

Emmanuel CABASSOT

Dossier de projet titre développeur web et web mobile

SOMMAIRE

1- Présentation	3
• Présentation	3
• Présentation de la formation	3
2- Introduction	
• Résumé des projets	4
3- Boutique de boutiques	5
• Résumé boutique de boutiques	5
• Ce projet couvre les compétences	5
• Fonctionnalités	5
4- Organisation	6
• Architecture du projet	6
5- Front-end	9
• Réaliser une interface utilisateur web et mobile statique et adaptable	9
6- Back-end	10
• Développer les composants d'accès aux données	12

• Développer la partie back-end d'une application web ou web mobile	19
7- Veille et sécurité	21
A COMPLETER ?	
8- Réseau social	22
• Résumé du réseau social	22
• Le projet couvre les compétences	22
9- Organisation	24
• Architecture du projet	24
• Spécifications techniques	25
10- Front-end	25
• Maquetter une application	26
• Développer une interface utilisateur web dynamique	26
• Réaliser une interface utilisateur avec une solution de contenu ou e-commerce	26
11- Back-end	34
• Créer une base de données	34
• Développer les composants d'accès aux données	35
• Développer la partie back-end	39
• Elaborer et mettre en œuvre des composants dans une application de contenu ou e-commerce	43
12- Veille et sécurité	49
A COMPLETER ?	

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.

Introduction

Résumé des projets

Je vais vous présenter 2 projets.

-**Boutique de boutiques** qui permet à des particuliers d'acheter des articles et de pouvoir aussi créer une boutique ainsi qu'à des professionnels de créer une boutique.

Techno: HTML, CSS, PHP en programmation orientée objet et le design pattern MVC

Points concernés :

-**Réseau social** avec des créations de posts avec vidéo, photos et la possibilité de les mettre en story et le CRUD admin.

Technos : HTML, CSS, PHP en POO, Javascript avec requêtes AJAX

Points concernés :

Boutique de boutiques

Résumé Boutique de boutiques

Créer une boutique de boutiques qui permet à un utilisateur d'acheter des articles.

L'utilisateur peut créer sa boutique pour y vendre des articles juste en confirmant son adresse et son nom.

Un professionnel crée sa boutique grâce aux informations de sa société et son numéro SIRET.

Une fois la boutique de pro ou de particulier créée, le membre se retrouve avec **une liste déroulante 'boutique'** depuis laquelle il peut :

- ajouter des articles
- modifier ou supprimer ses articles
- modifier les informations de sa boutique
- modifier la photo de présentation de sa boutique
- un page de présentation de la boutique est visitable par tout utilisateur ce qui lui permet d'avoir des informations sur les articles vendus par la boutique depuis quand a été créée la boutique, boutique pro ou particulier...

List déroulante user ('profil') depuis laquelle un user qu'il ait une boutique ou non:

- voir son profil
- modifier sa photo, son adresse
- voir la liste de ses commandes

Une page d'accueil avec une barre de recherche et les derniers articles mis en ligne.

L'idée était de faire un site avec une inscription facile qui permet de passer d'un statut d'utilisateur sans boutique et donc acheteur à utilisateur ayant une boutique très facilement.

Il n'y a pas d'échange d'argent entre le vendeur et l'acheteur puisque l'acheteur nous paie et une fois la livraison effectuée c'est le site qui paie le vendeur.

Ce projet a été fait avec le design pattern MVC ce qui a entraîné l'utilisation d'un router, de la POO avec les namespaces et un autoloader 'maison' .

Le projet couvre les compétences :

Le projet boutique de boutiques 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 » :

- Réaliser une interface utilisateur web statique et adaptable.
- 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 » :

- Développer les composants d'accès aux données.

Fonctionnalités

Nous avons décidé de pimenter le sujet en faisant une boutique de boutiques.

Liste de ce que j'ai créé

Pour les utilisateurs:

- Un accès utilisateur avec inscription et connexion
- page de vue de leur profil lien vers : modifier profil, créer boutique/ voir boutique
- page modifier profil : changement avatar, code, nom.....
- page modifier adresse
- page mes commandes : historique ou en cours
- onglet panier
- Système de paiement (stripe)

Pour création d'une boutique de particulier

-
- Vérification qu'un utilisateur est bien créé. Nom de la boutique, possibilité de changer les informations de l'utilisateur (placeholder) et son adresse (placeholder)

-

Pour création d'une boutique de professionnel

- Pas besoin d'être inscrit en tant qu'utilisateur car la boutique pro n'est pas liée à un utilisateur. Nom de la société, email, adresse, numéro de siret, mot de passe

Pour les boutiques:

- Un accès boutique professionnel avec connexion, pour les boutiques de particulier la connexion se fait par l'utilisateur
- page du profil de la boutique : carte de la boutique avec lien vers modifier profil, produits vendus avec le lien vers page du produit
- page vendre un article : titre, catégorie, description, poids (calcul de livraison), image, nombre d'articles en stock.
- page paramètres de la boutique : image de la boutique, nom de la boutique. Pour les boutiques de professionnels vient s'ajouter en plus l'adresse.

Page d'accueil:

- barre de recherche des articles soit par rapport au nom et/ou catégorie (créé par Mathias)
- affichage des derniers articles mis en ligne avec le lien vers l'article sélectionné

Page d'un article

- titre, description, stock, prix, prix de la livraison, publié il y a...
- si cela n'est pas mon annonce : ajouter au panier sinon modifier annonce ou supprimer
- carte de la boutique qui vend l'article avec son nom, son image, pro ou particulier, créée il y a... et le lien vers la boutique

Paielement par stripe

- paielement par l'API Stripe

Organisation

Nous étions 2 sur le projet boutique de boutiques, Mathias Tavernier et moi-même. Vu que j'avais déjà commencé à apprendre le design pattern MVC j'ai commencé à créer le site pendant que Mathias apprenait le MVC. Cela m'a permis de créer la structure MVC du site et les parties décrites au-dessus.

