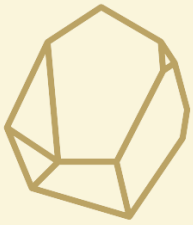
TITRE RNCP NIV III

DÉVELOPPEUR.SE WEB ET WEB MOBILE

DOSSIER DE SOUTENANCE



Cave of Wonders

Cave of Wonders - Eternity in Luxury
Site web de petites annonces de luxe

Présenté par Huong-Mây NGUYEN-PHUOC
Étudiante de l'école la Plateforme_

Juillet 2021

TABLE DES MATIÈRES

1 / CADRE DU PROJET

1 - Présentation de la formation à la Plateforme_

La Plateforme_ est une jeune école du numérique située à Marseille. Elle est membre de la **Grande École du Numérique**. Elle propose plusieurs parcours autour des métiers du digital - Intelligence Artificielle, Cybersécurité, Coding School.

La formation **Coding School**, que je suis cette année, est entièrement financée par les collectivités et les entreprises de la région. Elle s'adresse aux personnes de tout âge, avec ou sans le bac, et s'adapte aux besoins de chacun.

La pédagogie y est **active et inductive**. Nous alternons entre courtes périodes de Runtrack - où nous découvrons un nouveau langage au travers de tutoriels live, d'exercices et de How To - et périodes plus longues de Project Pool - où nous développons des projets de sites web complets.

2 - Résumé du projet

Cave of Wonders est une application web qui a été réalisée dans le cadre de ma formation, en collaboration avec ma camarade Alicia Cordial. Il s'agit du dernier projet de l'année et il a été développé entre les mois de mai et juillet 2021.

Il consiste en la réalisation d'un **site web dynamique de petites annonces**, destiné aux particuliers et mettant en relation l'offre et la demande locales. Les annonces sont consultables sans inscription mais il faut être connecté pour envoyer un message au vendeur.

Parmi les exigences du sujet, on retrouve :

- la gestion de comptes utilisateur et d'espaces client et vendeur
- la publication et modification d'annonces
- une boutique, avec barre de recherche, catégories et fiches produits
- une messagerie interne
- un espace d'administration permettant de modérer les utilisateurs et les annonces

Le thème étant libre, ma partenaire de travail et moi avons choisi de créer le concept *Cave of Wonders*, un site permettant de **chiner des objets de valeur**. Il vise uniquement la France métropolitaine.

Sans jamais perdre de vue le cahier des charges, nous avons développé l'intégralité du projet, depuis les toutes premières maquettes jusqu'à l'hébergement du site et son référencement.

Le site est intégralement **responsive** et l'ensemble des fonctionnalités sont accessibles depuis mobile.

Ce projet a pour objectif de synthétiser l'ensemble des compétences techniques acquises tout du long de la formation. Il a été soutenu lors d'un pitch et a permis de valider ma première année.

3 - Liste des compétences couvertes par le projet

Le projet couvre **l'ensemble des compétences** définies par le Référentiel Emploi Activités Compétences.

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

- 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 » :

- Créer une base de données.
- Développer les composants d'accès aux données.
- Développer la partie back-end d'une application web ou web mobile.

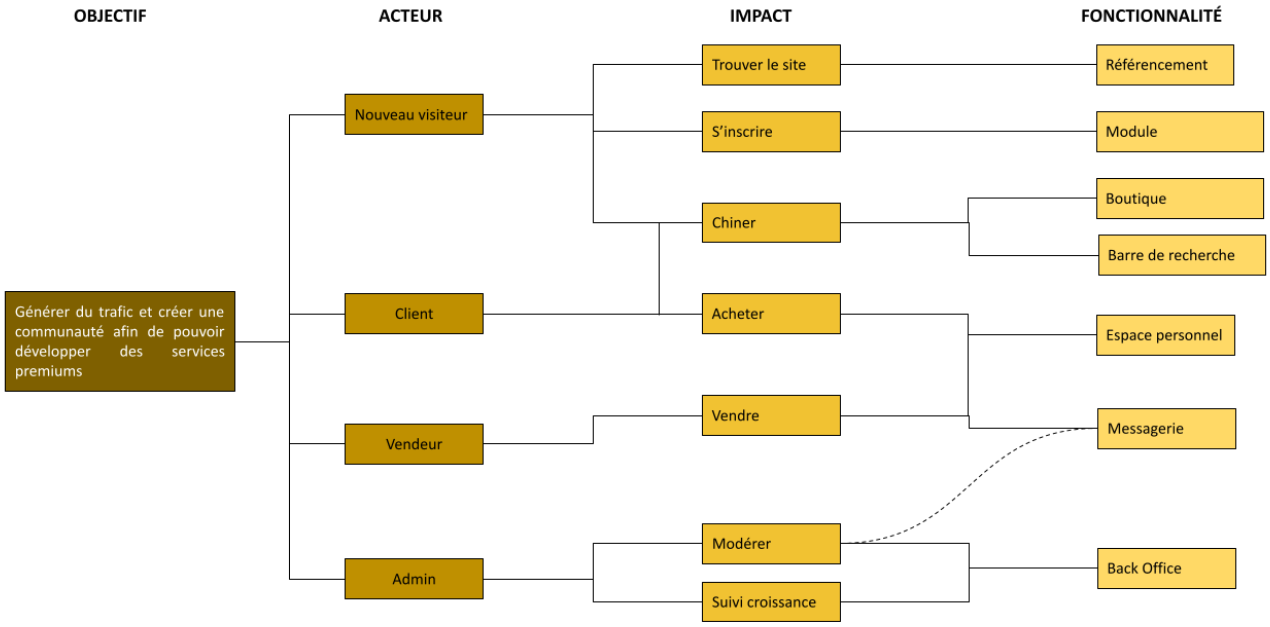
2 / CAHIER DES CHARGES

1 - Spécifications fonctionnelles

a - Périmètre fonctionnel

L'objectif de ce site est d'agrandir sa communauté de membres pour, à terme, proposer des services payants et toucher une commission sur les ventes.

Chaque utilisateur du site a un rôle, des besoins et des droits spécifiques. Les **fonctionnalités** à implémenter **découlent des actions** que chaque acteur peut effectuer.



b - Fonctionnalités détaillées

Les annonces sont consultables par tous les utilisateurs du site.

Le système de messagerie n'est disponible que pour les utilisateurs ayant un compte.

Les utilisateurs du site ont **4 statuts** distincts : **anonyme, acheteur, vendeur, admin**.

BOUTIQUE	
Fonctionnalités	**Détails**
Rechercher un produit	• prix (type fourchette) • localisation (code postal) • nom • catégorie • autocomplétion
Rechercher un vendeur	• identifiant
Consulter le profil d'un vendeur	• nombre de ventes • note • tous les articles en vente • date d'inscription • localisation • envoyer un message
Consulter la fiche produit	• description • catégorie • photo • prix, ouvert aux négociations • état • localisation • accès au profil vendeur • envoyer un message • signaler l'annonce

ESPACE PERSONNEL	
Fonctionnalités	**Détails**

S'inscrire	• statut (vendeur ou acheteur) • login et email non utilisés • mot de passe sécurisé • code postal
Se connecter	• login ou email existant • mot de passe correspondant
Modifier son profil	• changer de statut • updater ses infos personnelles
Voir sa messagerie	• sous forme de conversations • répondre à un message
Ajouter une annonce	• 5 annonces max • suggestion de nouvelle catégorie (à valider par l'admin)
Gérer les annonces en vente	• modifier les détails • marquer comme vendu (à un contact) • supprimer l'annonce
Gérer son historique d'achat / de vente	• détails d'une vente/achat • supprimer l'annonce
Noter un vendeur	• après achat

BACK OFFICE	
Fonctionnalités	**Détails**
Gérer les utilisateurs	• filtre (statut) • détails d'un utilisateur • supprimer • contacter
Gérer les catégories	• updater • ajouter • supprimer si vide • voir les articles de la catégorie • supprimer un article
Modérer les annonces (signalées 2 fois, catégorie 'autre')	• filtre (signalé ou nouvelle catégorie) • rendre visible

