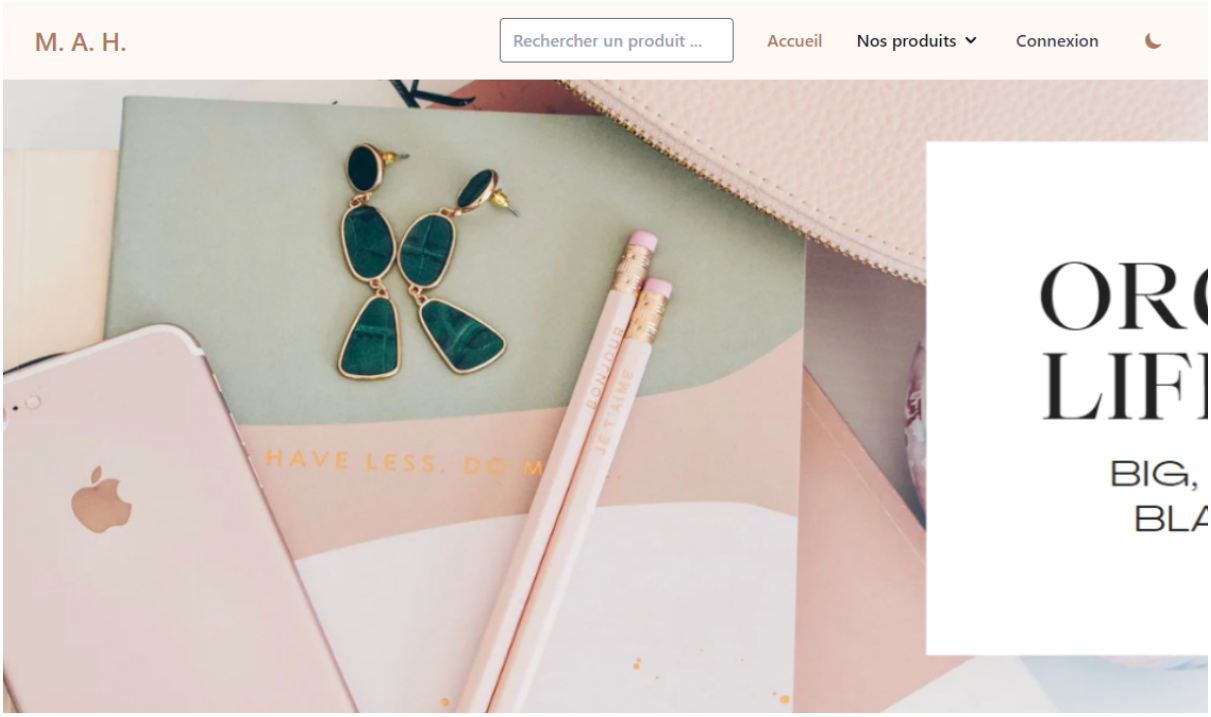
DOSSIER DE PROJET PROFESSIONNEL

Titre : Développeur Web et Web Mobile



Aurélien GALLEA

Table des matières

Compétences du référentiel couvertes par le projet

Le projet couvre les compétences énoncées ci-dessous.

Pour l'activité 1, **"Développer la partie front-end d'une application web et web mobile en intégrant les recommandations de sécurité"**:

- ✔️ Maquetter une application
- ✔️ Réaliser une interface utilisateur web ou mobile statique et adaptable
- ✔️ Développer une interface utilisateur web dynamique
- ✔️ Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce

Pour l'activité 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
- ✔️ Élaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce.

Résumé

M.A.H est une boutique en ligne innovante et passionnante créée par trois développeurs.Notre objectif principal est de proposer des vêtements de haute qualité, tendance et abordables à nos clients.