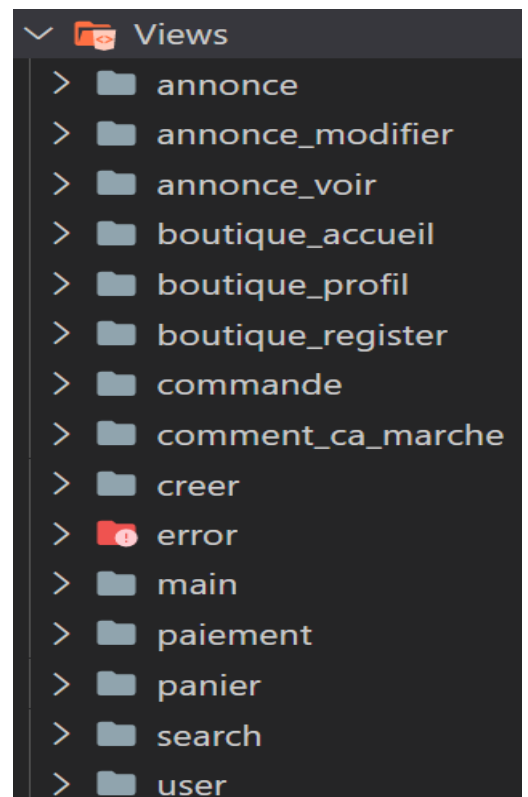
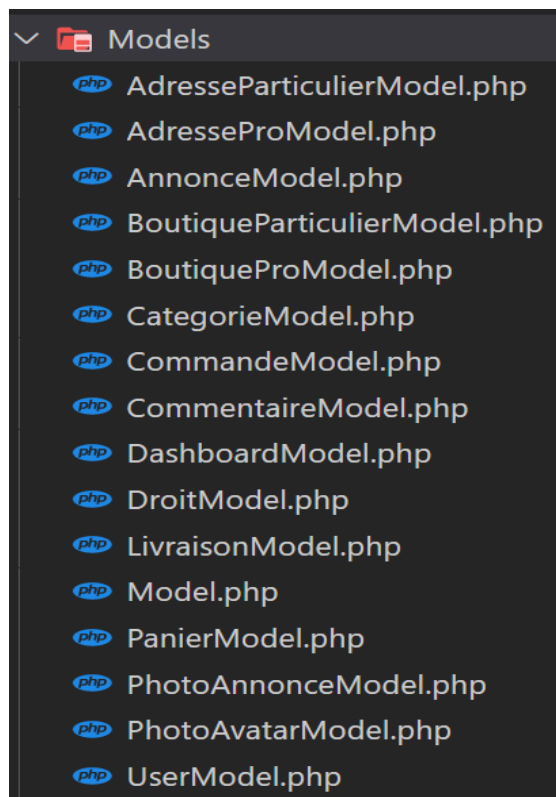
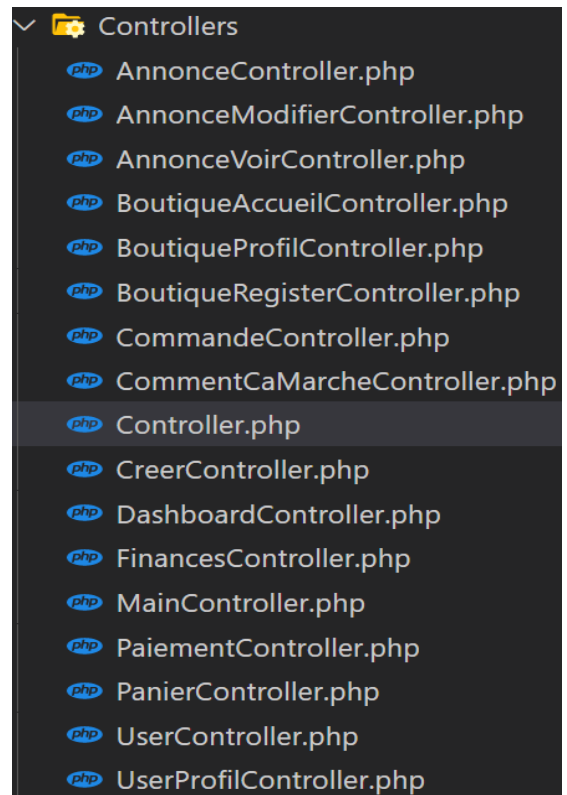
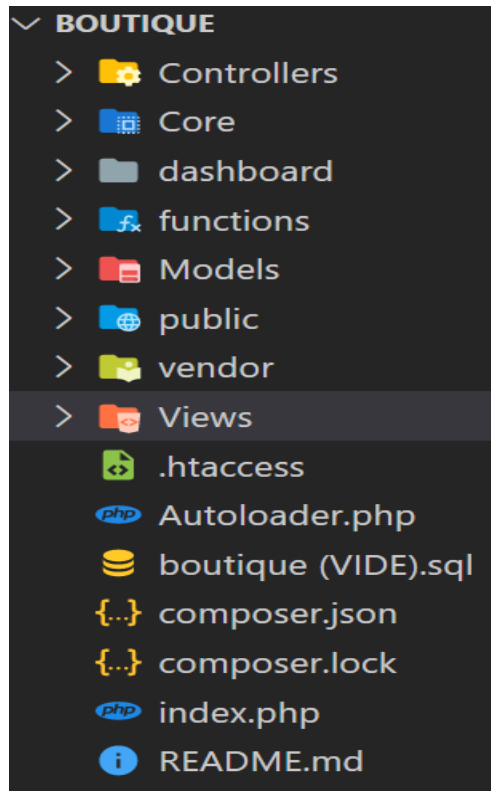
Dans un deuxième temps Mathias a créé la barre de recherche, le panier, l'historique des commandes et le CRUD Admin.

Architecture du projet

Dans le but d'une maintenabilité du site et d'un code propre, nous avons donc opté pour le design pattern MVC.

A la racine du projet nous avons donc :

- dossier Controller :
- dossier Core :
- dossier functions :
- dossier Models
- dossier Public
- dossier Views
- fichier .htaccess
- fichier Autoloader.php
- fichier index.php
- Sont venus s'ajouter d'autres éléments pour l'API Stripe.



Front-end

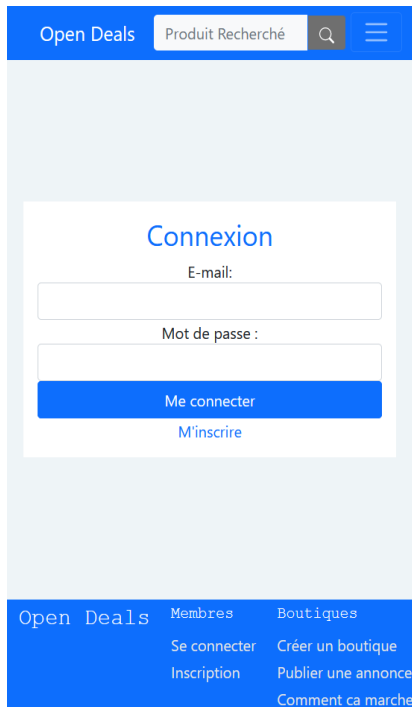
Réaliser une interface utilisateur web et mobile statique et adaptable

J'ai créé ce projet dans l'optique qu'il puisse être consulté de n'importe quel écran, du portable à l'écran d'ordinateur.

J'ai utilisé bootstrap pour adapter le design du site aux différentes largeurs d'écran.

The screenshot displays a web application interface. At the top, a blue navigation bar contains the text 'Open Deals' on the left, a search bar with the placeholder 'Produit Recherché' and a magnifying glass icon, and a list of links on the right: 'Comment ça marche', 'Créer sa boutique', 'Inscription', and 'Connexion'. The main content area has a light blue background and features a white login form centered on the page. The form is titled 'Connexion' in blue. It includes two input fields: 'E-mail:' and 'Mot de passe :'. Below these fields is a blue button labeled 'Me connecter', and a link labeled 'M'inscrire' is positioned directly underneath the button. At the bottom of the page, a blue footer bar is divided into four sections. The first section on the left contains 'Open Deals'. The second section, 'Membres', lists 'Se connecter' and 'Inscription'. The third section, 'Boutiques', lists 'Créer un boutique', 'Publier une annonce', and 'Comment ça marche'. The final section on the right, 'Retrouvez nous sur', displays social media icons for Twitter, Facebook, and Instagram.