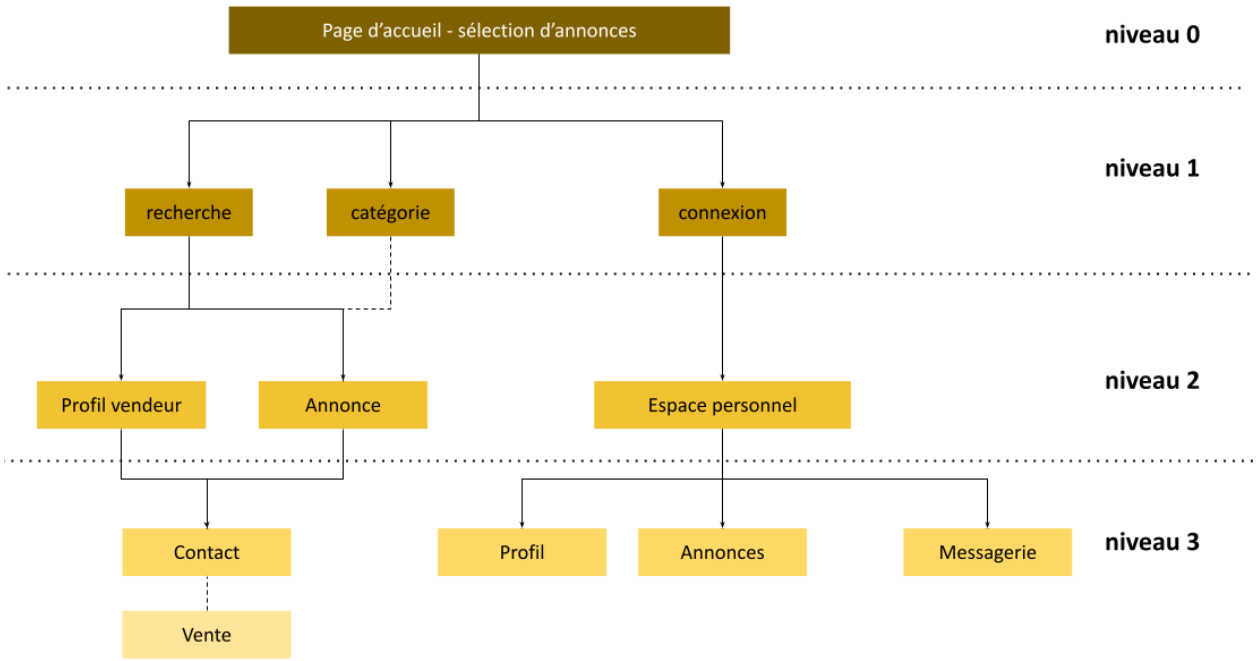
	• supprimer • contacter le vendeur

c - Arborescence du site

Le site se décompose en 3 sections principales :

- La page d'accueil, qui inclut une barre de recherche et une sélection d'annonces
- La boutique, qui donne accès à des catégories, des fiches produits et le profil des vendeurs
- L'espace utilisateur, avec une messagerie, le profil personnel et la gestion des annonces

Il comporte aussi un back-office, uniquement accessible par un administrateur.



2 - Spécifications techniques

a - Choix technologiques

i - Langages informatiques

HTML5 et CSS3 : pour la structure de site et le style.

PHP7 /JS/JQuery : pour les interactions utilisateurs et la manipulation des données.

ii - Responsive

Pour gagner en temps et en efficacité, le site utilise le framework front-end Materialize CSS.

iii - Framework / CMS

Le site est développé sans framework ni CMS.

iv - Serveur web

Apache 2.4.46

v - Base de données

Le SGBD est mySQL, par le biais de l'interface phpMyAdmin.

La base de données s'appelle « caveOfwonders ».

vii - IDE

PhpStorm

viii - Versioning

La gestion de versions est assurée par le logiciel Git, par le biais de l'interface graphique GitHub Desktop. Le projet web est sauvegardé sur le cloud grâce à Github.

Des branches annexes sont créées lors du développement de nouvelles fonctionnalités. Une fois stabilisées et testées, elles sont fusionnées avec la branche principale.

b - Hébergement

Le site est hébergé sur **Plesk**. Il sera disponible à l'adresse suivante :

https://huong-may-nguyen-phuoc.students-laplateforme.io/pages/demo/lbp

c - Accessibilité

i - Compatibilité navigateur

Le site est testé sur et compatible avec les navigateurs suivants :

- Internet Explorer
- Mozilla Firefox
- Google Chrome
- Safari

ii - Adaptabilité et responsivité

Le site assure une navigation optimale pour toutes les résolutions d'écran et sur tous types d'appareil :

- Téléphones mobiles
- Tablettes
- Ordinateur portables
- Ordinateur de bureau

3 / GESTION DE PROJET

1 - Calendrier du projet

a - Dates butoirs

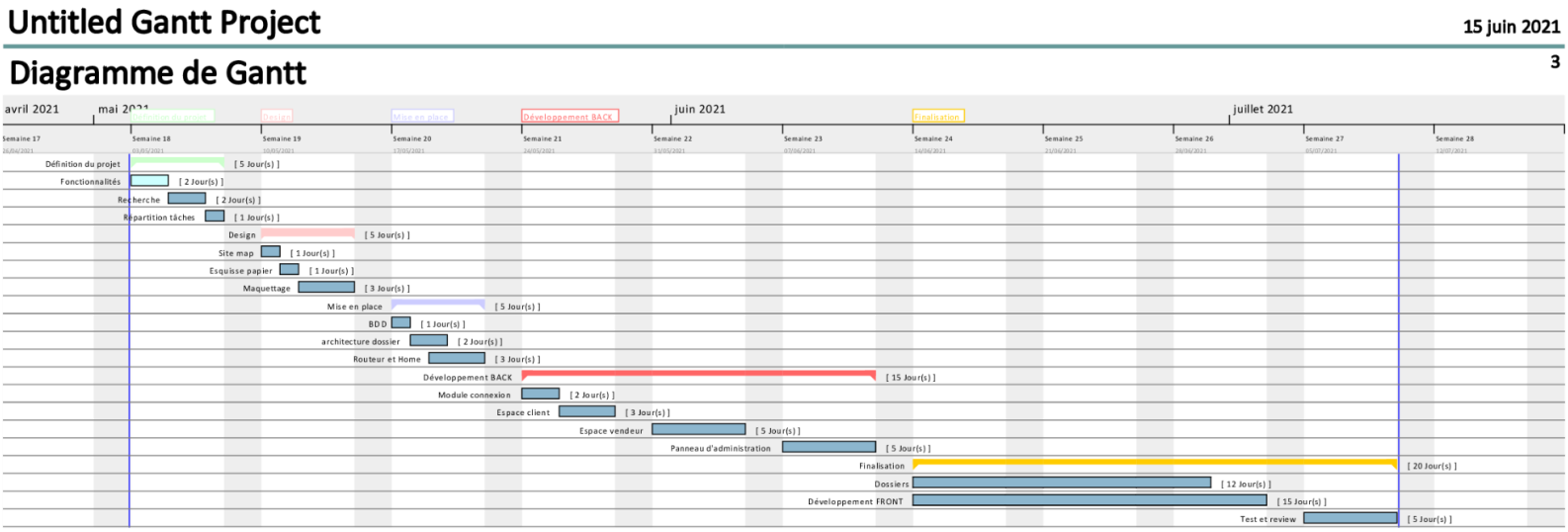
Le projet est débloqué le 3 mai 2021 et doit être terminé avant la fin de l'année scolaire, le 31 juillet 2021. Il doit donc être entièrement développé en l'espace de **3 mois**.

b - Planification des tâches

Nous avons créé un **diagramme de Gantt** afin de mieux nous organiser dans le temps. Nous avons distingué plusieurs grandes étapes, que nous avons divisées en tâches.

PLANNING		
Tâche	**Détails**	**Nombre de jours**
Définition du projet		5
Fonctionnalités	lister toutes les fonctionnalités à développer	2
Recherche	trouver de l'inspiration, des pistes	2
Répartition des tâches	diviser les tâches intelligemment pour ne pas créer de conflit	1
Design		5

Site map	créer l'arborescence du site	1
Esquisse papier	imaginer les vues de façon grossière	1
Maquettage	maquetter toutes les interfaces	3
Mise en place		5
BDD	schéma et création de la bdd	1
Architecture dossier	squelette du projet, partage du projet sur github	2
Routeur et Home	création du système de routage et de la vue home	3
Développement back		15
module connexion	connexion, inscription	2
espace client	messagerie, modification du profil	3
espace vendeur	gestion des annonces	5
panneau d'administration	gestion des utilisateurs, catégories, modération	5
Finalisation		20
dossiers	premier jet du dossier de Projet et du dossier Professionnel	12
développement front	style CSS et implémentation de Materialize	15
test et mise en production	revue complète du projet, test des fonctionnalités et hébergement en ligne	5



2 - Répartition des tâches

Ma camarade Alicia Cordial s'est occupée de toute la partie boutique et de la barre de recherche.