Ce projet à été réalisé sur 8 semaines. Afin de mettre en place celui-ci, nous avons dû établir les limites de réalisation tout en répondant à un cahier des charges et plusieurs problématiques. Pour cela, nous avons effectué un brainstorming avec l' équipe pour vérifier notre vision du projet. Il en est ressorti plusieurs axes de développement pour le projet de boutique en ligne. Pour répondre à la charte graphique, nous avons réalisé une maquette du site internet. Nous avons également réfléchi ensemble sur la conception de la base de données. A la suite de cela, nous avons défini les technologies front-end et back-end à utiliser, en s'appuyant sur celles apprises tout au long de la formation et qui correspondent à la mise en œuvre du projet. L'idée était de proposer un site avec un design actuel, responsive et couvrir tout le processus d'achat (avec/sans API de paiement).

Spécifications fonctionnelles

I. Expression des besoins

M.A.H cherche à innover sur le secteur de la mode, ce site répond à la problématique de pouvoir toucher un public plus large sur une zone de chalandise plus étendue. Le site est disponible pour format mobiles, tablettes et ordinateurs.

II. Arborescence du site

Voici l'arborescence du site pour un utilisateur lambda :

- Accueil
- Connexion
- Inscription
- Déconnexion
- Page profil utilisateur
- Page tous les produits
- Page produits triés par catégorie
- Page produit
- Page panier
- Page choix/ajout de la livraison avec choix du livreur
- Page récapitulative de commande
- Page paiement, confirmation de commande
- Page mes commandes

Une partie back-office est également présente afin de permettre la gestion du site (disponible uniquement pour les administrateurs).

III. Description des fonctionnalités

1. Authentification

L'utilisateur doit s'inscrire avec :

- son adresse email
- son mot de passe,
- son prénom et nom.

Ces derniers pourront servir pour la livraison.
Pour la connexion seul l'email et le mot de passe sont nécessaires.

2. Profil utilisateur

L'utilisateur connecté peut s'il le souhaite, mettre à jour ses informations personnelles :

- Prénom
- Nom
- Mot de passe

3. Catalogue des produits, filtres et recherches

Une page permet d'afficher tous les produits disponibles, il est également possible de filtrer les résultats par catégories via la barre de navigation. Une autocomplétion est également disponible pour rechercher directement un produit par son nom.

4. Fiche produit

L'utilisateur peut choisir la couleur, la taille ainsi que la quantité de produit. Les tailles disponibles peuvent varier selon la couleur choisie. Dans la version actuelle de l'application seul un utilisateur connecté peut ajouter des produits à son panier.

5. Panier client

Le panier contient tous les produits ajoutés par l'utilisateur connecté. L'utilisateur peut s'il le souhaite changer la taille du/des produits ajoutés ainsi que la quantité. L'utilisateur peut également choisir de supprimer un/des produits du panier.

6. livraison

Pour continuer le processus d'achat l'utilisateur est invité à fournir une adresse de livraison, il peut pour cela :

- Utiliser une adresse existante (si ajouté au préalable)
- Ajouter une nouvelle adresse

Il sera ensuite invité à choisir son prestataire pour la livraison.

7. récapitulatif de commande et paiement

Le récapitulatif présente la liste des produits commandés ainsi que le prix total de la commande avec la livraison choisie.
Une fois validé, une simulation de paiement est mise en place pour apporter une interaction.

8. Commandes utilisateur

Cette page permet à l'utilisateur de visualiser toutes les commandes faites précédemment sur le site.

9. Back office

9.1 Gestion des catégories produit

L'administrateur peut gérer les catégories des produits :

- Ajouter
- Renommer
- Supprimer

9.2 Gestion des produits

L’administrateur a également la capacité de créer un produit, mettre à jour les informations, le supprimer.

Lors du processus de création :

- Nom du produit
- Description
- Image (une ou plusieurs)
- Prix
- Couleur
- Taille
- quantité
- Choisir la collection (catégorie) à laquelle l’ajouter.

9.3 Gestion des transporteurs

Ajout /mise à jour / suppression également.

9.4 Gestion des utilisateurs

L’administrateur peut voir/supprimer un utilisateur enregistré.