Page profil version pc



Open Deals

Produit Recherché

Connexion

E-mail:

Mot de passe :

Me connecter

M'inscrire

Open Deals Membres Boutiques

Se connecter Créer un boutique

Inscription Publier une annonce

Comment ça marche

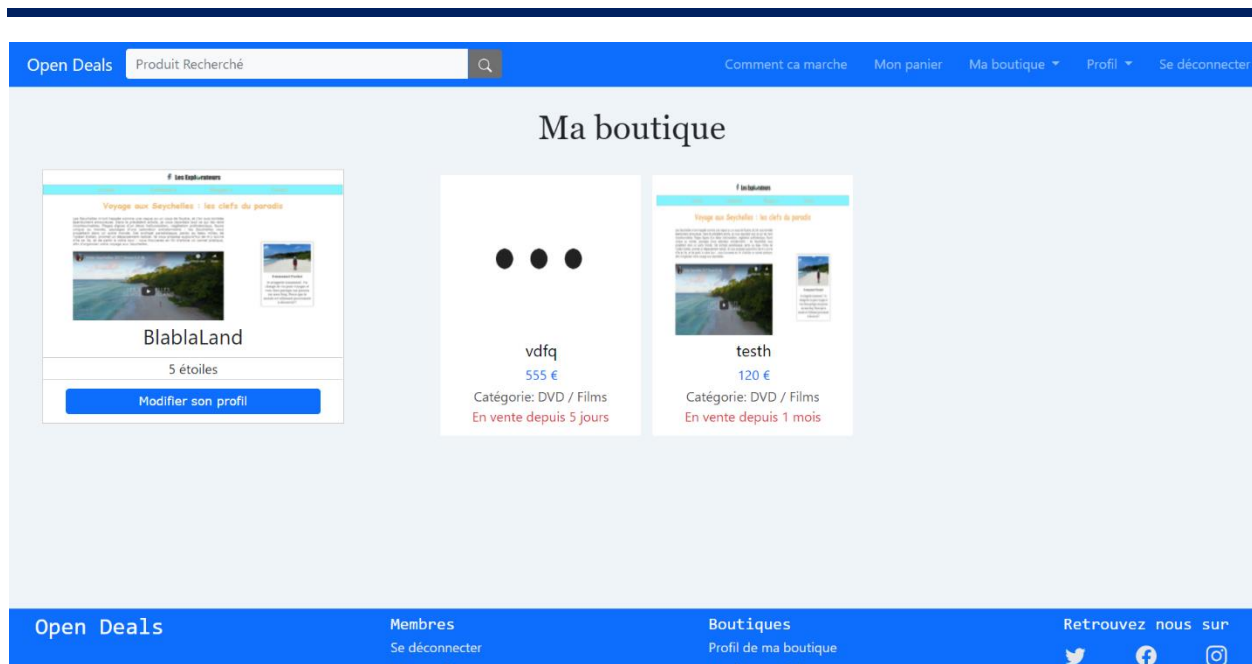
Page profil version mobile

Développer une interface utilisateur web dynamique

Le site est généré entièrement en PHP. Les utilisateurs du site pourront grâce à cela, ajouter des articles, les modifier ou les supprimer dans leur espace boutique.

Pour le style des formulaires, j'ai choisi Bootstrap car cela me permet une adaptation plus facile pour tous les écrans.

Page d'accueil d'une boutique



Les tableaux de variables qui servent à afficher la page d'accueil de la boutique de particulier sont récupérés grâce à la méthode `accueilPar` située dans la classe `boutiqueAccueilController.php`.

`Boutique/Controllers/boutiqueAccueilController.php`

```
/**
 * Affiche le profil de la boutique de particulier
 *
 * @return void
 */
public function accueilPar()
{
    if (isset($_SESSION['user']) and $_SESSION['user']['droit'] = 10) {

        (int) $id = $_SESSION['user']['id'];
        $photos = new PhotoAvatarModel;
        $photo = $photos->findPhotoBoutiquePar($_SESSION['user']['boutique_id']);
        $photo_boutique = $photos->findPhotoBoutiquePar($_SESSION['user']['boutique_id']);

        $boutiques = new BoutiqueParticulierModel;
        $boutique = $boutiques->findBoutiqueByUser($id);

        $adresse = new AdresseParticulierModel;
        $adresse = $adresse->findAdresse($id);

        $annonce = new AnnonceModel;
        $annonce = $annonce->findAnnonceParLimit($boutique->id);

        $this->render('boutique_accueil/accueil_par', ['boutique' => $boutique, 'adresse' => $adresse, 'annonce' => $annonce, 'photo' => $photo]);
    } else {
        $_SESSION['erreur'] = "Vous n'avez pas les accès pour cette page";
        header('location: ' . ACCUEIL);
    }
}
```

L'affichage se fait grâce au fichier `accueil_par.php` étant situé dans le dossier `boutique_accueil` étant lui-même dans le dossier `views`.

`Boutique/Views/boutique_accueil`

Partie gauche : carte de la boutique avec l'image de la boutique, son nom, et lien pour modifier les informations.

```
<section class="profil">
  <section class="avatar">
    <?php
    if ($photo_boutique == false) { ??
      
    <?php
    } else { ??
      
    <?php
    }
    ?>
    <p><?= ucfirst($boutique->nom_boutique) ?></p>
  </section>
  <section class="note">
    <?php
    if (isset($note)) {
      echo "note";
    } else {
      echo "5 étoiles";
    }
    ?>
  </section>
  <section class="modifier">
    <a class="btn btn-primary col-12" href="<?= ACCUEIL ?>boutiqueprofil/profilparticulier">Modifier son profil</a>
  </section>
</section>
```

Partie droite : Les annonces publiées par la boutique. Lorsque l'on clique sur l'annonce cela mène l'utilisateur vers une page qui lui permet de supprimer son annonce ou la modifier.