Je me suis chargée de **l'espace utilisateur** (acheteur et vendeur), de la messagerie, de la gestion des annonces et de **l'espace d'administration**.

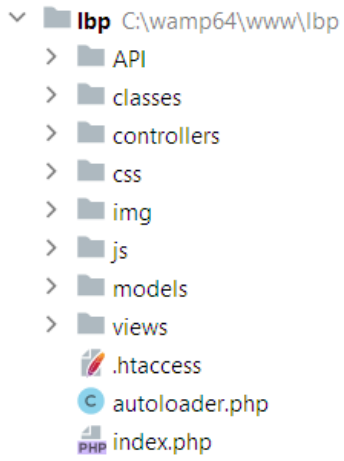
Le style et l'intégration de Materialize ont été réalisés de concert.

3 - Structure des dossiers

Avant de commencer à développer nos fonctionnalités de façon séparée, nous avons mis en place l'arborescence de notre projet pour partir sur une **base commune**.

Nous nous sommes **inspirées** de l'architecture **Modèle-Vue-Contrôleur** afin de diviser du mieux possible notre HTML, nos requêtes SQL et les modifications de l'affichage suite à des actions de l'utilisateur.

Nous avons créé des dossiers différents pour stocker les images, les scripts, le style, les classes et le traitement API. A la racine du projet, nous avons l'index, l'autoloader et le fichier htaccess.



4 / RÉALISATIONS FRONT-END

1 - Maquettage

Nous avons d'abord étudié des sites de vente d'objets d'occasion et des sites de galeries d'art pour mieux cerner le design que nous voulions donner à notre projet.

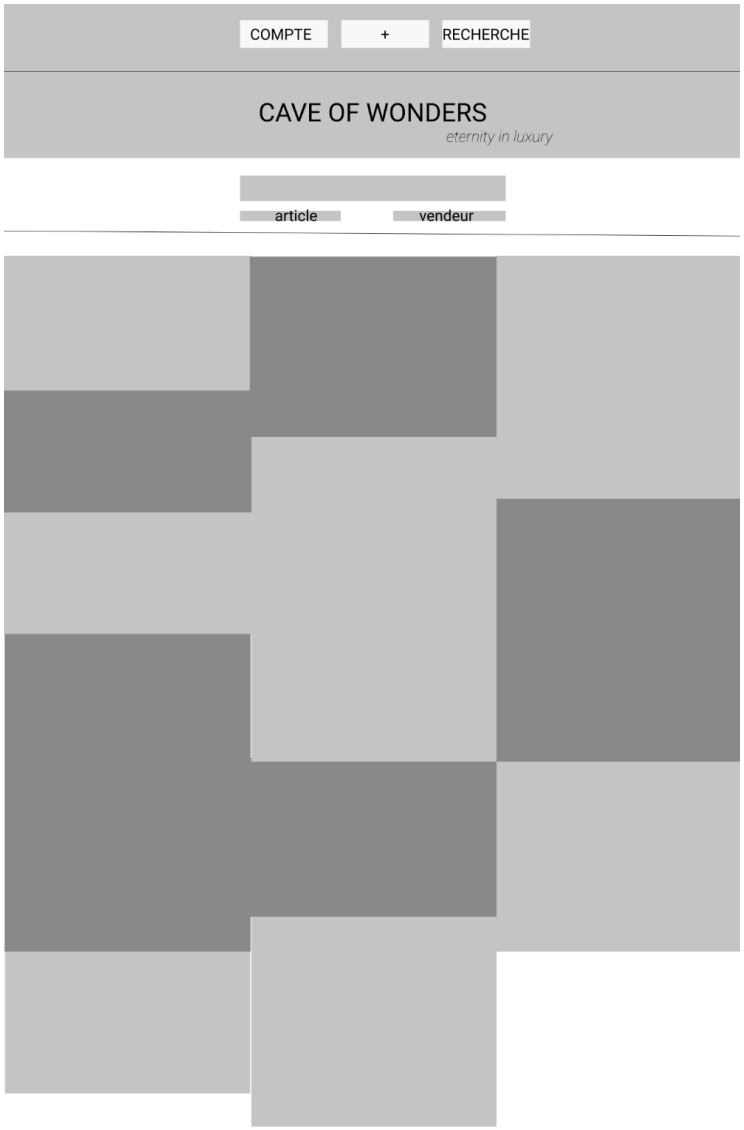
Nous avons aussi créé un **logo** grâce à **jimdo**. Bien que Jimdo soit à l'origine un système de gestion de contenu (CMS), il offre aussi la possibilité de générer un logo facilement et gratuitement.

Après une première esquisse sur papier, nous avons réalisé des maquettes grâce à l'outil de prototypage en ligne **Figma**. Nous avons travaillé simultanément sur le projet depuis nos deux ordinateurs.

a - Vignette globale du projet

b - Maquette des écrans principaux

i - Page d'accueil

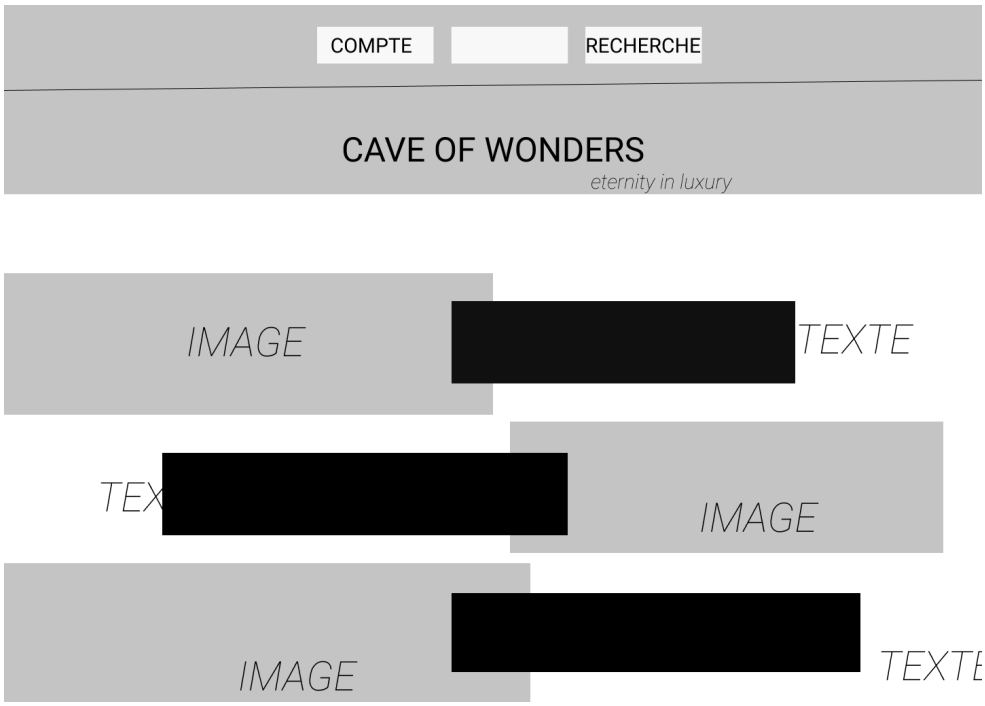


C'est la porte d'entrée du site. On peut y voir :

- la barre de navigation. Elle donne accès à l'espace personnel et à la création d'une nouvelle annonce. Elle comporte une barre de recherche à autocomplétion.
- Le titre et le slogan du site.
- Une barre de recherche avancée, qui permet de trouver une annonce ou un vendeur à l'aide de filtres.
- Une mosaïque d'annonces sélectionnées de façon random.
- Le pied de page, avec le copyright, l'accès à des pages statiques secondaires et les réseaux sociaux.