Specifications techniques

Choix techniques et environnements de travail

I. Front-end

1. Technologies Front-end

Afin de réaliser la partie front-end de l’application, nous avons utilisé les langages HTML,CSS et Javascript. Nous avons également utilisé Tailwind CSS avec la librairie flowbite pour nous aider dans le design et le responsive.

II. Back-end

1. Technologies Back-end

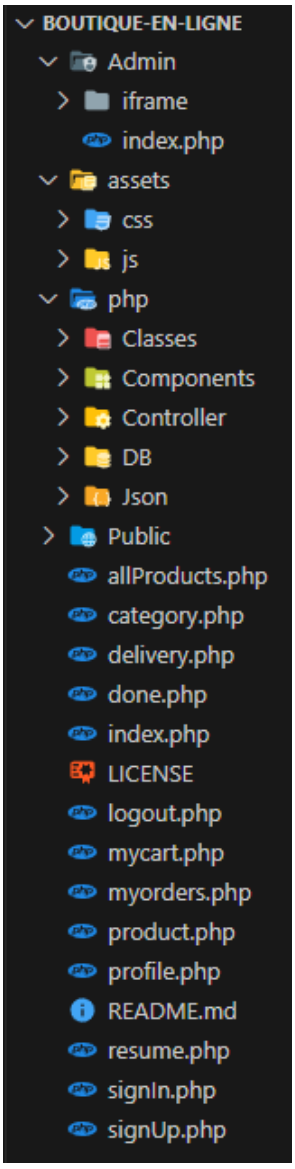
En ce qui concerne la partie du langage back-end nous avons fait le choix de PHP.

2. Base de données

La base de données choisie est en langage SQL, et le Système de Base de Données (SGBD) est MySQL. Nous avons également utilisé phpMyAdmin, application Web de gestion pour les systèmes de gestion de base de données MySQL et MariaDB.
Nous avons également créé un fichier permettant de rapidement mettre en place la base de données, en cas de déploiement sur un nouvel hébergeur, par la création automatique des tables (voir en annexe).

3. Arborescence des fichiers

Ayant pour objectif d'héberger l'application sur notre plesk étudiant, n'ayant pas la possibilité d'utiliser l'url rewriting et voulant avoir absolument un url propre. Le modèle **MVC**[1], bien que largement utilisé, n'a pas été utilisé ici.



[1] Model View Controller : est un motif d'architecture logicielle destiné aux interfaces graphique, il est composé de 3 modules :

- Un modèle (Model) contient les données à afficher (notre classe PHP, qui fait des requêtes SQL) ;
- Une vue (View) contient la présentation de l'interface graphique ;
- Un contrôleur (Controller) contient la logique concernant les actions effectuées par l'utilisateur.

III. Autres Outils

1. Git et GitHub

Durant le processus d'évolution du projet, nous nous sommes servi de Git, outil de gestion des versions afin d'enregistrer et partager notre code sur GitHub.

2. IDE

Pour élaborer ce projet, j'ai utilisé l'éditeur de code : Visual Studio Code.

3. Communication

Afin de communiquer rapidement une information, nous avons fait le choix de Discord. Son système de chat / forum moderne était parfait pour ce projet et se tenir au courant des features en cours de production. Nous aurions pu utiliser Trello avec son système kanban également, mais nous n'en avons pas ressenti le besoin.

Réalisation

I. Charte graphique

1. Logo

thème clair / thème sombre

M. A. H. M. A. H.

2. Palette de couleurs

#AD785D #FFF9F5 #1F2937

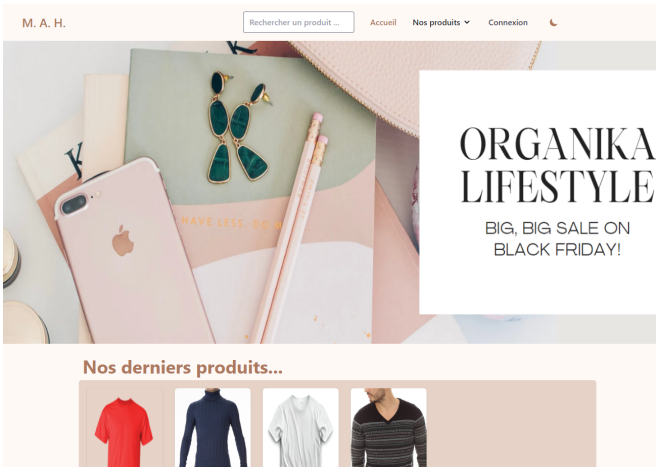
3. Police d'écriture

```
font-family : ui-sans-serif, system-ui, -apple-system;
```