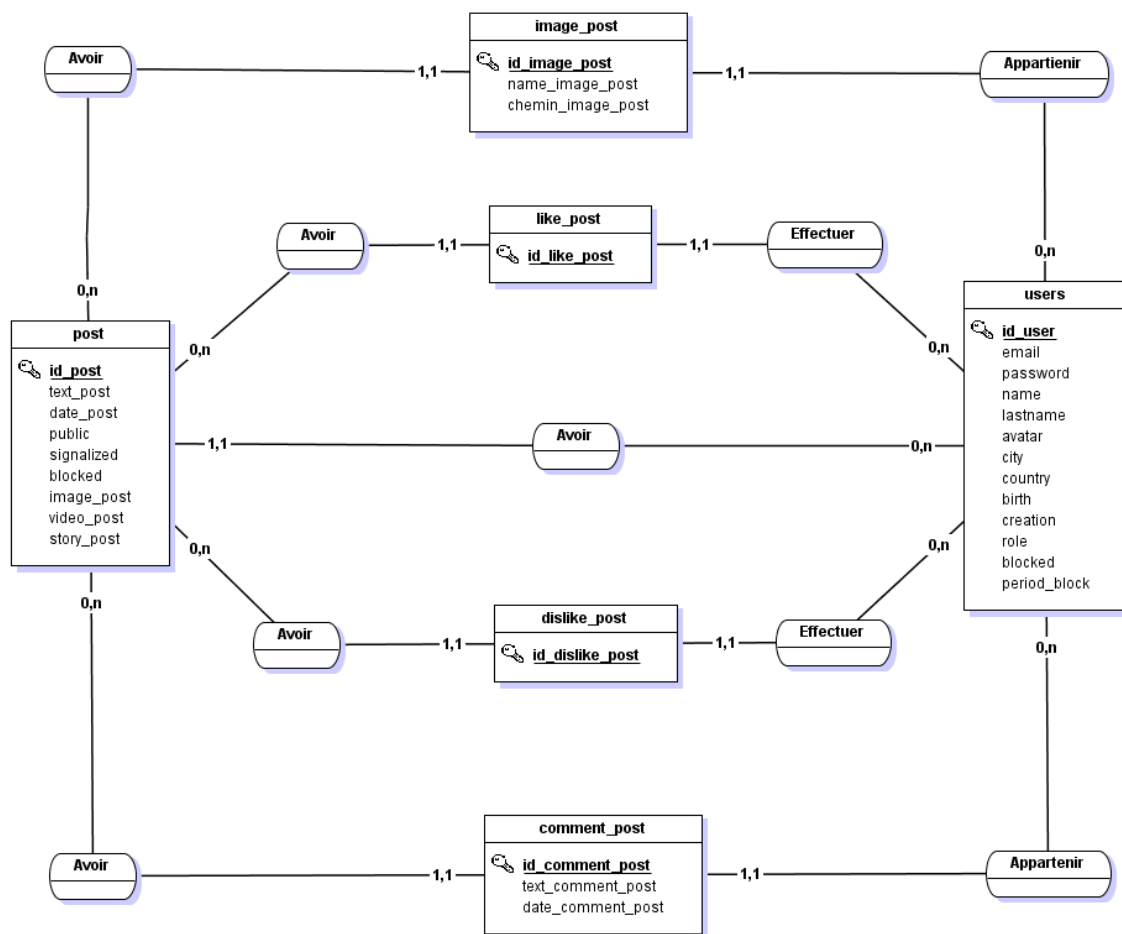
```
<section class="annonces">
  <?php
  $photo_annonce = new PhotoAnnonceModel;
  $categorie = new CategorieModel;
  if (isset($annonce) and !empty($annonce)) {
    foreach ($annonce as $annonces) {
      $categories = $categorie->findByIdCategorie($annonces->categorie_id);?>
      <section class="annonce">
        <a href="<?= ACCUEIL ?>annonceVoir/boutiquePar/<?= $boutique->id ?>/<?= $annonces->id ?>">
          <section class="photo">
            <?php
            $photo_annonces = $photo_annonce->findPhotoByAnnonceId($annonces->id);
            ?>
            
          </section>
          <section class="description">
            <div class="titre"><?= $annonces->titre ?></div>
            <div class="prix"><?= $annonces->prix ?> €</div>
            <div class="categorie">Catégorie: <?= $categories->nom ?></div>
            <div class="date_annonce">En vente depuis <?= depuis($annonces->create_at) ?></div>
          </section>
        </a>
      </section>
    <?php
    }
  } else { ??
    <p>Pas d'annonces publiées</p>
  <?php
  }
  ?>
</section>
```

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 innnoDB pour prendre en charge les contraintes.

Développer les composants d'accès aux données

Création de la class Db qui extends de PDO et va faire la connexion à la BDD
Bouique/Core/Db.php

```
// On "importe" PDO
use PDO;
use PDOException;

class Db extends PDO
{
    // Instance unique de la classe
    private static $instance;

namespace App\Core;

// On "importe" PDO
use PDO;
use PDOException;

class Db extends PDO
{
    // Instance unique de la classe
    private static $instance;

    // Informations de connexion
    private const DBHOST = 'localhost:3306';
    private const DBUSER = 'emmanuel';
    private const DBPASS = '*****';
    private const DBNAME = 'emmanuel-cabassot_boutique';

    private function __construct()
    {
        // DSN de connexion
        $_dsn = 'mysql:dbname='. self::DBNAME . ';charset=utf8; host=' . self::DBHOST;

        // On appelle le constructeur de la classe PDO
        try{
            parent::__construct($_dsn, self::DBUSER, self::DBPASS);
```

```

        $this->setAttribute(PDO::MYSQL_ATTR_INIT_COMMAND, 'SET NAMES utf8');
        $this->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_OBJ);
        $this->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    }catch(PDOException $e){
        die($e->getMessage());
    }
}

public static function getInstance():self
{
    if(self::$instance === null){
        self::$instance = new self();
    }
    return self::$instance;
}
}

```

Ici sera L'explication de la class db pour la connexion à la bdd

Classe Models qui extends de Db :

```

namespace App\Models;

use App\Core\Db;

class Model extends Db
{
    // Table de la base de données
    protected $table;

    // Instance de Db
    private $db;
}

```

Elle intègre les méthodes :

- requete(),
- create(),
- update(),
- delete()
- hydrate().
- findBy()
- find()

Méthode requete :

```
public function requete(string $sql, array $attributs = null)
{
    // On récupère l'instance de Db
    $this->db = Db::getInstance();

    // On vérifie si on a des attributs
    if ($attributs !== null)
    {
        // Requête préparée
        $query = $this->db->prepare($sql);
        $query->execute($attributs);
        return $query;
    }
    else
    {
        // Requête simple
        return $this->db->query($sql);
    }
}
```

Méthode create :

```
public function create()
{
    $champs = [];
    $inter = [];
    $valeurs = [];

    // On boucle pour éclater le tableau
    foreach ($this as $champ => $valeur) {
        // INSERT INTO annonces (titre, description, actif) VALUES (?, ?, ?)
        if ($valeur !== null && $champ != 'db' && $champ != 'table') {
            $champs[] = $champ;
            $inter[] = "?";
            $valeurs[] = $valeur;
        }
    }

    // On transforme le tableau "champs" en une chaîne de caractères
    $liste_champs = implode(' ', $champs);
    $liste_inter = implode(' ', $inter);

    // On exécute la requête
```

```

        return $this->requete('INSERT INTO ' . $this->table . ' (' . $liste_champs . ')VALUES(' . $liste_inter . ')', $valeurs);
    }

