Emmanuel CABASSOT
Dossier de projet titre développeur web et web mobile

SOMMAIRE

Présentation

Présentation personnelle

Je m’appelle Emmanuel CABASSOT, j’ai 42 ans et j’habite Marseille. Auparavant, j’étais technicien de maintenance pour les appareils au gaz. Passionné par le domaine de l’informatique, j’ai décidé de passer par une reconversion. Après maintes recherches en matière de formation, j’ai décidé de suivre celle de Laplateforme qui m’a paru être la plus adaptée à mes besoins.

Présentation de la formation

La Plateforme_ est une école de code qui forme des personnes au métier de développeur web et web mobile sur deux ans. Avec ou sans bac, cette formation est totalement gratuite permettant une accessibilité à tous.
Cette formation est une chance de pouvoir accéder au métier de développeur web. L’apprentissage se fait par projet seul ou en équipe permettant une montée rapide en compétences et le travail en équipe.

Réseau social

Résumé de réseau social

Il nous était demandé de créer une boutique avec au minimum :

- chaque utilisateur doit avoir son propre mur et un fil d'actualité présent sur la page d'accueil devra résumer l'actualité de ses amis.
- Il devra être possible de réagir aux posts à l'aide de commentaires et de réactions (likes, émoticônes...)
- Il devra vous être possible de créer des conversations et de discuter avec des membres du réseau sans rechargement de la page
-maquetter l'application
-concevoir sa base de données avant de la créer.
-le projet doit être entièrement responsive.

Le projet couvre les compétences :

Le projet couvre les compétences énoncées ci-dessous.
Pour l'activité type 1, « Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité » :

- Développer une interface utilisateur web dynamique.

Pour l'activité type 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 la partie back-end d'une application web ou web mobile.
- Élaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce.

Fonctionnalités

Liste de ce que j'ai créé :

Créer un post

- Ajout d'emoticones
- Enregistrer le texte
- Intégrer une ou plusieurs photos ou une vidéo
  * avec possibilité de drag an drop
  *visualisation et possibilité de suppression même après que les images ou la vidéo aient été importées dans le fichier temporaire
- Grâce à un button on peut mettre le post en story
- Quand le post est validé

*On enregistre le post en BDD grâce à une requête AJAX
* les photos ou la vidéo sont déplacées du dossier temporaire et vont se ranger dans leurs dossiers respectifs

- Après que le post soit publié on supprime les previews des images ou vidéo et on remet les classes des boutons à zéro

Visualisation d'un post

- Grace à de l'AJAX on récupère les informations du post
  *les informations du post
  *les informations du créateur du post (dont l'avatar)
  *les likes et dislikes
  *les commentaires….

Action sur un post

Tout doit être fait son rechargement de la page

- Si l'utilisateur est le créateur du post il peut supprimer le post
- Sinon le signaler
- Le click sur l'avatar mène au profil du créateur
- Nombre de likes, dislikes, et le nombre de commentaires
- Visualisation des commentaires en cliquant sur le nb de commentaires (toggle)
- Action de like ou dislike changeant le nombre total
- En cliquant sur le bouton commenter apparition du text area (toggle)
- Poster un commentaire (modifie le nombre de commentaires)
- Action sur les commentaires créés
  *Si l'utilisateur est le créateur du commentaire il peut le supprimer
  *Sinon le signaler

Partie admin

- Les utilisateurs
  *faire passer un utilisateur en admin
  *bloquer un utilisateur pour une période
- Les posts
  *supprimer
  *enlever le signalement
  *bloquer un utilisateur pour une période
- Les commentaires
  *supprimer
  *enlever le signalement
  *bloquer un utilisateur pour une période

Organisation

Nous étions 3 sur le projet : Thuc Nhi, Denis et moi-même.
Nous avons commencé par créer le modèle de la BDD (MCD).
Je me suis occupé dans tâches marquées ci-dessus, Thuc Nhi et Denis étant habitués à travailler ensemble se sont partagésle reste.

Architecture du projet

- ➢ Un dossier **API**
  - ⇨ **config**
  - ⇨ **controllers**
  - ⇨ **models**
- ➢ Un dossier **assets**
  - ⇨ **css**
    - fonts
  - ⇨ **Images**
    - upload
      - events
      - groups
      - post
        - 1 (id_post)
        - …
      - temporaire
        - 1(id_user)
        - …
      - users
  - ⇨ **Js**
  - ⇨ **videos**
    - Upload
      - post
        - 1(id_post)
        - …
      - Temporaire
        - 1(id_user)
        - …

- ⇨ **Webfonts**

- ➢ Un dossier **function**
- ➢ Un dossier **includes**

Spécifications techniques

PHP ainsi que la POO pour toute la partie back de mon projet.
Javascript et AJAX pour rendre mes pages totalement dynamiques, et pour optimiser au maximum l'expérience de l'utilisateur.

Front-end

Affichage des posts

Pour l'affichage des posts le but était le même : qu'ils soient dynamiques et facilement utilisables.

On se rend compte qu'il y a beaucoup d'informations sur un post, telles que le créateur (avatar, nom, prénom), il y a combien de temps que le post a été publié, le nombre de likes, dislikes, le nombre de commentaires, les commentaires, les créateurs des commentaires…