II. Maquette

Pour le maquettage de l'application nous avons utilisé Figma qui est un outil de prototypage. Figma nous a permis de concevoir toute l'architecture visuelle du site.(Voir annexe Maquette p.33).

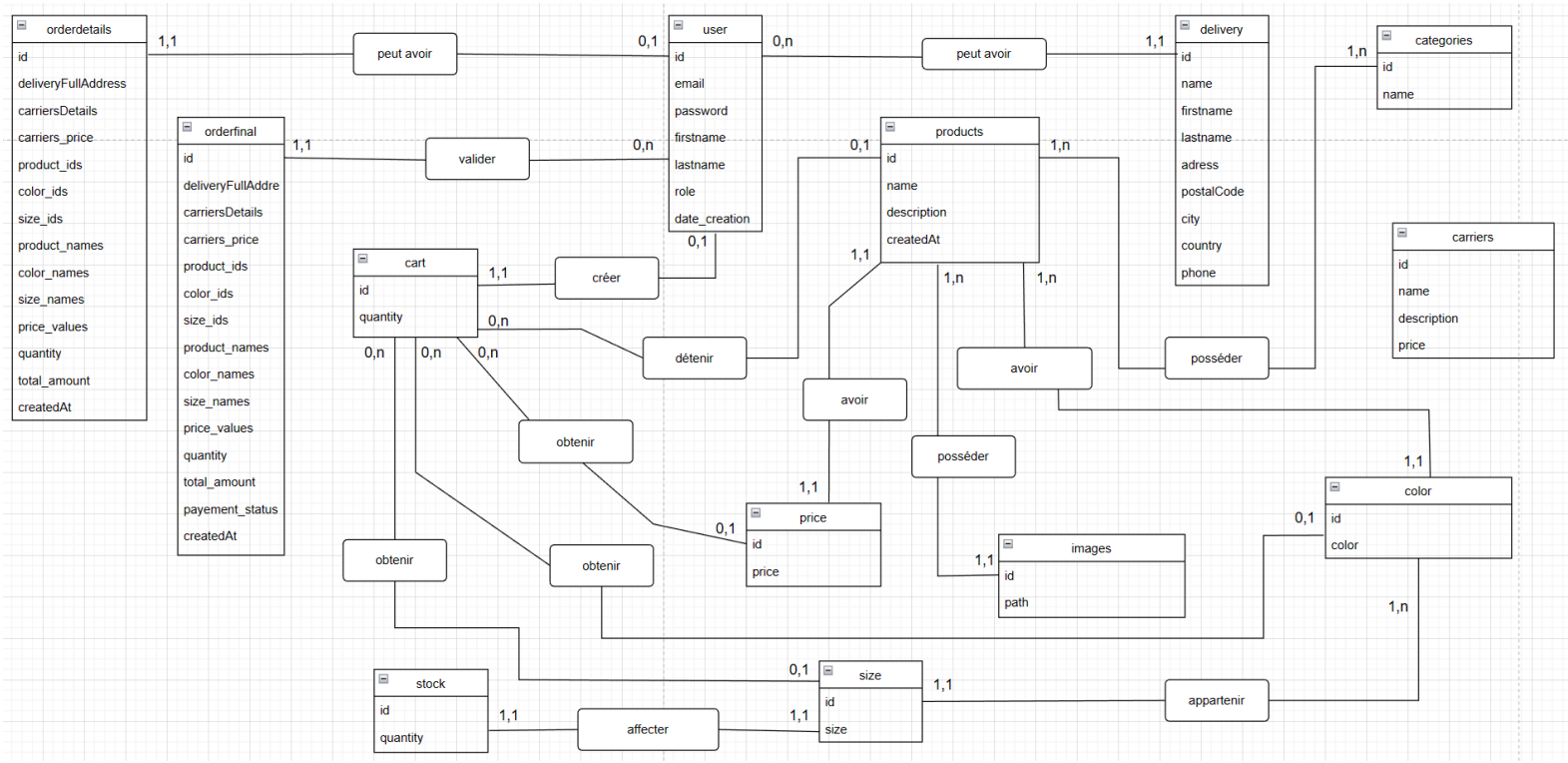
Figma



Ici nous avons conçu la maquette en partant du mobile (320px de largeur) vers la tablette (768px de largeur) pour aller vers le format desktop (1280px de largeur).Le responsive est directement géré via les classes CSS de tailwindcss, les breakpoints utilisant des medias queries(voir annexes p34).

III. Conception de la base de donnée

1. Modèle Conceptuel de Données (MCD)



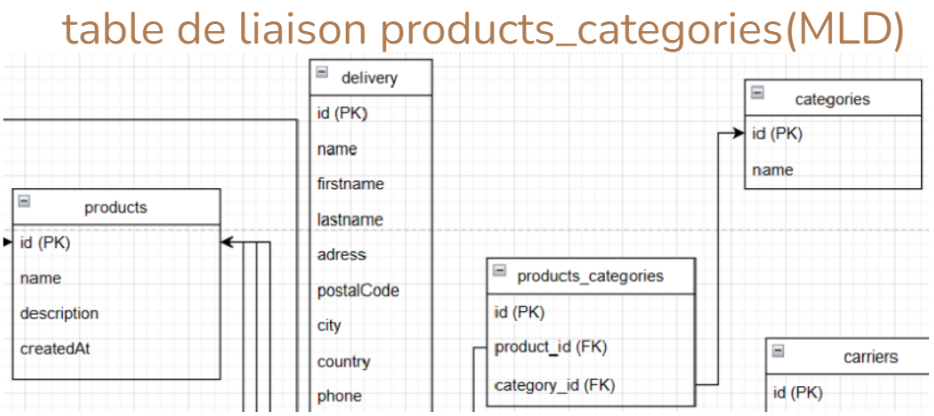
Vous trouverez également dans les annexes (p.34) le modèle logique de données ainsi que le modèle physique de données (p 35).

Comme illustré ci dessus, on peut voir que la base de données s'articule autour de 3 tables principales :

- user(les clients) est lié aux tables :
  - ➔ delivery (la livraison)
  - ➔ cart (le panier client)
  - ➔ orderdetails (la commande en cours non payé)
  - ➔ orderfinal (la commande passé)

```
'user' => 'CREATE TABLE `user` (
                `id` INT AUTO_INCREMENT PRIMARY KEY,
                `email` varchar(255) NOT NULL UNIQUE,
                `password` varchar(255) NOT NULL,
                `firstname` varchar(255) NOT NULL,
                `lastname` varchar(255) NOT NULL,
                `role` varchar(80) NOT NULL,
                `createdAt` timestamp NOT NULL DEFAULT current_timestamp()
                ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',
```

- products (le produit) est lié aux tables :
  - ➔ price
  - ➔ images
  - ➔ color
  - ➔ categories (par l'intermédiaire d'une table de liaison product_categories, due aux **liaisons many to many** du **MCD** ci-dessus)



  - ➔ cart

```
'products' => 'CREATE TABLE `products` (
                `id` INT AUTO_INCREMENT PRIMARY KEY,
                `name` varchar(255) NOT NULL,
                `description` varchar(255) DEFAULT NULL,
                `createdAt` timestamp NOT NULL DEFAULT current_timestamp()
                ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',
```