```

Méthode update :

```

public function update()
{
    $champs = [];
    $valeurs = [];

    // On boucle pour éclater le tableau
    foreach ($this as $champ => $valeur) {
        // UPDATE annonces SET titre = ?, description = ?, actif = ? WHERE id
        = ?

        if ($valeur !== null && $champ != 'db' && $champ != 'table') {
            $champs[] = "$champ = ?";
            $valeurs[] = $valeur;
        }
    }
    $valeurs[] = $this->id;

    // On transforme le tableau "champs" en une chaîne de caractères
    $liste_champs = implode(', ', $champs);

    // On exécute la requête
    return $this->requete('UPDATE ' . $this->table . ' SET ' . $liste_champs . ' WHERE id = ?', $valeurs);
}

```

Méthode hydrate :

```

public function hydrate($donnees)
{
    foreach ($donnees as $key => $value) {
        // On récupère le nom du setter correspondant à la clé (key)
        // titre -> setTitre
        $setter = 'set' . ucfirst($key);

        // On vérifie si le setter existe
        if (method_exists($this, $setter)) {
            // On appelle le setter
            $this->$setter($value);
        }
    }
    return $this;
}

```

```
}
```

Des classes sont associées aux tables, elles étendent de la classe Model.php.

Classe AdresseParticulierModel.php

```
namespace App\Models;

class AdresseParticulierModel extends Model
{
    protected $id;
    protected $user_id;
    protected $adresse;
    protected $code;
    protected $ville;

    public function __construct()
    {
        $this->table = 'adresse_particulier';
    }
}
```

On crée des attributs correspondants aux colonnes de la table en BDD en protected.
Création d'un constructeur qui donne à l'attribut table le nom de la table correspondante.

Les setters et getters :

```
/**
 * Get the value of id
 */
public function getId()
{
    return $this->id;
}

/**
 * Set the value of id
 *
 * @return self
 */
public function setId($id)
{
    $this->id = $id;
}
```

```
        return $this;
    }
```

Exemple de méthode dans la classe :

```
public function findAdresse($user_id)
{
    return $this->requete("SELECT * FROM $this->table WHERE user_id = $user_id")->fetch();
}
```

Exemple d'utilisation des Models dans les Controllers :

```
$boutique = new BoutiqueProModel;
$boutique = $boutique->find($boutique_pro_id);

$photo = new PhotoAvatarModel;
$photo = $photo->findPhotoBoutique($boutique->id);

$adresse = new AdresseProModel;
$adresse = $adresse->findAdresse($boutique->id);

$annonce = new AnnonceModel;
$annonce = $annonce->findAnnoneProLimit($boutique->id);

$user = new UserModel;
// On vérifie en BDD d'un utilisateur à un email qui correspond
$user_exist = $user->findOneByEmail(strip_tags($_POST['email']));
if (!empty($user_exist)) {
    // l'utilisateur existe et on hydrate l'user avec user_exist
    $user = $user->hydrate($user_exist);
}
```

Développer la partie back-end d'une application web ou web mobile

Publier une annonce

Traitement des informations remplies par l'utilisateur :

```
public function ajouterPar()
{
    // On vérifie que le formulaire est complet
    if (Form::validate($_POST, ['titre', 'description', 'prix', 'stock'])) {
        // Le formulaire est complet
        // On se protège des failles xss

        var_dump($_POST);
        $titre = strip_tags($_POST['titre']);
        $description = strip_tags($_POST['description']);
        $prix = (int) $_POST['prix'];
        $poids = (int) $_POST['poids'];
        $stock = (int) $_POST['stock'];

        // On instancie notre modele et la classe boutique_par
        $annonce = new AnnonceModel;
        $boutique = new BoutiqueParticulierModel;
        $boutique = $boutique->findBoutiqueByUser($_SESSION['user']['id']);

        // On hydrate l'objet
        $annonce->setTitre($titre)
            ->setCategorie_id($_POST['categorie'])
            ->setDescription($description)
            ->setPrix($prix)
            ->setPoids($poids)
            ->setStock($stock)
            ->setBoutique_particulier_id($boutique->id);

        // On crée l'annonce en BDD
        $annonce->create();

        // On retour chercher l'annonce créée pour obtenir son id
        $annonce = $annonce->findAnnonceByPar($boutique->id);

        // On instancie la classe photo
        $photo = new PhotoAnnonceModel;

        //Taille max de la photo
        $tailleMax = 2000000;
    }
}
```

```

// Extensions valides pour la photo
$extensionValides = ['jpg', 'jpeg', 'gif', 'png'];

if ($_FILES['photo_principale']['size'] <= $tailleMax) {
    // La taille du fichier est bien inférieure à ce que l'on demande
    // On vérifie l'extension
    $extensionUpload = strtolower(substr(strrchr($_FILES['photo_principale']['name'], '.'), 1));
    if (in_array($extensionUpload, $extensionValides)) {
        // La taille et l'extension de la photo sont valides

        // Chemin et nom du fichier que l'on va enregistrer
        $chemin = 'public/img/annonce/' . $annonce->id . '.' . $extensionUpload;

        // On enregistre le fichier grâce à move et $resultat = false ou true
        $resultat = move_uploaded_file($_FILES['photo_principale']['tmp_name'],
$chemin);

        if ($resultat) {
            // On hydrate l'objet
            $photo->setAnnonce_id($annonce->id)
                ->setPhoto($annonce->id . '.' . $extensionUpload);
            // On crée insert la photo en BDD
            $photo->create();
        } else {
            $_SESSION['erreur'] = "Erreur durant l'importation du fichier";
            header('location: ' . ACCUEIL . 'annonce/ajouterpar');
            exit;
        }
    } else {
        $_SESSION['erreur'] = "Votre photo de profil doit être au
format jpg, jpeg, gif ou png";
        header('location: ' . ACCUEIL . 'annonce/ajouterpar');
        exit;
    }
} else {
    $_SESSION['erreur'] = "Votre photo de profil ne doit pas dépasser 2 mo";
    header('location: ' . ACCUEIL . 'annonce/ajouterpar');
    exit;
}

// On redirige
$_SESSION['success'] = "votre annonce à été enregistrée avec succès";
header('location: ' . ACCUEIL . 'boutiqueAccueil/accueilPar');

```

```
}
```

Veille et sécurité

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 injections. Je vérifie tout ce qui provient de l'utilisateur en vérifiant que les valeurs reçues sont bien celles demandées.

Veilles

Dans le cadre de ce projet je me suis intéressé à la fonction PHP htmlspecialchars() afin de bien comprendre son fonctionnement et son utilité.

Cette fonction est utilisée de différentes manières :

- Avant un enregistrement en base de données, ce qui permet d'éviter que des informations contenant des scripts peuvent nuire à l'intégrité des données.
- Avant un affichage, c'est lors des affichages de données que les attaques XSS peuvent faire du dommage.

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 » :

- Maquetter une application.
- 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'émoticones
- 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 bouton 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

- Grâce 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 les wireframes et ensuite le modèle de la BDD.