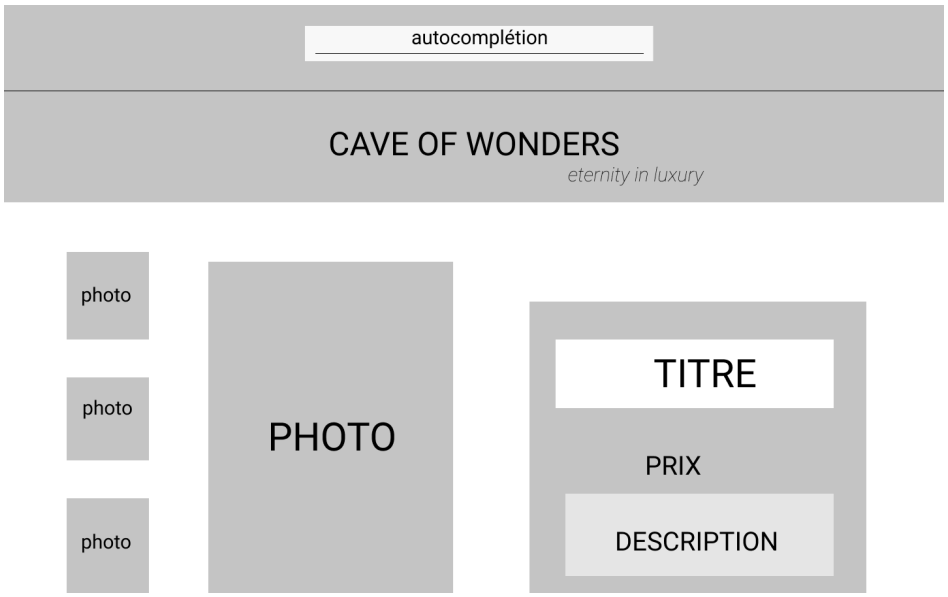
ii - Résultats de recherche

C’est la page qui affiche les annonces correspondant à la recherche effectuée.



iii - Fiche produit

En cliquant sur une annonce, on accède à la fiche détaillée de l’article en vente. Il y a une photo, les informations essentielles, un bouton pour contacter directement le vendeur et un bouton pour accéder à son profil.



iv - Profil public vendeur

Le profil du vendeur met en avant ses articles en vente, sa date d'inscription et son évaluation. Il y a aussi un bouton permettant d'envoyer un message.

2 - Exemples d’interfaces utilisateur

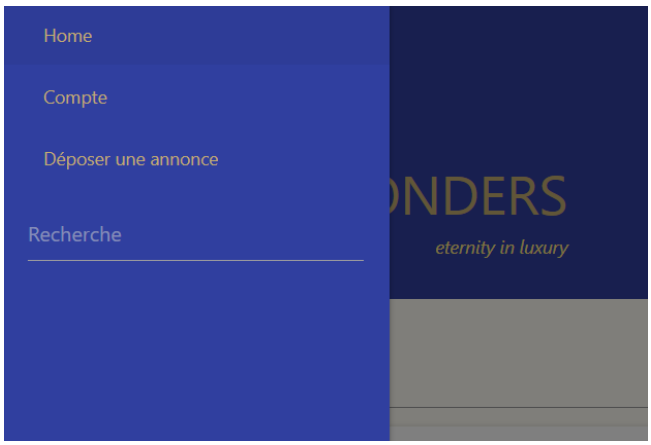
A partir du style minimal de Materialize, j’ai apporté des modifications et personnalisation.

a - Vue de l’accueil

b - Vue espace client

c - Barre de navigation

Nous avons utilisé la navbar par défaut de Materialize et l’avons modifiée avec CSS.



d - Modal nouveau message

J'ai utilisé la librairie Javascript **jQuery Modal** pour créer le pop-up de nouveau message. Il faut lier l'id de la div à faire apparaître avec le lien href du bouton déclencheur.

```
<div id="ex1" class="modal">
    <div id="nameDestinataire"></div>
    <form id='newMessage'>
        <input placeholder='votre message' required>
        <button type='submit'>Envoyer</button>
    </form>
    <div id="infoMessage"></div>
    <a href="#" rel="modal:close">Close</a>
</div>
```

Le HTML de la modale est écrit dans la vue.

```
><a id ='" + article.id_vendeur + "' class='contactUser' href='#ex1' rel='modal:open'>Contacter le vendeur</a>
```

Le bouton déclencheur a été ajouté par javascript.

3 - Accessibilité

a - Responsivité

La responsivité a été testée grâce à l'outil du navigateur mais aussi sur appareils réels (téléphone et tablette).

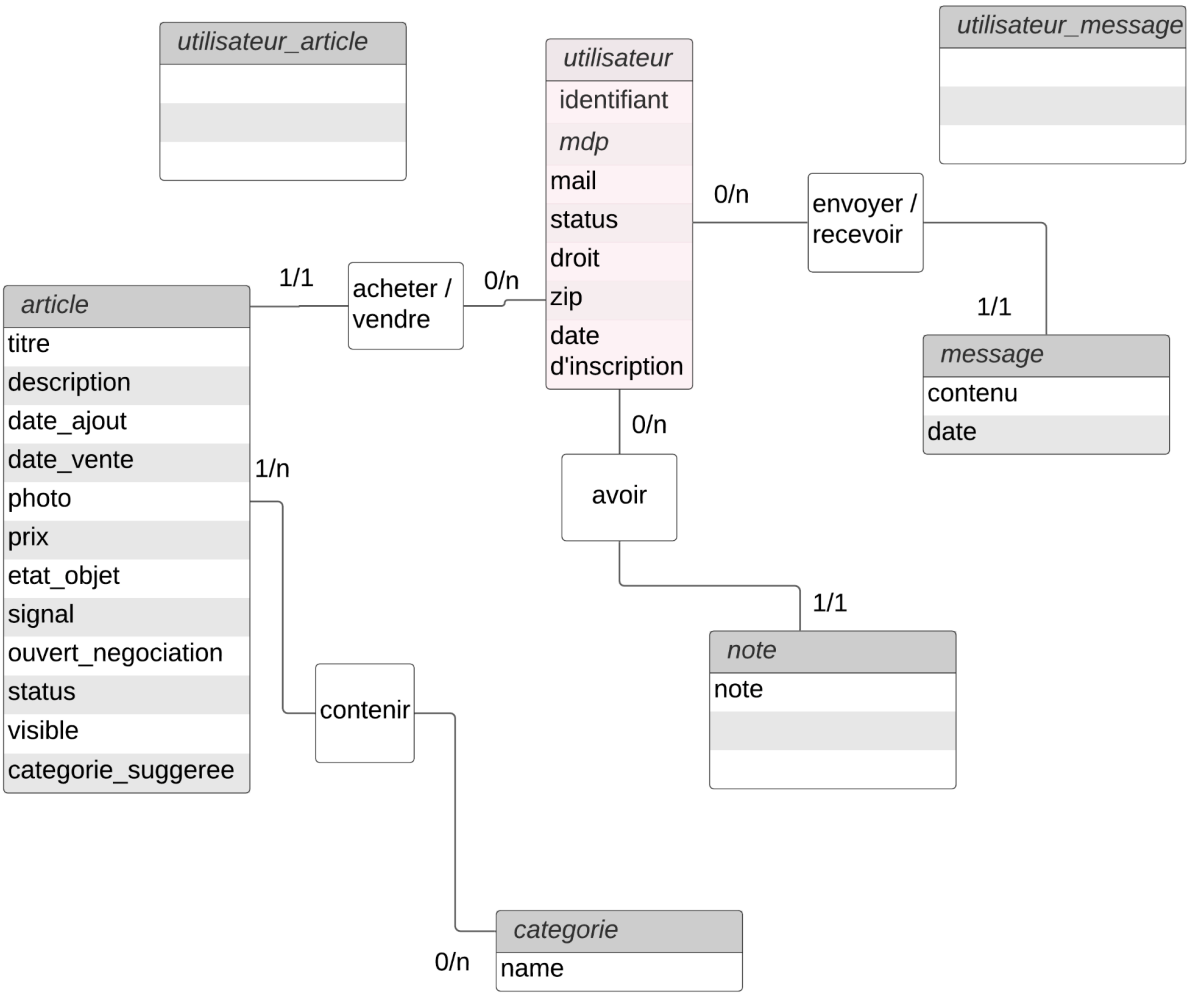
b - Adaptabilité

Le site s'adapte et s'agence différemment selon le format de la fenêtre.

5 / RÉALISATIONS BACK-END

1 - Base de données

a - Modèle Conceptuel de la base de données (MCD)



b - Création de la base de donnée

Nous avons utilisé phpMyAdmin pour créer la base de données du projet, que nous avons appelée “caveOfwonders”. Elle contient 7 tables.