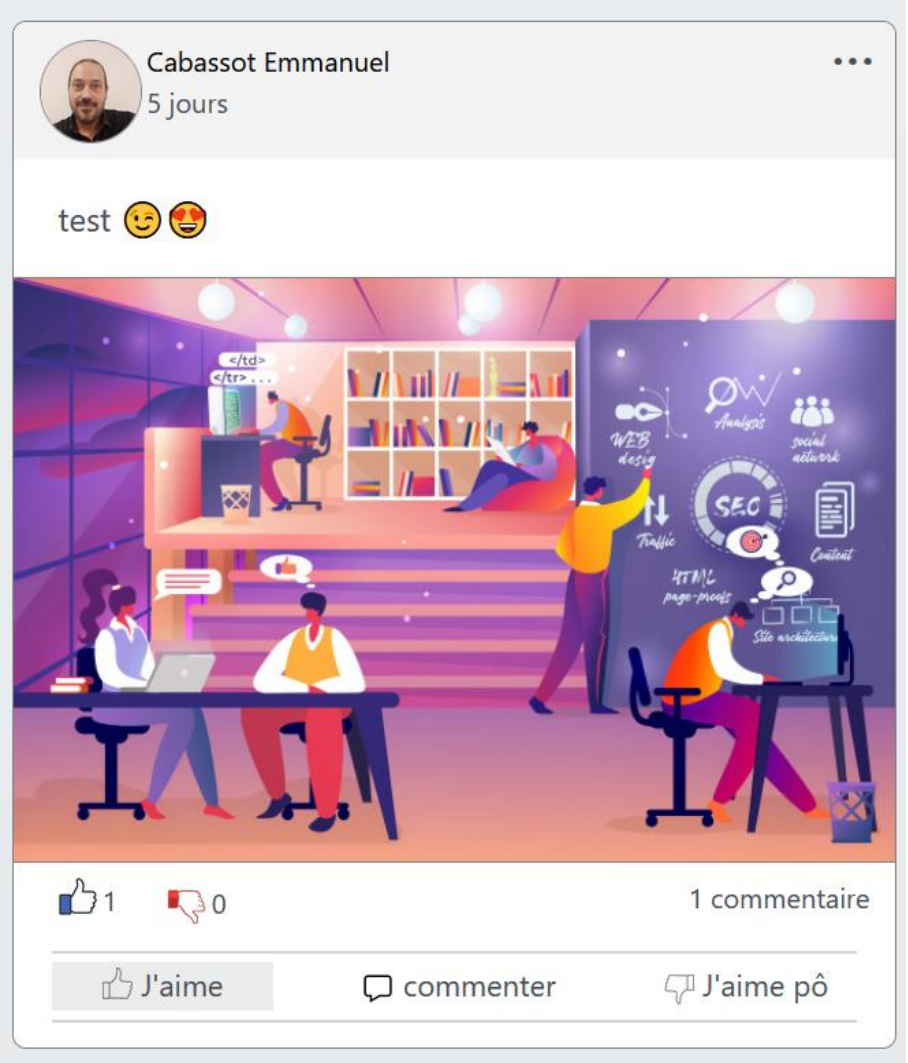
A tout ça viennent se greffer les actions possibles :

-signaler, supprimer, liker, disliker, commenter un post

-signaler ou supprimer un commentaire

Tout ça devait être fait de façon dynamique sans rechargement et devait actualiser les informations du post aussi bien en BDD qu'au visuel de l'utilisateur.

Visualisation d'un post



Les dossiers qui vont gérer l'affichage des posts et les actions possibles

social-network/api/controllers/postShow.php

Grâce à ce fichier on récupère un grand tableau contenant les informations des posts à afficher. Ce fichier est lié grâce à une requête AJAX au fichier **assets/js/murPost.js.**

Instancie la classe Database

```
// Instancie database and product object
$database = new Database();
$db = $database->getConnection();
```

Récupère les informations de la requête

```php
// On récupère les informations de la requête dans $data
$data = json_decode(file_get_contents("php://input"));
$user = $data->user;
```

Utilise la méthode idPostUser pour récupérer les posts d'un utilisateur

```php
// Récupère les posts d'un utilisateurs
$posts = $postUser->idPostUser($user);

// Initialisation de l'incrémentation pour le tableau $arrayPost
$increment = 0;
```

On commence à boucler dans les posts grâce à un foreach

```php
// On boucle dans les posts
foreach ($posts as $post) {

    // Récupère l'id du post dans la variable $id_post
    $id_post = $post['id_post'];

    // Ré-instancie la class Post
    $classPost = new Post($db);

    // Récupère les informations du post
    $post = $classPost->showPostById($id_post);
    $post['date_post'] = depuis($post['date_post']);
    $post['text_post'] = nl2br($post['text_post']);
```

On reste dans la boucle des post et l'on récupère les infos du créateur du post

```php
    // On récupère les informations du créateur du post
    $user = $classPost->userById($post['id_user']);
    $post['userLastname'] = ucfirst($user['lastname']);
    $post['userName'] = ucfirst($user['name']);
    $post['userAvatar'] = $user['avatar'];
```

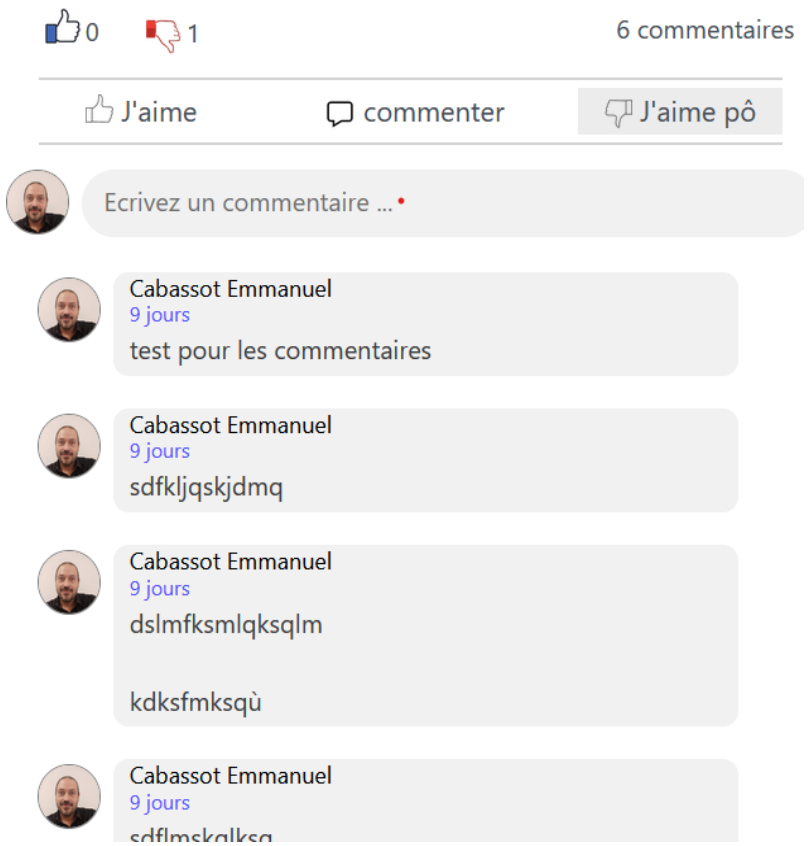
Va servir à avoir des actions différentes sur le post si l'utilisateur est ou pas le créateur

```php
// Est ce moi qui ai publier le post?
if ($post['id_user'] == $_SESSION['id_user']) {
    $post['myPost'] = 'oui';
} else {
    $post['myPost'] = 'non';
}
// Instancie la class User pour récupérer les infos de l'utilisateur
$classUserSession = new User($db);
$userSession = $classUserSession->userById($_SESSION['id_user']);
$post['avatarUserSession'] = $userSession['avatar'];
```