Je me suis occupé des tâches marquées ci-dessus, Thuc Nhi et Denis étant habitués à travailler ensemble se sont partagés le 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**

ICI SERA L'ARCHITECTURE DU PROJET QUI ME RESTE A CREER

Spécifications techniques

J'ai utilisé le PHP ainsi que la POO pour toute la partie back de mon projet.

J'ai aussi utilisé Javascript et AJAX pour rendre mes pages totalement dynamiques, et pour optimiser au maximum l'expérience de l'utilisateur.

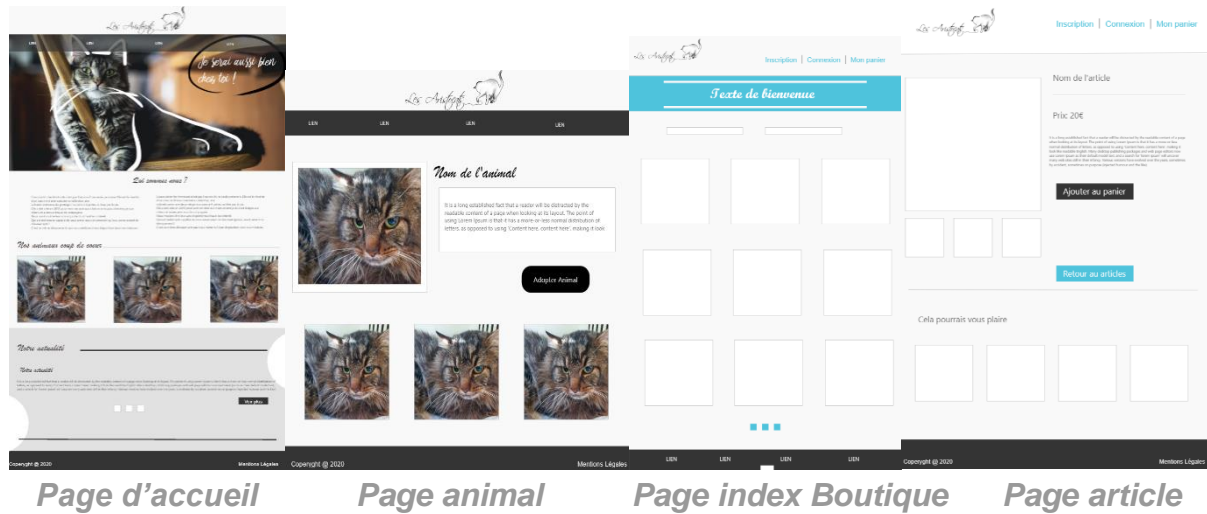
Pour la mise en place des dons et des paiements,

Front-end

Maquetter une application

Nous avons réalisé des wireframes

CECI SERA REMPLACÉ PAR LES WIREFRAMES QUE NOUS AVONS CREER



Développer une interface utilisateur web dynamique

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 clic sur un bouton il peut faire passer son post en mode story (durée de vie 24h).

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

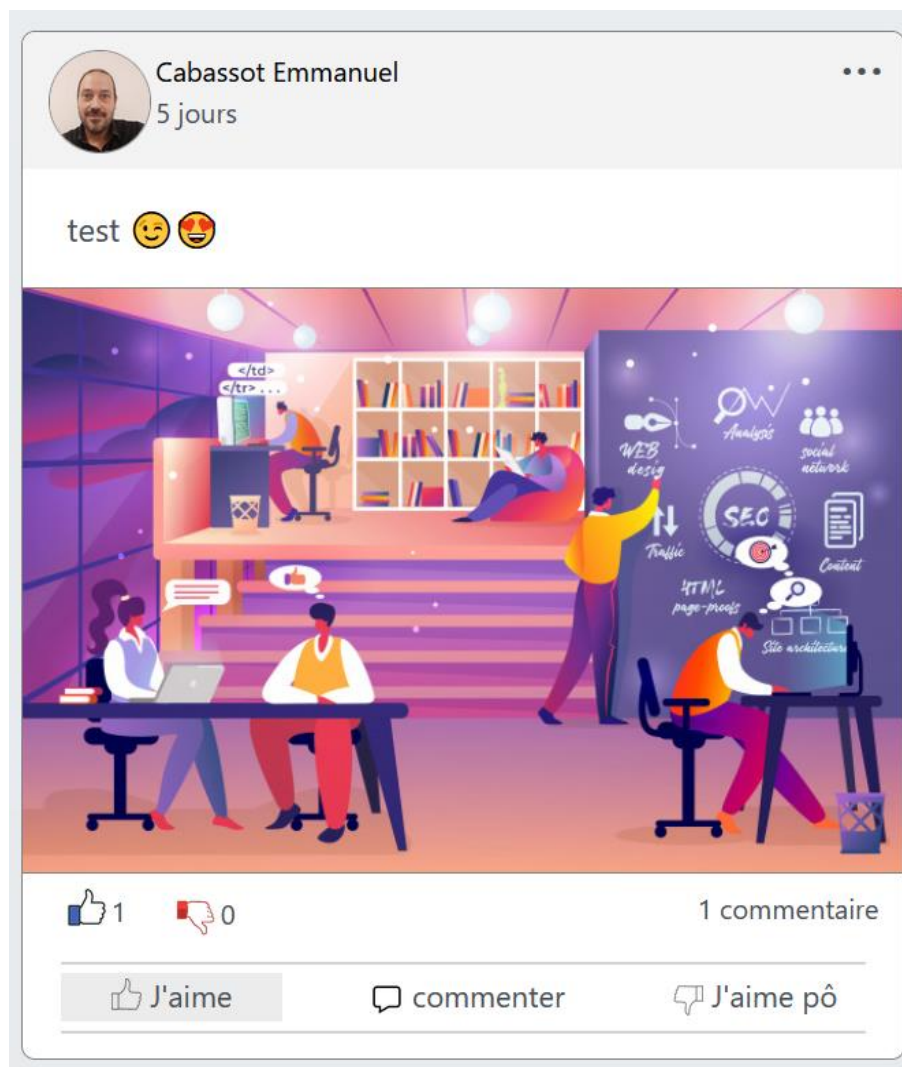
likes, dislikes, le nombre de commentaires, les commentaires, les créateurs des commentaires...

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.