Nous avons utilisé le moteur de stockage innoDB pour ajouter des clés étrangères et des contraintes, qui garantissent l’intégrité des données.

Ci-dessous, des exemples de scripts de génération de table :

```sql
CREATE TABLE IF NOT EXISTS `utilisateur` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `identifiant` varchar(255) NOT NULL,
  `mdp` varchar(255) NOT NULL,
  `mail` varchar(255) NOT NULL,
  `status` enum('client','vendeur','supprimé') NOT NULL,
  `droit` int(1) DEFAULT '0',
  `zip` int(5) NOT NULL,
  `date_inscription` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8;


CREATE TABLE IF NOT EXISTS `article` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `titre` varchar(255) NOT NULL,
  `description` text NOT NULL,
  `date_ajout` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `date_vente` datetime DEFAULT NULL,
  `photo` varchar(255) DEFAULT NULL,
  `prix` int(20) NOT NULL,
  `etat_objet` enum('neuf','très bon état','bon état') NOT NULL,
  `signal` int(1) DEFAULT NULL,
  `id_categorie` int(1) DEFAULT NULL,
  `ouvert_negociation` enum('oui','non') NOT NULL,
  `status` enum('disponible','vendu') NOT NULL,
  `id_vendeur` int(11) NOT NULL,
  `id_acheteur` int(11) DEFAULT NULL,
  `visible` int(255) NOT NULL DEFAULT '1',
  `categorie_suggeree` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `acheteur` (`id_acheteur`),
  KEY `vendeur` (`id_vendeur`),
  KEY `id_categorie` (`id_categorie`)
) ENGINE=InnoDB AUTO_INCREMENT=48 DEFAULT CHARSET=utf8;
```

c - Description des tables principales et de leurs relations avec les autres tables

Les deux tables autour desquelles s'articulent la base de données sont la table "article" et la table "utilisateur".

i - Table article

La table “article” est en relation avec la table “utilisateur” (id du vendeur et éventuellement id de l’acheteur), avec la table "categorie" (id de la catégorie). L’id de la ligne est référencé par la table de liaison “utilisateur_article”.

Chaque ligne correspond à une annonce déposée par un vendeur.

Article	
colonne	**information supplémentaire**
id	clé primaire, id unique de la ligne
titre	nom de l’article
description	texte descriptif
date_ajout	current timestamp (rempli à l’insertion)
date_vente	NULL par défaut (rempli quand marqué comme vendu)
photo	nom de l’image
prix	nombre entier
etat_objet	énum : neuf, très bon état, bon état
signal	NULL par défaut (1 ou 2 si signalé)
id_categorie	clé étrangère, liée à l’id de la table “categorie”
ouvert_negociation	enum : oui, non
status	enum : disponible, vendu
id_vendeur	clé étrangère, liée à l’id de la table “utilisateur”
id_acheteur	NULL par défaut. Lié à l’id de la table “utilisateur”
visible	1 par défaut. Devient 0 si l’annonce est en modération.

categorie_suggeree	NULL par défaut. Rempli par le vendeur si aucune catégorie ne correspond.

ii- Table utilisateur

L'id de l'utilisateur est référencé dans la table "article", la table "note", la table "message" (id de l'expéditeur) et la table "utilisateur_message" (id du destinataire).

Chaque ligne correspond à un utilisateur enregistré.

Utilisateur	
colonne	**information supplémentaire**
id	id unique de la ligne
identifiant	nom unique - devient "utilisateur.ice supprimé.e" à la suppression du profil
mdp	mot de passe crypté
mail	email unique
status	enum : client, vendeur, supprimé
droit	0 par défaut, 1 pour l'admin
zip	nombre entier
date_inscription	current timestamp (rempli à l'insertion)

d - Connexion à la base de données

Toutes les classes modèles héritent d'une classe-mère nommée **Database.php**. Cette classe permet d'établir une connexion avec la base de données.

Sa méthode *__construct()* renseigne les attributs "username", "hostname" et "dbname". Cette méthode appelle aussi la méthode *connexionDb()*, qui fait de l'attribut "pdo" un **objet pdo.**

Les deux autres méthodes *SelectAll($table)* et *findById($table, $id)* permettent de faire des requêtes de sélection de données génériques.

```php
<?php

class Database
{
    public $username;
    public $pass;
    public $hostname;
    public $dbname;
    public $pdo;

    public function __construct()
    {
        $this->username = "root";
        $this->hostname = "localhost";
        $this->dbname = 'lbp';
        $this->connexionDb();
    }

    public function connexionDb()
    {
        try {
            $this->pdo = new pdo("mysql:dbname=lbp;host=localhost;
charset=UTF8", 'root', '');
        } catch (Exception $e) {
            echo $e . "<br>";
        }
    }

    public function closeDb()
    {
        $this->pdo = null;
    }

    public function selectAll($table)
    {
        $query = $this->pdo->prepare("SELECT * from ". $table);
        $query->execute();
        $result = $query->fetchAll(PDO::FETCH_ASSOC);
        return $result;
    }

    public function findById($table, $id){
        $query = $this->pdo->prepare("SELECT * from ". $table ." WHERE
 id = ? ");
        $query->execute([$id]);
        $result = $query->fetchAll(PDO::FETCH_ASSOC);
        return $result;
    }

}
```

2 - Exemples de développement back

La grande majorité des fonctionnalités du site suit le même processus. Ainsi, avant de parler plus en détails de certains exemples que je juge intéressant, je vais tout d'abord présenter :

- le système de routes qui permet l'affichage d'une vue
- l'utilisation d'AJAX dans le projet pour récupérer et traiter des données

Je vais prendre le cas de la connexion d'un utilisateur pour mettre en lumière les fichiers sollicités.

a - Système de routes

i - Routeur et url

fichiers sollicités : index.php, .htaccess, Routeur.php

Nous avons tenté sur ce projet d'utiliser une architecture inspirée de MVC. De ce fait, toutes les pages sont affichées sur **l'index** du site.

- **index.php**

```php
<?php
require_once('autoloader.php');
Autoloader::register();
$routeur = new Routeur();
```

Ce fichier appelle l'autoloader (qui permet d'instancier les classes sans importer manuellement le fichier). Il **crée** aussi un **routeur**.

- **.htaccess**.

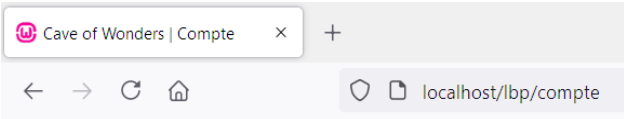
```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.+)$ index.php?r=$1 [QSA,L]
```

Ce fichier se charge de **réécrire les url**. Il fait passer en paramètre $_GET[‘r’] le nom de la page demandée.

- **clic sur le lien**

```
<a href="compte">Compte</a>
```

Ainsi, quand on clique sur le **lien** “compte” dans la barre de navigation, on se rend sur **l’url** suivante :



- **Routeur.php**

```php
<?php


class Routeur
{
//    tableau associatif des pages et de leurs controllers
    private $controllers = [
        "home" => "Home",
        "compte" => "Compte",
        "article" => "Article",
        "resultatArticles" => "ResultatArticles",
        "profilVendeur" => "ProfilVendeur",
        "admin" => "Admin"
    ];

//    controller sélectionné
    private $controller;


    public function __construct()
    {
        if (isset($_GET['r']) && key_exists($_GET['r'], $this->controllers)) {
            $this->controller = new $this->controllers[$_GET['r']]();
        } else {
            header('location: home');
            new Home();
        }
    }
}
```

Le **routeur** récupère le nom de la page demandée ($_GET[‘r’] c’est à dire “compte”) et instancie le **controller associé** (“Compte()”).