Il y a-t-il des photos ou vidéo associées au post ?

```php
// Il y a t'il des images?
if ($post['image_post'] == "oui") {
    $classImages = new PhotoPost($db);
    $images = $classImages->affichePhotoPost($id_post);
    $post['images'] = $images;
}else{
    $post['image_post'] = 'non';
}

// Il y a t'il une vidéo
if (isset($post['video_post']) and $post['video_post'] == "oui") {
    $classVideo = new VideoPost($db);
    $video = $classVideo->afficheVideoPost($id_post);
    $post['cheminVideo'] = $video['chemin'];
    $post['nomVideo'] = $video['name_video_post'];
}
```

Les commentaires du post



On est toujours dans la boucle des posts et on instancie la classe Comment

```php
// Instancie la class comment
    $classComment = new Comment($db);
    $countComments = $classComment->CountCommentById($id_post);
    $post['countComment'] = $countComments;
```

Si il y a des commentaires on boucle et on récupère les infos de celui qui a commenté

```php
if ($countComments > 0) {
        // Il y a des commentaires
        $classUser = new User($db);
        $comments = $classComment->showCommentById($id_post);
        $i = 0;
        // On boucle dans les commentaires
        foreach ($comments as $comment) {
             $idUserComment = $comment['id_user'];
             $userComment = $classUser->userById($idUserComment);
```

```
            $post['comments'][$i] = $comment;
            $post['comments'][$i]['date_comment_post'] =
            depuis($post['comments'][$i]['date_comment_post']);
            $post['comments'][$i]['text_comment_post'] =
            nl2br($post['comments'][$i]['text_comment_post']);
            $post['comments'][$i]['userName'] = ucfirst($userComment['name']);
            $post['comments'][$i]['userlastName'] =
            ucfirst($userComment['lastname']);
            $post['comments'][$i]['userAvatar'] = $userComment['avatar'];
            $i++;
}
```

On va chercher les infos des likes et dislikes : le nombre et si l'utilisateur a déjà liker ou disliker

```
    // On instancie la class Like
    $classLike = new Like($db);
    $countLike = $classLike->countLike($id_post);
    $post['countLike'] = $countLike;
    $possibleLike = $classLike->possibleLike($id_post, $_SESSION['id_user']);
    $post['possibleLike'] = $possibleLike;

    // On instancie la class Dislike
    $classDislike = new Dislike($db);
    $countDislike = $classDislike->countDislike($id_post);
    $post['countDislike'] = $countDislike;
    $possibleDislike =
    $classDislike->possibleDislike($id_post, $_SESSION['id_user']);
    $post['possibleDislike'] = $possibleDislike;
```

On finit par intégré $post au tableau $arrayPost et a echo ce tableau

```
// On intègre les informations de $post dans $arrayPost
    $arrayPost[$increment] = $post;
    $increment++;
}

echo json_encode($arrayPost);
```

C’est le fichier ***assets/js/murPost.js*** qui grâce à une requête AJAX récupère ce tableau pour ensuite afficher le post et écouter les différentes actions. Ce fichier fait plus de 600 lignes et je vais donc vous présenter les actions sur like et dislike.

Les actions sur like et dislike

Tout commence par la création des sections correspondantes et surtout à la fonction onclick. Dans le tableau json que je récupère il y a les lignes possibleLike et possibleDislike.

Si possibleLike == 0 cela veut dire que l’utilisateur n’a pas liké ce post et pareil pour possibleDislike. La section n’aura pas la classe validate.

J’intègre donc ces 2 variables dans une fonction onclick: likeShowPost(possibleLike, possibleDislike, id_post).

```
if (post.possibleLike == 0) {
   output += `
   <section class="like-action-showPost" id="like${post.id_post}" onclick=
      "likeShowPost(${post.possibleLike}, ${post.possibleDislike}, ${post.id_post})">
          <img src="assets/images/likeNoColor.png" alt="like"></img> J'aime
    </section>`
} else {
    output += `
    <section class="like-action-showPost validate" id="like${post.id_post}" onclick=
       "likeShowPost(${post.possibleLike}, ${post.possibleDislike}, ${post.id_post})">
           <img src="assets/images/likeNoColor.png" alt="like"></img> J'aime
    </section>`
}

if (post.possibleDislike == 0) {
   output += `
   <section class="dislike-action-showPost" id="dislike${post.id_post}" onclick=
      "dislikeShowPost(${post.possibleLike}, ${post.possibleDislike}, ${post.id_post})">
           <img src="assets/images/dislikeNoColor.png" alt="dislike"></img> J'aime pô
   </section>`
} else {
   output += `
   <section class="dislike-action-showPost validate" id="dislike${post.id_post}" onclick=
       "dislikeShowPost(${post.possibleLike}, ${post.possibleDislike}, ${post.id_post})">
            <img src="assets/images/dislikeNoColor.png" alt="dislike"></img> J'aime pô
   </section>`
}
```

Les fonctions likeShowPost() et dislikeShowPost() *:*

-Changent l'apparence des boutons like et dislike grâce à la classe 'validate'.

-Mettent à jour le nombre de like et dislike en dynamique sur l'affichage du post.

-Mettent à jour les valeurs de la fonction onclick.

-Enregistrent en BDD l'action effectuée.

Si like== 0 et dislike == 0 on crée un variable Like et dislike

Comme Like = 1 cela veut dire qu'il faudra ajouter un Like en BDD.

Si Like = -1 Cela aurait voulu dire qu'il faudrait enlever un like en BDD.

Les likes et dislikes sont en rapport avec un post et un utilisateur.