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

social-network/api/controllers/postShow.php

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

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

```
// 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

```
// 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 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'];

// 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 ?

```

// 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

 0  1

6 commentaires

 J'aime

 commenter

 J'aime pô



Ecrivez un commentaire ...



Cabassot Emmanuel

9 jours

test pour les commentaires



Cabassot Emmanuel

9 jours

sdfkljqskjdmq



Cabassot Emmanuel

9 jours

dslmfksmlqksqlm

kdksfmksqù



Cabassot Emmanuel

9 jours

edflmckalksa

```
// 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 info de celui qui a commenté

```
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 et si l'utilisateur à 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 intégrer \$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.

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.

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

S

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

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

```

        </img> 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 les valeurs de la fonction onclick et enregistrent en BDD l'action effectuée. Elles sont liées au onclick ci-dessus.

```

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

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

            Like = 1
            Dislike = 0

            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      + ")")

            sectionLike.classList.add("validate")
            countLike = document.querySelector("#likeCount" + post_id + " span")
            valueLike = countLike.textContent
            valueLike++
            countLike.textContent = valueLike
        }

        if (dislike === 1) {
            Like = 1
            Dislike = -1

            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 + ")")

        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) {
        Like = -1
        Dislike = 0

        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 + ")")

        sectionLike.classList.remove("validate")
        countLike = document.querySelector("#likeCount" + post_id + " span")
        valueLike = countLike.textContent
        valueLike = valueLike - 1
        countLike.textContent = valueLike
    }

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

```

```

    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)
        }
    }
}

```

La requête AJAX envoie le tout à likePost.php qui devient un traitement incroyablement simple. Elle ajoute en BDD.

```

$like = new Like($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);
}

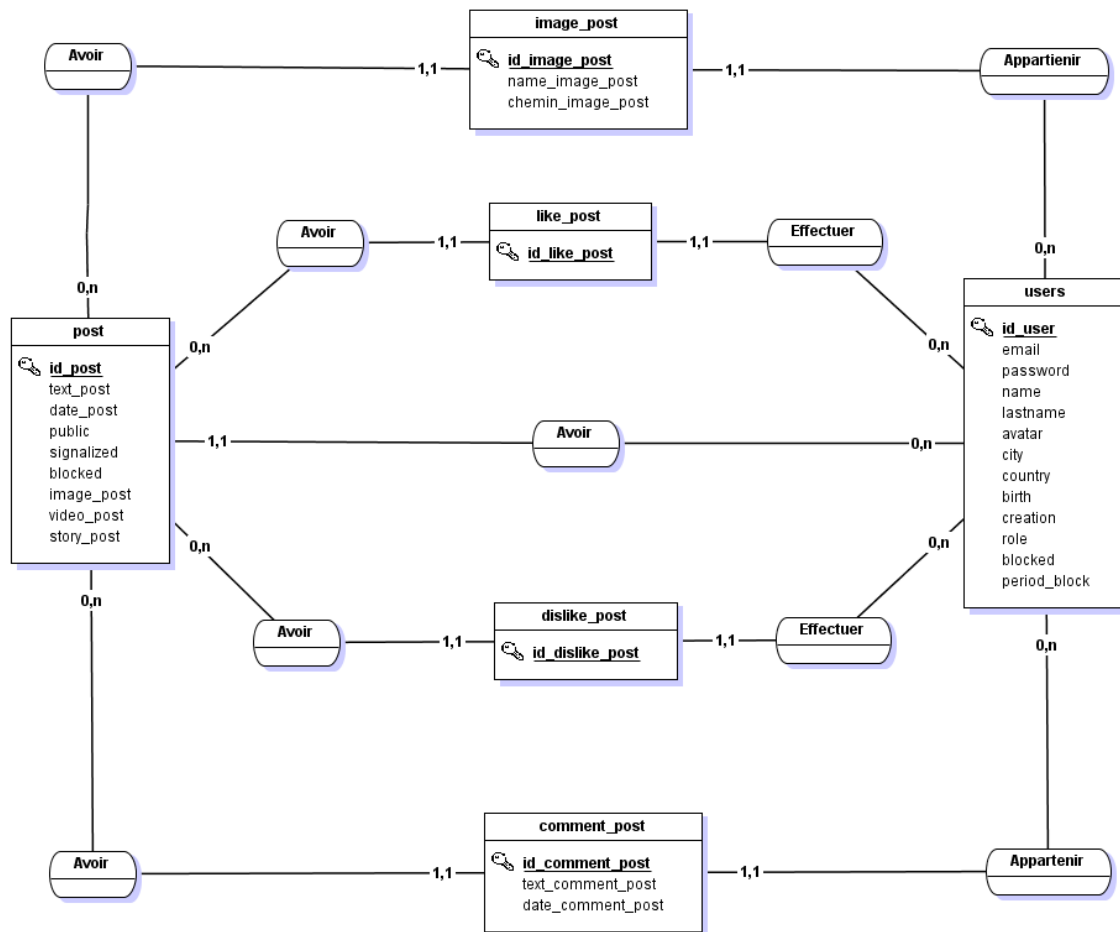
```

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 innnoDB 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.

```
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.

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

    // 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);
}

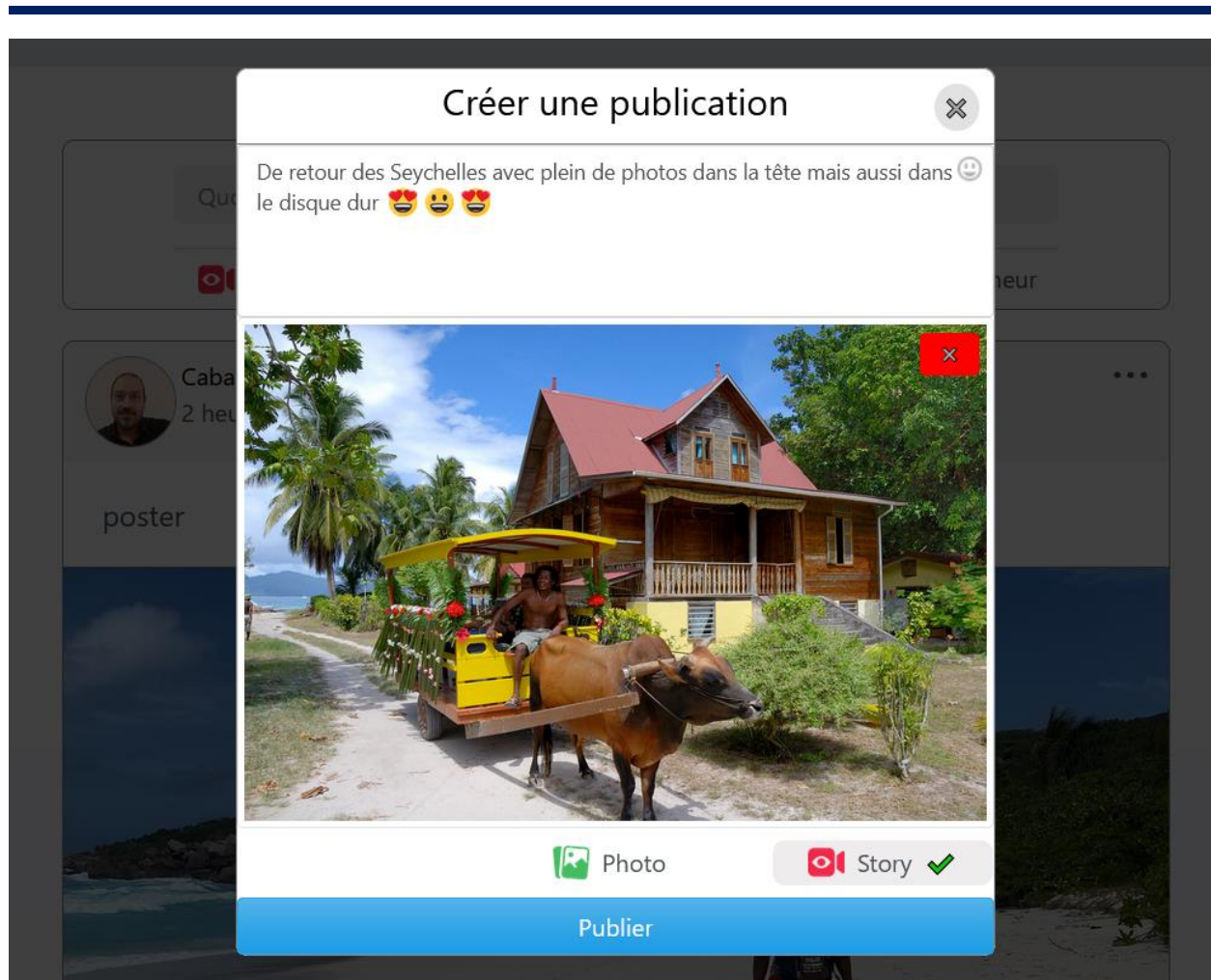
if ($data->bdDislike == '1') {
    $dislike->addDisLike($id_post,$id_user);
}

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

Dans le projet Boutique de boutiques j'ai utilisé les fonctions d'héritage de classe (dossier de professionnalisation)

Développer la partie back-end d'une application web ou web mobile

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);
```