Si aucune page ne correspond à la valeur de $_GET[‘r’], on instancie le controller Home().

ii - Controller et vue

fichiers sollicités : Compte.php, connexion.php

- **Controller : Compte()**

```php
class Compte
{
    function __construct()
    {
        $title = "Compte";
        $css = "compte.css";
        $js = ['module.js', 'compte.js', 'vendeur.js'];
        ob_start();
        $this->selectMain();
        $main = ob_get_clean();

        $render = new View($title, $css, $main, $js);
    }
```

Le controller Compte() appelle la méthode *selectMain()* qui **sélectionne la vue** à afficher **selon le statut** de l'utilisateur. Si l'utilisateur n'est pas connecté, il est renvoyé à la vue "connexion.php".

```php
    public function selectMain()
    {
        //Si pas connecté
        if (!isset($_SESSION['user'])) {
            require_once('views/user/connexion.php');
        } else {
            //Si connecté en admin
            if ($_SESSION['user']['droit'] === "1") {
                header( header: 'Location: admin');
                //Si connecté en vendeur
            } else if ($_SESSION['user']['status'] === 'vendeur') {
                require_once('views/user/vendeur.php');
                //Si connecté en acheteur
            } else if ($_SESSION['user']['status'] === 'client') {
                require_once('views/user/client.php');
            }
        }
    }
```

La page est finalement affichée par l'instanciation d'un objet *View()*, une classe qui construit **un gabarit de vue**.

- **Vue : connexion.php**

```php
<main id="mainCompte">
    <article>
        <h2>CONNEXION</h2>
    </article>
    <article class="form">
        <form id="formConnexion">
            <div class="formBloc" id="bloc1">
                <input type="text" id="login" name="login" placeholder="email ou identifiant">
            </div>

            <div class="formBloc" id="bloc2">
                <input type="password" id="password" name="password" placeholder="mot de passe">
                <button type="submit">Se connecter</button>
            </div>
        </form>

        <div class="formInfo">
            <div id="message"></div>
        </div>
    </article>
    <p>Vous n'avez pas encore de compte ? <span class="callForm" id="callFormInscription">Inscrivez-vous.</span></p>
</main>
```

Cette vue est sélectionnée si aucune session n'est ouverte.

b - Traitement en AJAX

i - Soumission du formulaire

fichiers sollicités : module.js, apiModule.php, userModel.php

- **script : module.js**

```javascript
//Submit connexion
$('body').on('submit', '#formConnexion', function (event) {
    $('#message').empty();
    event.preventDefault()
    $.post(
        'API/apiModule.php', {
            form: 'connexion',
            login: $('#login').val(),
            password: $('#password').val(),
        },
```

Quand on soumet le formulaire, le **script** envoie les données à une **page de traitement php**, sous forme de POST grâce à **AJAX**.

- **traitement php : apiModule.php**

```php
/*CONNEXION*/
if (isset($_POST['form']) && $_POST['form'] === 'connexion') {
    if (!empty($_POST['login']) && !empty($_POST['password'])) {
        $login = htmlspecialchars($_POST['login']);
        $password = htmlspecialchars($_POST['password']);
        $userExists = $model->userExists($login, $login);
```

Le fichier php **vérifie** si les valeurs ne sont pas vides et si elles ne contiennent pas de caractères malveillants. Il utilise ensuite le **modèle** pour demander à la base de données si l'utilisateur existe.

- **modèle : userModel.php**

```php
public function userExists($email, $login)
{
    $request = $this->pdo->prepare("SELECT * FROM utilisateur WHERE mail = ? OR identifiant = ?");
    $request->execute([$email, $login]);
    $userExists = $request->fetchAll(PDO::FETCH_ASSOC);
    return $userExists;
}
```

La méthode ***userExists()*** récupère les lignes en **bdd** qui correspondent aux paramètres envoyés.

ii - Résultat de l'appel

fichiers sollicités : apiModule.php, module.js, connexion.php

- **traitement php : apiModule.php**

```php
if (!empty($userExists)) {
    $userExists = $userExists[0];
    if (password_verify($password, $userExists["mdp"]) || $password === $userExists["mdp"]) {
        $_SESSION['user'] = $userExists;
        $result = ['success'];
    } else {
        $result = ['Vérifiez votre mot de passe'];
    }
    echo json_encode($result, flags: JSON_UNESCAPED_UNICODE | JSON_UNESCAPED_SLASHES);
```

Si l'utilisateur existe, le fichier "apiModule.php" vérifie que les **mots de passe** correspondent. Il sauvegarde les informations en **session** et retourne un

message de succès en **format JSON**.

- **script : module.js**

```
function (data) {
    console.log(data);
    let messages = JSON.parse(data);
    for (let message of messages) {
        if (message === "success") {
            $("#message").append("<p>Connexion réussie !</p> <a href='compte'>Voir votre profil</a>");
        } else {
            $('#message').append("<p>" + message + "</p>");
        }
    }
```

Ce message de succès est capturé par la fonction **success** de la requête **AJAX** et ajouté à la div #message.

c - Marquer un article comme vendu

i - récupérer les annonces mises en ligne par le vendeur ainsi que sa liste de contacts

- **script : vendeur.js**

```
        $.post(
            'API/apiVendeur.php', {action: 'articlesSelling'},
            function (data) {
                let articles = JSON.parse(data);
                console.log(articles)
                if (articles == 'none') {
                    $("#articlesSelling").append("<tr><td>Il n'y a rien ici.</td><td class='navUser
navNewArticle'> + Déposer une annonce</td></tr>");
                } else {
                    for (let article of articles) {
                        $('#articlesSelling').append("<tr id ='" + article.id_article + "'><td><a href='
article?id=" + article.id_article + "'>" + article.titre + "</a></td><td>Annonce créee le : " + article.
date_ajout + "</td><td><button class='afficherDetails' >Modifier</button></td><td><select class='
marquerCommeVendu'><option value=''>Vendu à : </option></select></td><td><button class='supprimerArticle' >
Supprimer</button></td></tr>");
                    }
```

La **liste des annonces en ligne** d'un vendeur est générée grâce à un **appel AJAX** qui se déclenche dès que la page est demandée. Si cet appel renvoie un résultat non vide, les articles sont ajoutés à la div #articlesSelling.

- **script : vendeur.js**

```
21                          let select = $('.marquerCommeVendu')
22                          $.post(
23                              'API/apiMessagerie', {action: 'selectContacts'},
24                              function (data) {
25                                  let contacts = JSON.parse(data);
26                                  console.log(data);
27                                  if (contacts == 'none') {
28                                      select.append("<option>Aucun contact</option>");
29                                  } else {
30                                      $.each(contacts, function (key, value) {
31                                          if (value.status != 'supprimé') {
32                                              select.append("<option value='" + value.id + "'>" + value.
   identifiant + "</option>")
33                                          }
34                                      })
35                                  }
36                              },
37                          );
38                      }
39                  })
```

La **liste de contacts** proposée est générée elle aussi par un appel AJAX. Celui-ci récupère les noms et id des personnes avec qui le vendeur a échangé.

- **modèle : userModel.php**

```
95      public function selectContacts($id)
96      {
97          $request = $this->pdo->prepare("SELECT DISTINCT utilisateur.identifiant, utilisateur.id, utilisateur.
   status FROM utilisateur JOIN message INNER JOIN utilisateur_message on utilisateur_message.id_message =
   message.id WHERE (id_expediteur = $id AND id_destinataire = utilisateur.id) OR (id_destinataire = $id AND
   id_expediteur = utilisateur.id) AND droit = 0 AND utilisateur.id != $id");
98          $request->execute();
99          $contacts = $request->fetchAll(PDO::FETCH_ASSOC);
00          return $contacts;
01      }
```

La méthode ***selectContacts()*** prend l'id du vendeur comme paramètre et retourne la liste des contacts de cet utilisateur.

La requête SQL fait une jointure sur les tables "utilisateur", "utilisateur_message" et "message".

Afin de retrouver les acheteurs potentiels, elle pose 3 conditions :

- l'id de l'utilisateur doit être référencé comme id_expediteur et l'id du vendeur comme id_destinataire (ou vice-versa)
- le droit doit être égal à 0 (donc pas à un admin)

- l'id ne doit pas être identique à celui du vendeur

ii - indiquer que l'article a été vendu à tel acheteur

- **script : vendeur.js**

```
//Marquer comme vendu
$('body').on('click', '#confirmerVente', function () {
    let row = $(this).parents('tr')
    let idArticle = row.attr('id')
    if ($('option:selected').val().length > 0) {
        $.post(
            'API/apiVendeur', {
                action: 'marquerCommeVendu',
                idArticle: idArticle,
                idAcheteur: $('option:selected').val()
            },
            function (data) {
                let message = JSON.parse(data);
                row.hide()
                console.log(message)
            }
        );
    }
});
```

Quand le vendeur sélectionne un contact et clique sur le bouton **confirmer la vente**, le script envoie en **AJAX** l'action à effectuer, l'id de l'article et l'id de l'acheteur.