Fonction Like

```
 /* Function Like */
function likeShowPost(like, dislike, post_id) {

  if (like === 0) {
      if (dislike === 0) {

       //Variable qui vont être envoyées par la requête ajouter/supprimer en BDD
       Like = 1
       Dislike = 0

       //Cible la section du like ou dislike, supprime la fonction puis la recrée
       sectionLike = document.querySelector("#like" + post_id)
       sectionLike.removeAttribute("onclick")
       sectionLike.setAttribute("onclick", "likeShowPost(1, 0, " + post_id + ")")

       sectionDislike = document.querySelector("#dislike" + post_id)
       sectionDislike.removeAttribute("onclick")
       sectionDislike.setAttribute("onclick", "dislikeShowPost
       (1, 0, " + post_id      + ")")

       // Ajout de la classe validate et met à jour le nombre de like
       sectionLike.classList.add("validate")
       countLike = document.querySelector("#likeCount" + post_id + " span")
       valueLike = countLike.textContent
       valueLike++
       countLike.textContent = valueLike
```

```
      }

       if (dislike === 1) {
        //Variable qui vont être envoyées par la requête ajouter/supprimer en BDD
         Like = 1
         Dislike = -1

     //Cible la section et like ou dislike, supprime la fonction puis la recrée
         sectionLike = document.querySelector("#like" + post_id)
         sectionLike.removeAttribute("onclick")
         sectionLike.setAttribute
         ("onclick", "likeShowPost(1, 0, " + post_id + ")")

         sectionDislike = document.querySelector("#dislike" + post_id)
         sectionDislike.removeAttribute("onclick")
         sectionDislike.setAttribute
         ("onclick", "dislikeShowPost(1, 0, " + post_id + ")")

         // Mise à jour des classe validate et le nombre de like et dislike
         sectionLike.classList.add("validate")
         sectionDislike.classList.remove("validate")

         countLike = document.querySelector("#likeCount" + post_id + " span")
         valueLike = countLike.textContent
         valueLike++
         countLike.textContent = valueLike

         countDislike = document.querySelector("#dislikeCount" + post_id + " span
")
                valueDislike = countDislike.textContent
                valueDislike = valueDislike - 1
                countDislike.textContent = valueDislike
            }
        }
if (like === 1) {

     //Variable qui vont être envoyées par la requête: ajouter/supprimer en BDD
     Like = -1
     Dislike = 0

     //Cible la section du like ou dislike, supprime la fonction puis la recrée
     sectionLike = document.querySelector("#like" + post_id)
     sectionLike.removeAttribute("onclick")
     sectionLike.setAttribute("onclick", "likeShowPost(0, 0, " + post_id + ")")
```

```
    sectionDislike = document.querySelector("#dislike" + post_id)
    sectionDislike.removeAttribute("onclick")
    sectionDislike.setAttribute
    ("onclick", "dislikeShowPost(0, 0, " + post_id + ")")

    // Ajout de la classe validate et met à jour le nombre de like
    sectionLike.classList.remove("validate")
    countLike = document.querySelector("#likeCount" + post_id + " span")
    valueLike = countLike.textContent
     valueLike = valueLike - 1
     countLike.textContent = valueLike
}
```

L'objet qui va être envoyé par la requête AJAX

```
        let data = {
            bdLike: Like,
            bdDislike: Dislike,
            post: post_id
        }
```

*Requête AJAX qui envoie l'objet data vers **likePost.php** pour traitement*

```
        let xhr = new XMLHttpRequest();
        xhr.open("POST", "api/controllers/likePost.php");
        xhr.setRequestHeader("Content-Type", "text/plain");
        xhr.responseType = "json";
        xhr.send(JSON.stringify(data));
        xhr.onreadystatechange = function () {
            if (xhr.readyState === 4 || xhr.status == 201) {
                console.log(xhr.response)
            }
        }


    }
```

Page de traitement likePost.php

LikePost.php

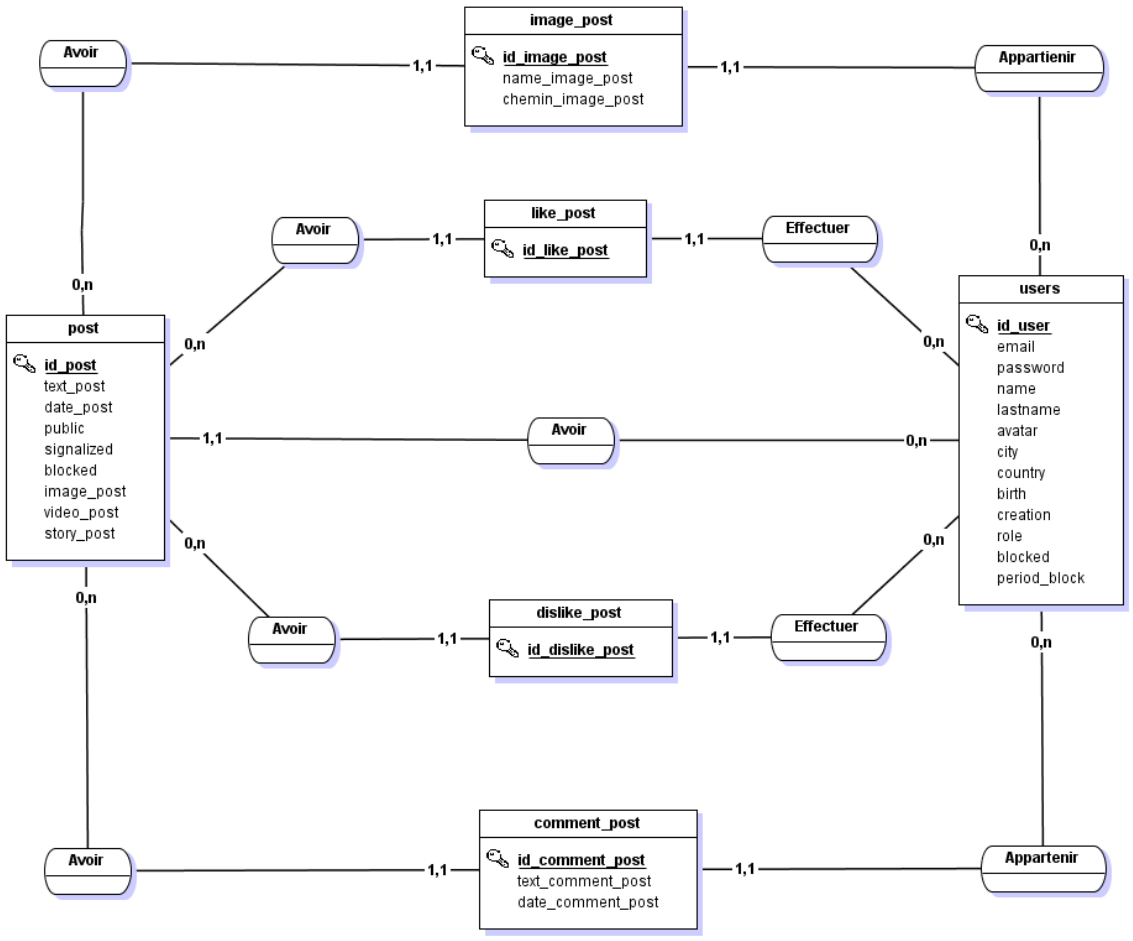
```php
$like = new Like($db);
$dislike = new Dislike($db);
if ($data->bdLike == '1') {
    // Utilise la méthode addLike pour ajouter un like
    $like->addLike($id_post,$id_user);
}

if ($data->bdLike == '-1') {

    // Utilise la méthode deleteLike pour supprimer un like
    $like->deleteLike($id_post,$id_user);
}

if ($data->bdDislike == '1') {

    // Utilise la méthode addDislike pour ajouter un dislike
    $dislike->addDisLike($id_post,$id_user);
}

if ($data->bdDislike == '-1') {

    // Utilise la méthode deleteDislike pour supprimer un dislike
    $dislike->deleteDisLike($id_post,$id_user);
}
```

Back-end

Créer une base de données

Pour notre projet je me suis occupé de la création du MCD pour la partie posts.