La requête AJAX envoie les infos vers la page de traitement *social-network/api/controllers/postSend.php*.

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

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

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

On instancie la class 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();
```

On crée un nouveau répertoire portant le nom de l'id du post .

On instancie la classe model PhotoPost

```
// 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`.

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 fichier 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);  
        }  
    }  
}
```

Elaborer et mettre en oeuvre des composants dans une application de gestion de contenu ou e-commerce

Partie administrateur

Il fallait donner la possibilité aux administrateurs d'avoir une remontée des posts et commentaires violents. 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.

Pour 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 ▼	<input type="text" value="non"/>	01/01/0001 ⓘ	<input type="button" value="Submit"/>
33	Lou	Cabassot	lou@hotmail.fr	admin	User ▼	<input type="text" value="oui"/>	01/01/0001 ⓘ	<input type="button" value="Submit"/>
32	Emmanuel	Cabassot	emmanuel.cabassot@laplateforme.io	user	User ▼	<input type="text" value="oui"/>	01/01/0001 ⓘ	<input type="button" value="Submit"/>


Liste des postes signalés :0

- supprimer le post
- désignaler le post
- bloquer le créateur du post pour une période donnée



Cabassot Emmanuel
2 heures

De retour des Seychelles avec des photos plein la tête mais aussi dans le disque dur 🤔🤔🤔




✓ Supprimer post

✓ Désigner post

Utilisateur


Blocked	Période
non	01 / 01 / 0001 ✕
Valider	



Cabassot Emmanuel
2 heures

La Nouvelle-Zélande

Encore une île? On se refait pas.... 🤔



Supprimer post

✓ Désigner post

Utilisateur

Blocked	Période
non	01 / 01 / 0001 ✕
Valider	

Liste des commentaires signalés :

- supprimer le commentaire
- désigner le commentaire
- bloquer le créateur du commentaire pour une période donnée

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

```
response.forEach(post => {
    supprimePost = listPosts.querySelector("#supprime-adminOption" + post.id_post)
    deleteSignalPost = listPosts.querySelector("#deleteSignal-adminOption" + post.id_post)

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

        this.classList.toggle("valide")

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

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

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

```

submit = listPosts.querySelector("#submit" + post.id_post)
submit.addEventListener("click", function (e) {
  supprimePost = listPosts.querySelector("#supprime-adminOption" +
    post.id_post)
    if (supprimePost.classList.contains("valide")) {
      supprime = 'oui'
    }else {
      supprime = 'non'
    }

  deleteSignalPost = listPosts.querySelector
    ("#deleteSignal- adminOption" + post.id_post)
  if (deleteSignalPost.classList.contains("valide")) {
    deleteSignal = 'oui'
  }else {
    deleteSignal = 'non'
  }

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

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

  data = {
    supprime: supprime,
    deleteSignal: deleteSignal,
    blocked: blocked,
    date: date,
    post: post.id_post,
    user: post.id_user
  }
  if (supprime == 'oui' || deleteSignal == 'oui') {
    postHidden = listPosts.querySelector("#showOnePost" +
      post.id_post)
    postHidden.classList.add("hidden")
  }

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

  xhr.addEventListener("readystatechange", function() {

```

```

        if (xhr.readyState === 4 && xhr.status == 200) {
            console.log("ca marche")
        }
    })
}
});

```

En appuyant sur valider on envoie les informations pour traitement

social-network/api/controllers/modifPost.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"));

$user = new User($db);

$user->blocked = $data->blocked;
$user->period_block = $data->date;

$user->updateUserBlockedPost($data->user);

$post = new Post($db);

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

if ($data->deleteSignal == 'oui') {
    $post->deleteSignalPostById($data->post);
}

```

Veille et sécurité

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.

Veilles

Dans le cadre de ce projet je me suis intéressé à la fonction PHP htmlspecialchars() afin de bien comprendre son fonctionnement et son utilité.

Cette fonction est utilisée de différentes manières :

- Avant un enregistrement en base de données, ce qui permet d'éviter que des informations contenant des scripts peuvent nuire à l'intégrité des données.
- Avant un affichage, c'est lors des affichages de données que les attaques XSS peuvent créer des dommages.

Traduction d'une recherche

Lors du développement de ce projet, j'ai à de nombreuses reprises fait des recherches en anglais. Effectivement, beaucoup plus de sources sont disponibles en anglais. Je vous présente donc un extrait du site de stripe que j'ai traduit dans le cadre du développement de mon module de paiement :

The Stripe API is organized around REST. Our API has predictable resource-oriented URLs, accepts form-encoded request bodies, returns JSON-encoded responses, and uses standard HTTP response codes, authentication, and verbs.

You can use the Stripe API in test mode, which does not affect your live data or interact with the banking networks. The API key you use to authenticate the request determines whether the request is live mode or test mode.

The Stripe API differs for every account as we release new versions and tailor functionality. These docs are customized to your version of the API and display your test key and test data, which only you can see.

Stripe est une API organisée autour de REST. Notre API a des ressources orientées URL, elle accepte les données codées des formulaires et retourne une réponse au format JSON et utilise des réponses au standard HTTP, authentification et vocabulaires.

Vous pouvez utiliser l'API Stripe en mode test qui n'affectera pas vos données et n'interagit pas avec les réseaux bancaires. La clé API est utilisée pour identifier que les requêtes sont en mode normal ou test.

L'API Stripe est différente pour chaque compte avec de nouvelles versions et fonctionnalités. Les documents sont personnalisés avec votre version de l'API et affiche votre clé test et vos données mais vous êtes le seul à pouvoir les voir.