ 'user_id' ] = $ user->id;
    $_SESSION[ 'email' ] = $ user->email;
    $_SESSION[ 'name' ] = $ user->name;
    header('location: ' . URLROOT . '/pages/index');
}
```

Le *login()* peut aussi appeler la méthode *createUserToken()* qui est liée aux modèles *user*. Si la case *keep me logged* a été sélectionnée, un cookie de connexion est généré.

```
$ this->userModel->stayConnectedTwo ($_SESSION[ 'user_id' ]);
```

member()

La méthode *member()* affiche l'espace personnel de l'utilisateur. Elle affiche un formulaire avec lequel il est possible de modifier les informations personnelles comme le nom et l'email. Elle vérifie aussi si l'email n'est pas déjà utilisé avec la fonction *findUsersByEmail(\$email)* (app/models/users.php). Les données sont passé en POST afin

d'empêcher la modification ou la suppression d'un account dont nous ne sommes pas propriétaires. La méthode *member()* permet aussi d'effacer l'account avec la méthode POST.

sendtoken()

La méthode *sendtoken()* s'active seulement si l'utilisateur n'est pas connecté. Après avoir vérifié que l'email inséré correspond à un account dans la base de données, un token est créé avec la méthode *createResetToken()* (*app/models/user.php*).

```
public function createResetToken($userId){
    $token = bin2hex(random_bytes(16));

    $this->db->query("INSERT INTO users_reset (user_id,recover_token) VALUES (:userId,:token)");
    $this->db->bind(':token', $token);
    $this->db->bind(':userId', $userId);
    if ($this->db->execute()) {
        return true;
    }else{
        return false;
    }
}
```

Le token est ensuite récupéré avec la méthode *recoverResetToken()* (*app/models/user.php*).

```
public function recoverResetToken($userId){
    $this->db->query('SELECT * FROM users_reset WHERE user_id = :userId');
    $this->db->bind(':userId', $userId);
    $row=$this->db->single();
    return $row;
}
```

Un email contenant un lien avec le token (controller/method/param [token]) est généré et envoyé avec la librairie PHP-Mailer.

PHPMAILER

PHPMailer fournit une interface orientée objet, alors que mail () n'est pas orienté objet. En outre, le mail () nécessite un serveur de messagerie local pour envoyer des e-mails, ce qui n'est pas toujours simple à configurer. PHPMailer peut utiliser un serveur de messagerie non local (SMTP) si vous disposez d'une authentification.

reset (\$token)

La méthode *reset (\$token)* s'active quand l'utilisateur clique sur le lien reçu par mail. Ensuite elle récupère l'*user_id* et la *created_at* depuis le reset token. Si le token est toujours valable, l'utilisateur pourra modifier son mot de passe. Une fois le mot de passe modifié, le token est détruit. Si le token a expiré, il sera détruit également.

createUserToken()

La méthode *createUserToken()* est une fonction privée. Elle fait appel au modèle user.php avec la fonction :

```
$ this→userModel→stayConnectedTwo ($_SESSION[ 'user_id'])
```

pour générer un cookie et l'insérer dans la base des données. L'utilisateur pourra se connecter sur autant d'appareils qu'il veut et à chaque fois un

nouveau cookie sera créé.

```
public function stayConnectedTwo($userid)
{
    $cookie_id = uniqid();
    $this->db->query("INSERT INTO connections_user (userId,remembertoken) VALUES (:userId,:token)");
    $this->db->bind(':token', $cookie_id);
    $this->db->bind(':userId', $userid);
    // $this->db->bind(':id', $userid);