J’ai utilisé Jmerise pour créer cette partie du MCD.



Les commentaires, posts, like, dislike, images, vidéo ont une contrainte en casque en DELETE. Toute ces tables sont de type innoDB pour prendre en charge les contraintes.

Développer les composants d'accès aux données

Nous avons un **dossier api** dans lequel nous avons 3 dossiers.
-config/Database.php
-controllers
-models

Social-network/api/config/database.php permet la connection à la BDD. Grâce au try and catch on capture les messages d'erreurs.

```php
class Database{

    private $host = "localhost";
    private $db_name = "network";
    private $username = "root";
    private $pass = "";
    public $conn;

    // get the database connection
    public function getConnection(){

        $this->conn = null;

        try{
            $this->conn = new PDO("mysql:host=" . $this-
>host . ";dbname=" . $this->db_name, $this->username, $this->pass);
            $this->conn->exec("set names utf8mb4");
        }catch(PDOException $exception){
            echo "Connection error: " . $exception->getMessage();
        }

        return $this->conn;
    }
}
```

Social-network/api/models/Like.php est un exemple de model qui utilise la classe Database pour se connecter à la BDD.

```php
class Like
{
    // database connection and table name
    private $conn;
```

```php
    // object properties
    public $id_like_post;
    public $id_post;
    public $id_user;

    // constructor with $db as database connection
    public function __construct($db)
    {
        $this->conn = $db;
    }

    public function possibleLike($id_post, $id_user) {
        $count = "SELECT COUNT(*) FROM like_post WHERE  id_post = $id_post AND
        id_user = $id_user";
        $possibleLike = $this->conn->query($count);
        return $possibleLike->fetchColumn();
    }

    public function countLike($id_post) {
        $count = "SELECT COUNT(*) FROM like_post WHERE id_post = $id_post";
        $countLike = $this->conn->query($count);
        return $countLike->fetchColumn();
    }

    public function addLike($id_post, $id_user)
    {
        $insert = "INSERT INTO like_post (id_post, id_user)
        VALUES (:id_post, :id_user)";

        // prepare the query
        $stmt = $this->conn->prepare($insert);

        // bind post, user
        $stmt->bindParam(':id_post', $id_post);
        $stmt->bindParam(':id_user', $id_user);

        // execute the query
        $stmt->execute();
    }

    public function deleteLike($id_post, $id_user)
    {
        $delete = "DELETE FROM like_post WHERE id_post = :id_post
        AND id_user = :id_user";
```

```
        // prepare the query
        $stmt = $this->conn->prepare($delete);

        // bind post, user
        $stmt->bindParam(':id_post', $id_post);
        $stmt->bindParam(':id_user', $id_user);

        // execute the query
        $stmt->execute();
    }
}
```

social-network/api/controllers/likePost.php est un exemple de controller utilisant la classe Like.php (model).

```
// Instancie database and product object
$database = new Database();
$db = $database->getConnection();

// instantiate Like object
$like = new Like($db);
$dislike = new Dislike($db);


if ($data->bdLike == '1') {
    $like->addLike($id_post,$id_user);
}

if ($data->bdLike == '-1') {
    $like->deleteLike($id_post,$id_user);
}

if ($data->bdDislike == '1') {
    $dislike->addDisLike($id_post,$id_user);
}

if ($data->bdDislike == '-1') {
    $dislike->deleteDisLike($id_post,$id_user);
}
```