- **modèle : userModel.php**

```
    public function marquerCommeVendu($idAcheteur, $id)
    {
        $request = $this->pdo->prepare("UPDATE article SET status = 'vendu', date_vente ='" . date('Y-m-d H:i:
s') . "', id_acheteur = ? WHERE id = ? ");
        $request->execute([$idAcheteur, $id]);
        return true;
    }
```

Le fichier de traitement php fait usage de la méthode *marquerCommeVendu()* du **modèle** pour **updater la ligne correspondante** en base de données. La méthode change le statut de l'article, ajoute la date de vente et spécifie l'id de la personne qui l'a acheté.

d - Système de messagerie

Le système de messagerie intervient de 2 façons dans l'application :

Quand on crée une nouvelle conversation avec un utilisateur

Quand on répond à un message dans une conversation existante

i - Envoi d'un nouveau message sous forme de pop up

Quand un client ou un administrateur clique sur un bouton *contacter*, une **modale** s'ouvre. Elle comporte le pseudonyme du destinataire, un champ de texte et un bouton *envoyer*.

J'utilise javascript pour récupérer le message et l'id du destinataire. Grâce à AJAX, à mon fichier de traitement "apiMessagerie.php" et à mon modèle "userModel.php", j'insère le message en base de données.

Ci-dessous, la méthode *sendNewMessage()*. Elle prend comme paramètre l'id du destinataire, l'id de l'expéditeur et le contenu du message.

J'insère l'id de l'expéditeur et le contenu du message dans la table "message".

Je récupère l'id de la dernière ligne insérée grâce à la méthode pdo lastInsertId(), puis j'insère l'id du destinataire et l'id du message dans la table utilisateur_message.

Je retourne finalement les données du message.

```php
public function sendNewMessage($idDestinataire, $idUser, $messageContent)
{
    $request = $this->pdo->prepare("INSERT into message (id_expediteur, contenu) VALUES (?, ?)");
    $request->execute([$idUser, $messageContent]);
    $idMessage = $this->pdo->lastInsertId();
    $request2 = $this->pdo->prepare("INSERT into utilisateur_message (id_destinataire, id_message) VALUES (?, ?)");
    $request2->execute([$idDestinataire, $idMessage]);
    $request3 = $this->pdo->prepare("SELECT * FROM message WHERE id = $idMessage");
    $request3->execute();
    $message = $request3->fetch(PDO::FETCH_ASSOC);
    return $message;
}
```

ii - Affichage d'une conversation existante

Toutes les conversations de l'utilisateur connecté apparaissent dans l'onglet *messagerie* de son espace personnel.

Je récupère sa liste de contacts comme expliqué plus tôt. Lorsque l'utilisateur clique sur un contact, il déclenche une requête AJAX qui retourne les messages échangés avec cet utilisateur.

Si l'id de l'expéditeur du message correspond à l'id de l'utilisateur connecté, j'ajoute la classe "messageUtilisateur" à la div, classe qui fait aligner le texte à gauche. Sinon, j'ajoute la classe "messageDestinataire".

iii - Réponse dans une conversation existante

Si le destinataire n'a pas le statut "supprimé", c'est-à-dire qu'il possède toujours un compte sur le site, on peut continuer la conversation. Le processus est le même que pour l'envoi d'un tout nouveau message, à ceci près que la fonction success de ma requête AJAX utilise la fonction *append* pour ajouter visuellement le nouveau message à la conversation.

iv - Droits des utilisateurs

- Anonyme

Il n'a absolument **pas accès** à la messagerie. En effet, le bouton n'est pas cliquable lorsqu'aucune session est ouverte. J'ai réalisé cette sécurisation à l'aide d'une condition PHP qui ajoute une classe CSS ('disabled') au bouton et empêche de cliquer sur le bouton.

Un pop-up informatif s'ouvre pour inviter l'utilisateur à se connecter.

- Client

Il peut **contacter** un vendeur. Je vérifie son statut avec la même condition PHP évoquée ci-dessus. Le bouton est cliquable.

Comme il est connecté, il a accès à la messagerie dans son espace utilisateur.

- Client

Un vendeur n'a pas la possibilité de contacter lui-même un autre utilisateur. Il peut uniquement **répondre** aux messages qui lui sont envoyés. Comme il est connecté, il a accès à la messagerie dans son espace utilisateur.

- Administrateur

Il peut contacter tous les utilisateurs du site. Il a accès à l'adresse email des utilisateurs.

e - Articles envoyés en modération

L'administrateur peut gérer - modifier ou supprimer - les utilisateurs et les annonces en ligne. Un onglet de son espace lui permet de voir les annonces dont la colonne "visible" en base de données est égale à 0.

Il peut soit les valider, soit les supprimer.

i - Articles signalés

Les clients ont la possibilité de signaler une annonce grâce à un bouton "signaler" sur la fiche produit. Ce bouton modifie en base de données la colonne "signal" - initialement nulle - de la ligne correspondant à l'annonce dans la table "article". Dès que la valeur de la colonne est égale à 2, la valeur de la colonne "visible" passe à 0.

Quand l'admin appuie sur "accepter", je récupère l'id de l'article et, grâce à apiAdmin.php et adminModel.php, je remet la valeur de "signal" à null en base de données et je met la colonne "visible" à 1.

ii - Articles avec catégorie suggérée

Par défaut, les annonces créées par les vendeurs sont immédiatement publiées sur le site. Cependant, si le vendeur ne trouve pas de catégorie d'article lui convenant, il a le choix de suggérer une nouvelle catégorie.

Quand il remplit le champ "catSuggeree" du formulaire de création d'une nouvelle annonce, l'annonce s'insère en base de données avec la valeur 0 pour "visible" et le nom de la catégorie pour la colonne "categorie_suggeree", originellement nulle.

Si l'admin accepte la suggestion de catégorie, il se passe deux choses : la catégorie est ajoutée en base de données et l'annonce est modifiée. La colonne "categorie_suggeree" est mise à nulle, on ajoute l'id de la catégorie nouvellement créée et on passe la colonne "visible" à 1.

6 / SÉCURITÉ ET RECHERCHE

1 - Présentation d'un jeu d'essai

Au cours du développement, plusieurs fonctionnalités ont nécessité d'être testées et découpées étape par étape. La plus représentative est l'inscription.

a - Récupération des données du formulaire (module.js)

Quand l'utilisateur envoie le formulaire d'inscription, la div de message est vidée grâce à la fonction *empty()*.

Les données du formulaire sont récupérées et envoyées en POST au fichier de traitement PHP grâce à AJAX.

b - Traitement PHP (apiModule.php)

Vérification formulaire rempli

Je vérifie que tous les champs sont renseignés grâce à la méthode *!empty()*.

Si le formulaire n'est pas entièrement rempli, *j'echo* le message "Veuillez remplir tous les champs SVP" en json.

Sinon, je crée des variables PHP en utilisant *htmlspecialchars()* ainsi qu'un tableau *$errors.*

Vérification utilisateur déjà existant

J'utilise la méthode *$userModel->userExists($login, $email)* qui retourne true ou false si l'identifiant ou l'adresse email figurent déjà en base de données.

Si l'utilisateur existe déjà, j'echo le message "Cet email ou ce login est/sont liés à un compte." en json.

Vérification email et mot de passe

- J'utilise la méthode *filter_var()* pour valider le format d'adresse email.
- J'utilise la méthode preg-match() avec le pattern suivant : '/^(?=.*\d)(?=.*[A-Za-z])[0-9A-Za-z!@#$%]{6,15}$/'

pour vérifier si le mot de passe est suffisamment fort. Je teste la présence de :

- Un chiffre
- Un caractère spécial
- 6 à 15 caractères

- Je vérifie l'égalité des deux chaînes de caractères *$password* et *$password2.*

Si ces trois conditions ne sont pas remplies, j'ajoute un message d'avertissement au tableau *$errors*.

Si le tableau *$errors* n'est pas vide, *j'echo* le tableau en json.

Sinon, je passe à l'insertion.

Insertion en base de données

Je hache le mot de passe avec la méthode *password_hash()* et l'algorithme *password_bcrypt*.

J'utilise la méthode *$userModel->insertUser()* qui insère la ligne et les valeurs en bdd.

Si l'insertion ne s'est pas faite, *j'echo* le message "Un problème est survenu".

Sinon, *j'echo* le message "success".

c - Fonction success (module.js)

Je récupère et parse les données renvoyées par l'appel AJAX. Je fais une boucle pour afficher les erreurs. Si le message est égal à "success", j'ajoute le message "Inscription réussie" à la div *#message*.