    $this->db->execute();
    setcookie('logged', $cookie_id, time() + 3600 * 24 * 30, '/');
}
```

Controller Pages Errors

Le controller *page error* envoie une *http_response_code (404)*; et affiche la page 404.

error ()

La méthode *error()* s'active quand l'utilisateur entre une mauvaise URL. Elle envoie une *http_response_code (404)*; et affiche la page 404.

Models

Les modèles gèrent toutes les requêtes SQL. Ils contiennent différentes méthodes.

CategoriesContainers.php

Cette classe récupère les données de la table SQL Categories. Elle est entre autres utilisée pour diviser les fichiers médias selon leurs *catégories Id*.

Entities.php

Cette classe récupère les données de la table SQL Entities. Elle est notamment utilisée pour gérer l'upload, la views et l'ajout dans la watch list.

Users.php

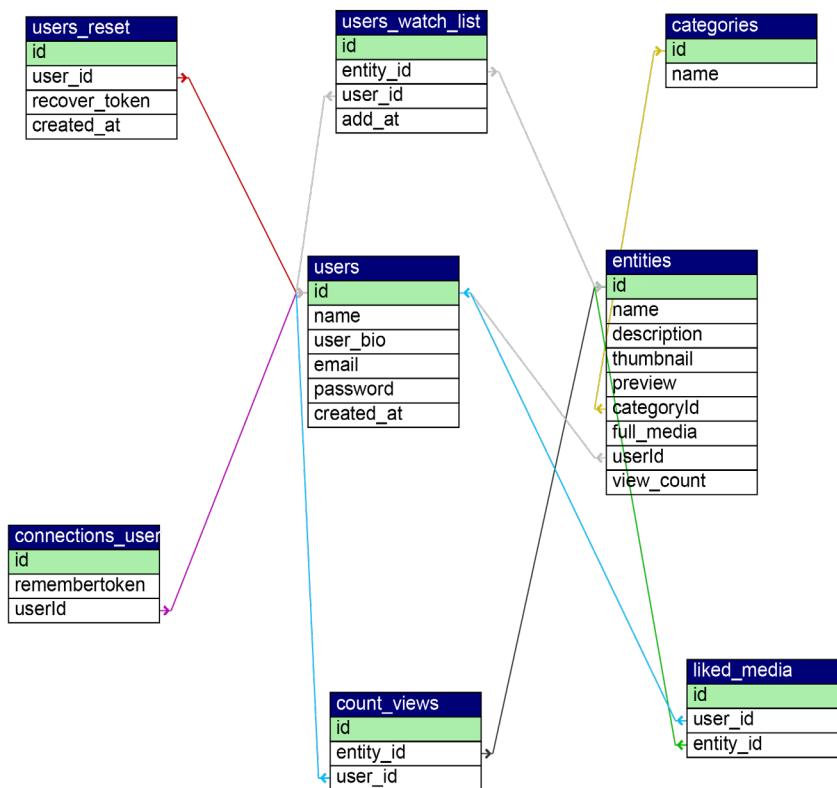
Il récupère le donné de la table SQL Users. Elle est notamment utilisée pour l'enregistrement, les connexions d'utilisateurs, le reset de mot de passe.

Pour plus d'informations sur les modèles, veuillez vous référer à ce [document](#)

SQL Database

La base des données est organisée autour de deux tables principales: User et Entities. C'est à partir de ces deux tables que toutes les clés étrangères ont été créées. En effet, si un utilisateur efface son

compte toutes les colonnes de table liées à cet account seront effacé automatiquement. De la même façon, si une vidéo est effacée, toutes les données relatives à cette vidéo seront effacées.



Mises à jour futures

Cette application web, étant conçu avec le design pattern MVC, elle est facilement évolutive. Par exemple nous pourrions ajouter une page portfolio pour les utilisateurs avec un lien personnalisé. Une autre modification pourrait consister à développer une partie de backoffice pour les administrateurs du système pour qu'ils puissent plus facilement changer les couleurs ou ajouter des catégories supplémentaires. Au vu de la tendance actuelle d'utiliser NFT, nous pourrions éventuellement permettre via un lien des transformér les fichiers médias en NFT.

Actuellement, la version mobile n'est pas pleinement fonctionnelle, bien que le site soit responsive il n'est pas encore adaptatif. La librairie FFMPEG est fonctionnelle mais risque de surcharger le serveur existe. Il serait peut-être envisageable de la remplacer avec un script personnalisé.