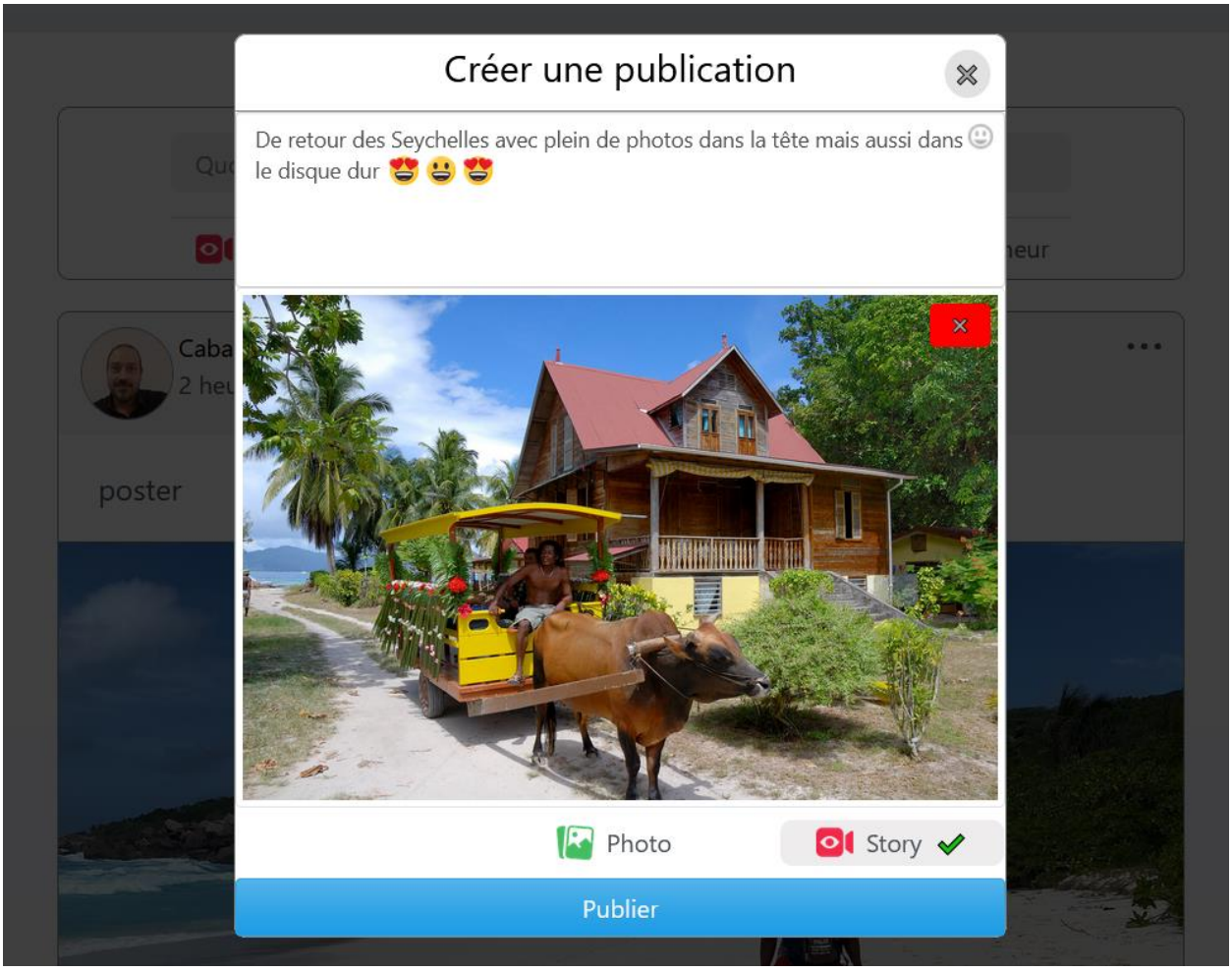
Création d’un post par l’utilisateur

Pour un meilleur confort d’utilisation **la création des posts** se fait de façon dynamique et sans avoir l’impression de remplir un formulaire grâce à l’apparition d’une modale.

L’utilisateur a accès à des emoticons. Quand il ajoute des photos ou une vidéo celles-ci viennent s’intégrer en-dessous du texte pour avoir un rendu de ce que sera son post.

À tout moment l’utilisateur peut changer d’avis en supprimant une photo ou vidéo et en remettre d’autres. Par simple clique sur un bouton il peut faire passer son post en mode story (durée de vie 24h).

Modale pour créer une publication



Je vais vous présenter une partie de la création d’un post Social-network/assets/js/sendPost.js à partir du moment où l’utilisateur appuie sur le bouton ‘publier’.

On récupère le texte du post

```
if (confirm('Voulez-vous publier?')) {
        // On récupère la valeur du textarea
        let myContent = $("#TextareaPostSend").data("emojioneArea").getText();
```

Est-ce une story ?

```
        // Est ce une story?
        let exists = !!document.querySelector(".button-story-media-
sendPost .validation-story");
        if (exists) {
            var story = "oui"
        }
        else {
            var story = "non"
        }
```

Il y a-t-il des photos ou une vidéo enregistrées ?

```
        // Il y a t'il une photo d'enregistrée et ou une vidéo
        let photo = document.querySelector('#preview').hasChildNodes()
        let video = document.querySelector('#preview-video').hasChildNodes()

        if (photo == false) {
            photo = 'non'
        }
        else {
            photo = 'oui'
        }
        if (video == false) {
            video = 'non'
        }
        else {
            video = 'oui'
        }
```

*On récupère les informations dans l'objet data et l'on envoie le tout grâce à une requête AJAX au fichier api/controllers/**postSend.php**.*

```
        data = {
            user: user,
            texte: myContent,
            photo: photo,
            video: video,
            story: story
        }
        data = JSON.stringify(data);
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "api/controllers/postSend.php");
        xhr.setRequestHeader("Content-Type", "text/plain");
        xhr.send(data);
```

Page de traitement du post (postSend.php)

On instancie la classe Database et l'on récupère les informations récupérées de la requête

```
// Instancie database and product object
$database = new Database();
$db = $database->getConnection();

$data = json_decode(file_get_contents("php://input"));
```

On instancie la classe Post dans l'objet $postSend, on hydrate l'objet et on crée en BDD le post.

```
// Instancie Modele post
$postSend = new post($db);

// Hydrate  l'objet
$postSend->image = $data->photo;
$postSend->video = $data->video;
$postSend->story = $data->story;
$postSend->user = $data->user;
$postSend->title = 'titre';
$postSend->texte = strip_tags($data->texte);


// Créer le post
$test = $postSend->createPost();
echo json_encode($test);
```

Si des photos sont présentes.

```
// Si des photos sont présentes
if ($data->photo == 'oui') {
    // Retourne l'id du post crée
    $id_post = $postSend->selectPostByText();

    // Création d'un nouveau repertoire upload/post/ + id du post
    mkdir('../../assets/images/upload/post/' . $id_post);

    // On instancie la classe PhotoPost
    $photoPost = new PhotoPost($db);
    $photoPost->id_post = $id_post;
    $photoPost->id_user = $_SESSION['id_user'];
```

Grâce à la fonction scandir() on récupère le nom des fichiers photos dans le dossier temporaire et l'on boucle dans les photos.

```
    // Liste des photos dans le fichier temporaire
    $photos = scandir('../../assets/images/upload/temporaire/' . $_SESSION['id_user']);

    foreach ($photos as $photo) {
        if ('.' !=  $photo && '..' != $photo) {
```

Supprime les éventuels espaces dans le nom du fichier, hydrate l'objet $photoPost et création en BDD la photo

```
            $photoRename = str_replace(' ', '', $photo);
            $photoPost->name_image_post = $photoRename;
            $photoPost->chemin = 'assets/images/upload/post/' . $id_post;
            $photoPost->insertPhotoPost();
```

Création de variables (chemin + nom). Fonction rename() qui déplace le fichier photo du dossier temporaire jusqu'à son dossier final.

```
            $dossierSource = '../../assets/images/upload/temporaire/'
            . $_SESSION['id_user'] .            '/' . $photo;
            $dossierDestination = '../../assets/images/upload/post/'
```

```
            . $id_post . '/' . $photoRename;
            rename($dossierSource, $dossierDestination);
        }
    }
}
```

Pour la vidéo l'on refait la même chose que pour les photos.

```
if ($data->video == 'oui') {
    // Retourne l'id du post crée
    $id_post = $postSend->selectPostByText();

    // Création d'un nouveau repertoire upload/post/ + id du post
    mkdir('../../assets/videos/upload/post/' . $id_post);

    // On instancie la classe VideoPost
    $videoPost = new VideoPost($db);
    $videoPost->id_post = $id_post;
    $videoPost->id_user = $_SESSION['id_user'];

    // Liste des photos dans le fichier temporaire
    $videos = scandir('../../assets/videos/upload/temporaire/' . $_SESSION['id_user']);


    foreach ($videos as $video) {
        if ('.' !=  $video && '..' != $video) {

            $videoPost->name_video_post = str_replace(" ", "", $video);
            $videoPost->chemin = 'assets/videos/upload/post/' . $id_post;
            $videoPost->insertVideoPost();


            $dossierSource = '../../assets/videos/upload/temporaire/' . $_SESSION['id_user'] .
            '/' . $video;
            $dossierDestination = '../../assets/videos/upload/post/' . $id_post . '/' .
            str_replace(" ", "", $video);
            rename($dossierSource, $dossierDestination);
        }
    }
}
```

Partie administrateur

Il fallait donner la possibilité aux administrateurs d'avoir une remontée des posts et commentaires non acceptables. Pour cela il est possible de signaler des posts et commentaires par les utilisateurs.