L'utilisateur est désormais inscrit en base de données et il peut se connecter.

2 - Veille informatique sur les vulnérabilités de sécurité

Tout au long de l'année, nous avons été sensibilisés à la sécurisation maximum de nos réalisations. Nous avons appris à ne jamais faire confiance aux données transmises par un utilisateur, car ce dernier peut avoir des desseins malveillants.

a - Documentation générale

J'ai parcouru les sites web suivants pour m'assurer de connaître les principales attaques et menaces, ainsi que les bonnes pratiques :

- L'Agence nationale de la Sécurité des systèmes d'information (https://www.ssi.gouv.fr/) notamment les guides « Recommandations pour la mise en œuvre d'un site web : maîtriser les standards de sécurité côté navigateur » et « Recommandations pour la sécurisation des sites web ». Ces guides sont très détaillés et plutôt techniques, à destination des professionnels.
- Le Mozilla Developer Network (https://developer.mozilla.org/fr/), notamment l'article « La sécurité d'un site web » qui expose les vulnérabilités les plus courantes, comme le XSS, les injections SQL ou les attaques CSRF.
- Openclassrooms (https://openclassrooms.com/), notamment les cours « Effectuez votre veille en cybersécurité » et « Sécurisez vos applications web avec l'OWASP ». Ils traitent des réglementations, des attaques courantes, et de la façon de sécuriser une application.

- Zeste de Savoir (https://zestedesavoir.com/), notamment le tutoriel sur les injections SQL, qui donne des exemples de code ayant des faiblesses et explique comment y remédier.

b - Bulletins et actualités

J'ai pris connaissance de l'existence de ces bulletins qui référencent les derniers dangers et vulnérabilités.

- Le CERT-FR, Centre gouvernemental de veille, d'alerte et de réponse aux attaques informatiques (https://www.cert.ssi.gouv.fr/).
- Vigil@nce d'Orange (https://vigilance.fr/).
- Le CVE, Common Vulnerabilities and Exposures (https://cve.mitre.org/).

3 - Sécurité

a - Envoi de données

Le site comportant différents formulaires qui envoient des données en bdd, il était primordial de vérifier les valeurs envoyées avant de les insérer. J'utilise ;

- la fonction php *!empty()* sur tous les champs. Si la condition n'est pas remplie, le formulaire n'est pas traité. Je vérifie aussi la longueur de la chaîne de caractères avec *strlen()*.
- *htmlspecialchars()* pour convertir les caractères spéciaux et balises HTML (comme les chevrons) en chaîne de caractères ordinaires. Cela permet de lutter contre les failles XSS, qui sont l'insertion de script malveillants du côté client.

- les requêtes préparées PDO pour déjouer les injections SQL. Par défaut, PDO traite la donnée comme une chaîne de caractères (il l'échappe). Cela permet de séparer les données et la syntaxe.
- l'instruction *SameSite* placée sur les cookies de sessions et les token anti-CSRF pour lutter contre l'exécution d'actions non désirées par l'utilisateur sur une application dans laquelle il est authentifié.
- la fonction *preg_match()* pour exiger un mot de passe fort, avec 6 caractères, une majuscule, un chiffre et un caractère spécial pour prévenir l'application des authentifications non autorisées. Le mot de passe est aussi haché grâce à la fonction *password_hash()*.

b - Contrôle d'accès

Les quatre groupes, niveaux d'accès ou rôles, peuvent accéder à des pages et des fonctionnalités différentes.

Rôle	Fonctionnalités	Droit en base de donnée
anonyme	Boutique uniquement.	Aucun
vendeur	Espace personnel.	0
acheteur	Espace personnel. Peut contacter un vendeur.	0
admin	Espace d'administration. Peut contacter tout utilisateur.	1

- Chaque page possède un contrôle d'authentification. L'utilisateur est renvoyé à la page *home* ou *compte* correspondant à son statut s'il tente d'accéder à une page qui ne lui est pas adressée.

- L'url est simplifiée et ne fait pas référence directement à la base de données pour empêcher l'utilisateur malveillant de tenter d'accéder à une autre page.
- J'ai utilisé la commande *Options -Indexes* dans le fichier .htaccess pour bloquer l'accès aux répertoires de l'application.
- Toute url de page ne figurant pas dans le tableau associatif des Controllers (défini dans le routeur) renvoie sur l'accueil.
- Le site a été scanné par Sucuri (https://sucuri.net/) pour détecter d'éventuelles failles de sécurité.

4 - Situation de travail nécessitant une recherche en anglais

a - Présentation du problème

Avec ma partenaire de travail, nous avions déjà réalisé une boutique en ligne en php où l'administrateur soumettait des articles avec photos. Je me souvenais ne pas avoir été satisfaite par la façon dont nous avions codé l'envoi de fichier image.
Cette fois-ci, j'ai donc pris le temps de faire ça le plus proprement et simplement possible.

J'ai préféré conduire ma recherche en anglais pour maximiser la chance de trouver une solution complète et simple.

b - Recherche

- Les mots clés recherchés :

send picture file in form php js

tutorial upload file picture ajax jquery

- Liste des sites retournés :

https://makitweb.com/how-to-upload-image-file-using-ajax-and-jquery/

https://stackoverflow.com/questions/40500535/how-to-upload-image-using-javascript-ajax-and-php

https://openwebsolutions.in/blog/upload-display-image-javascript-php/

https://www.w3schools.com/php/php_file_upload.asp

https://www.codexworld.com/upload-file-using-javascript-php/

- Critère de sélection du site :

J'ai choisi un article qui semblait simple et clair, avec des extraits de code et un processus bien expliqué. J'ai fait attention à la date de publication.

- Solution :

Le tutoriel a été une bonne base de travail.

c - Extrait du site anglophone

Last updated on May 16th, 2021 by Yogesh Singh

How to upload Image file using AJAX and jQuery

1. HTML

Create a `<form >` element where added `<img >`, file element, and a button.

Image preview display in `<img>` after successfully upload using jQuery.

2. CSS

Hide the `img` element.

3. PHP

Create an `upload.php` file and `upload` folder to store image files.

Read file extension. Initialized `$valid_extensions` Array with image extensions.

Check if file extension exists in `$valid_extensions` Array or not. If exists then assign file location to `$response` variable.

Return `$response` variable.

4. jQuery

On the upload button click get the selected file and create a `FormData` object.

Check if a file is selected or not. If not selected then `alert("Please select a file.")` otherwise, append `files[0]` to `'file'` key in `fd`.

Send an AJAX request where pass the `fd` object as data and on successful callback check the response is `0` or not.

If it is not `0` then update the `<img >` source otherwise `alert('file not uploaded')` message.

On page load `img` element is set `display: none;`.

This element is getting displayed when a file is successfully uploaded with jQuery – `$(".preview img").show();`.

d - Ma traduction

Comment uploader un fichier de type image en utilisant AJAX et jQuery

1. **HTML**

Créez un élément <form> où sont ajoutés un élément <img>, un élément fichier et un bouton.

La prévisualisation de l'image apparaîtra dans la balise <img> après la réussite du téléversement en utilisant jQuery.

2. **CSS**

Cachez l'élément <img>.

3. PHP

Créez un fichier upload.php et un dossier upload pour stocker les images. Lisez l'extension du fichier. Initialisez un tableau $extensions_valides avec les extensions d'image.

Vérifiez si l'extension du fichier existe dans $extensions_valides ou non. Si c'est le cas, assignez la localisation du fichier à la variable $reponse. Retournez la variable $reponse.

4. jQuery

Au clic sur le bouton upload, récupérez le fichier sélectionné et créez un objet FormData appelé fd. Vérifiez si un fichier est sélectionné ou non. S'il n'est pas sélectionné, faites l'alerte ("S'il vous plaît, sélectionnez un fichier"). Sinon, ajoutez files[0] à la clef 'file' dans fd.

Envoyez une requête AJAX qui passe l'objet fd en donnée et qui vérifie avec sa fonction de callback succès si la réponse est égale à 0 ou non.

Si elle n'est pas égale à 0, mettez à jour la source <img>. Sinon, faites l'alerte du message ("Fichier non envoyé").

Au chargement de la page, l'élément <img> a la propriété display: none; . Cet élément est affiché quand un fichier est bien envoyé avec jQuery - $(".preview img").show(); .