Il fallait aussi pouvoir faire passer des utilisateurs à admin et inversement.

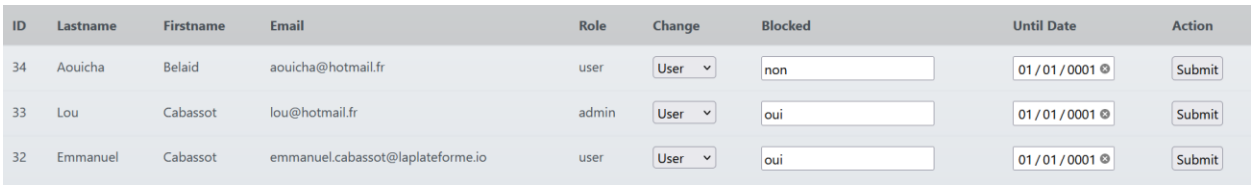
Nous avons la page crudAdmin.php

Elle permet :

Liste des utilisateurs :

-passer d'utilisateur à admin et inversement

-bloquer un utilisateur pour une période donnée ou l'inverse



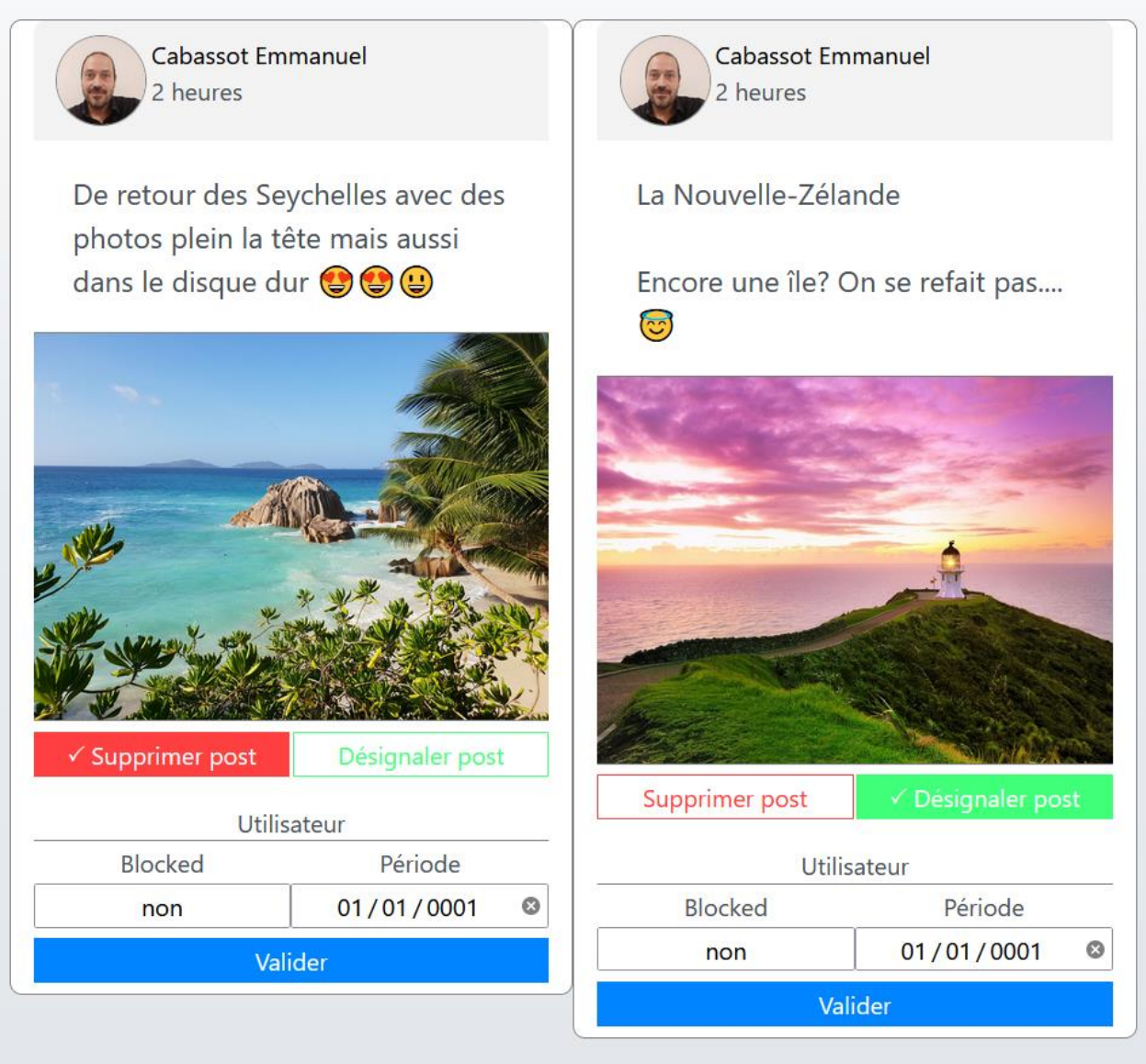
ID	Lastname	Firstname	Email	Role	Change	Blocked	Until Date	Action
34	Aouicha	Belaid	aouicha@hotmail.fr	user	User	non	01 / 01 / 0001	Submit
33	Lou	Cabassot	lou@hotmail.fr	admin	User	oui	01 / 01 / 0001	Submit
32	Emmanuel	Cabassot	emmanuel.cabassot@laplateforme.io	user	User	oui	01 / 01 / 0001	Submit

Liste des postes signalés :

-supprimer le post

-désignaler le post

-bloquer le créateur du post pour une période donnée

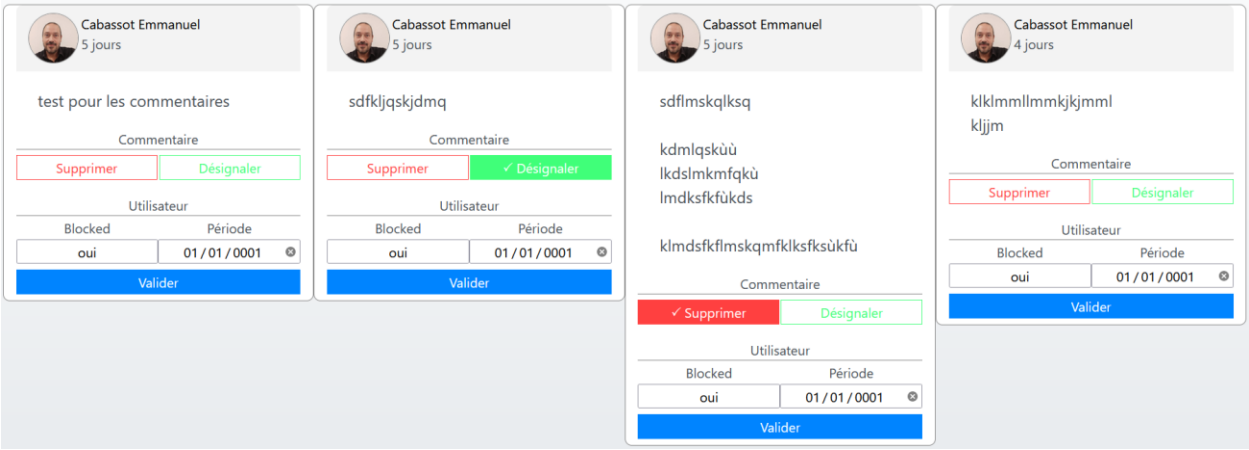


Liste des commentaires signalés :

-supprimer le commentaire

-désignaler le commentaire

-bloquer le créateur du commentaire pour une période donnée



Présentation de la gestion des actions des posts par l'administrateur

Fichier admPosts.js

Le tableau response contient les posts signalés

```
// Boucle dans le tableau des posts signalés
response.forEach(post => {
```

Va chercher les boutons supprime et deleteSignal par rapport à l'id du post

```
    // Cherche les boutons supprime et deleteSignal par rapport à l'id du post
    supprimePost = listPosts.querySelector("#supprime-
adminOption" + post.id_post)
    deleteSignalPost = listPosts.querySelector("#deleteSignal-adminOption"
 + post.id_post)
```

Ecoute le bouton deleteSignal

```
    deleteSignalPost.addEventListener("click", function (e) {

        this.classList.toggle("valide")

        // Si la bouton supprime a la classe valide on lui supprime
```

```
        supprime = listPosts.querySelector("#supprime-adminOption" +
        post.id_post)
        if (supprime.classList.contains("valide")) {
            supprime.classList.remove("valide")
        }
    })
```

Ecoute le bouton supprime au click avec un toogle sur la classe valide

```
    supprimePost.addEventListener("click", function (e) {
        this.classList.toggle("valide")

        // Si la bouton deleteSignal à la classe valide on lui supprime
        signal = listPosts.querySelector("#deleteSignal-adminOption" +
        post.id_post)
        if (signal.classList.contains("valide")) {
            signal.classList.remove("valide")
        }
    })
```

Ecoute du bouton de validation du forumlaire

```
        // Ecoute du bouton de validation
        submit = listPosts.querySelector("#submit" + post.id_post)
        submit.addEventListener("click", function (e) {

        // Si supprime contient classe valide alors supprime === 'oui'
        supprimePost = listPosts.querySelector("#supprime-adminOption" +
        post.id_post)
            if (supprimePost.classList.contains("valide")) {
                supprime = 'oui'
            }else {
                supprime = 'non'
            }

        // Si deleteSignal contient la classe valide alors deleteSignal === 'oui'
            deleteSignalPost = listPosts.querySelector
            ("#deleteSignal- adminOption" + post.id_post)
            if (deleteSignalPost.classList.contains("valide")) {
                deleteSignal = 'oui'
            }else {
```

```
        deleteSignal = 'non'
    }

    // On récupère les valeurs de blocked et date
    blocked = listPosts.querySelector("#blocked" + post.id_post)
    blocked = blocked.value

    date = listPosts.querySelector("#date" + post.id_post)
    date = date.value
```

Objet data qui va être envoyé grâce à la requête et hidden du post

```
    // Objet data qui va être envoyé grâce à la requête
    data = {
        supprime: supprime,
        deleteSignal: deleteSignal,
        blocked: blocked,
        date: date,
        post: post.id_post,
        user: post.id_user
    }

    // Ajoute la classe hidden au post
    if (supprime == 'oui' || deleteSignal == 'oui') {
        postHidden = listPosts.querySelector("#showOnePost"
        + post.id_post)
        postHidden.classList.add("hidden")
    }
```

Requête AJAX qui envoie les données à modifPost.php

```
    // Requête AJAX
    let xhrr = new XMLHttpRequest();
    xhrr.open("POST", "api/controllers/modifPost.php");
    xhrr.setRequestHeader("Content-Type", "text/plain");
    xhrr.responseType = "json";
    xhrr.send(JSON.stringify(data));

    xhrr.addEventListener("readystatechange", function() {
    if (xhrr.readyState === 4 && xhrr.status == 200) {
```

```
                console.log("ca marche")
            }
        })
        })
});
```

Page de traitement modifPost.php

```php
// get database connection
include_once '../config/database.php';

// instantiate product object
include_once '../models/User.php';
include_once '../models/Post.php';

// Instancie database and product object
$database = new Database();
$db = $database->getConnection();

$data = json_decode(file_get_contents("php://input"));

// On instancie la classe User
$user = new User($db);

// On hydrate l'objet User
$user->blocked = $data->blocked;
$user->period_block = $data->date;

// On met à jour l'utilisateur
$user->updateUserBlockedPost($data->user);

// On instancie la classe Post
$post = new Post($db);

// Si supprime === oui on supprime le post
if ($data->supprime == 'oui') {
    $post->deletePostById($data->post);
}

// Si deleteSignal === oui on enleve le signalement
if ($data->deleteSignal == 'oui') {
    $post->deleteSignalPostById($data->post);
}
```

Sécurité

Dans l'optique de rendre le site le plus sécurisé possible, je me suis penché sur les différentes failles qui pouvaient exister.

L'injection SQL

Une injection sql est type de piratage le plus courant, c'est un type d'injection de code, les pirates exploitent des failles de sécurité du site pour envoyer des requêtes SQL à la base de données, qui peut modifier les informations contenues dans celle-ci , voire les supprimer. Pour éviter cette attaque, je fais des vérifications côté client et serveur avant l'envoi des données en base de données, je prépare aussi mes requêtes grâce à PDO qui va de son côté appliquer un filtre pour vérifier le type du paramètre et utiliser sa fonction interne pour éviter la plupart des injonctions. Je vérifie tout ce qui provient de l'utilisateur en vérifiant que les valeurs reçues sont bien celles demandées.

Failles XSS

Pour éviter les failles XSS j'ai utilisé les fonctions strip_tags() et htmlspecialchars().

Traduction d'une recherche