La table products est liée à la table color et color elle-même liée à la table size. De cette façon les produits ont une option parente 'color' dont celle-ci a une option enfant 'size'. Cela permet d'avoir plusieurs tailles rattachées à une seule couleur, couleur rattachée à un seul produit. **products->color->size.**

- cart (le panier) est lié aux tables :
  - ➔ product
  - ➔ color
  - ➔ size
  - ➔ price
  - ➔ user

```
'cart' => 'CREATE TABLE `cart` (
            `id` INT AUTO_INCREMENT PRIMARY KEY,
            `quantity` INT NOT NULL,
            `product_id` INT NOT NULL,
            `color_id` INT NOT NULL,
            `size_id` INT NOT NULL,
            `price_id` INT NOT NULL,
            `user_id` INT NOT NULL,
            FOREIGN KEY (`user_id`) REFERENCES `user`(`id`),
            FOREIGN KEY (`price_id`) REFERENCES `price`(`id`),
            FOREIGN KEY (`size_id`) REFERENCES `size`(`id`),
            FOREIGN KEY (`product_id`) REFERENCES `products`(`id`),
            FOREIGN KEY (`color_id`) REFERENCES `color`(`id`),
            `createdAt` timestamp NOT NULL DEFAULT current_timestamp()
            ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;'
```

En ce qui concerne les tables orderdetails et orderfinal, les données y sont entrées en brut, au cas où un utilisateur supprime son compte après avoir passé commande, on pourra alors le livrer quoi qu'il arrive. Seule différence entre ces 2 tables la colonne payement_status présente sur orderfinal.

- orderdetails est la commande en cours
- ordefinal est la commande payé

```
'orderFinal' => 'CREATE TABLE `orderfinal` (
                `id` INT AUTO_INCREMENT PRIMARY KEY,
                `deliveryFullAddress` TEXT NOT NULL,
                `carriersDetails` TEXT NOT NULL,
                `carriers_price` FLOAT NOT NULL,
                `product_ids` TEXT NOT NULL,
                `color_ids` TEXT NOT NULL,
                `size_ids` TEXT NOT NULL,
                `product_names` TEXT NOT NULL,
                `color_names` TEXT NOT NULL,
                `size_names` TEXT NOT NULL,
                `price_values` TEXT NOT NULL,
                `quantity` INT NOT NULL,
                `total_amount` FLOAT NOT NULL,
                `payement_status` BOOLEAN NOT NULL,
                `user_id` INT NOT NULL,
                FOREIGN KEY (`user_id`) REFERENCES `user`(`id`),
                `createdAt` timestamp NOT NULL DEFAULT current_timestamp()
                ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',
```

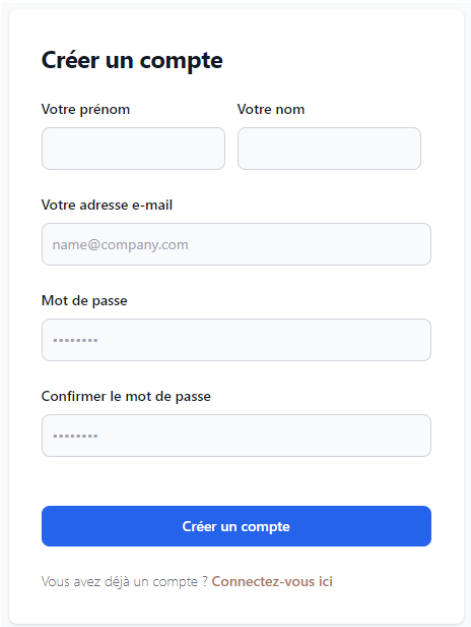
À ce propos, la table orderfinal n'est créée qu'une fois que la commande est validée (la table cart et orderdetails, seront alors vidés pour le client ayant passé commande car l'utilisateur ne peut avoir 0 ou 1 commande en cours/panier en cours), nous y viendrons plus tard via les extraits de code significatifs.

NB: retrouvez en annexe (p . 36) le fichier de création de la base de données.

IV. Extraits de code significatifs

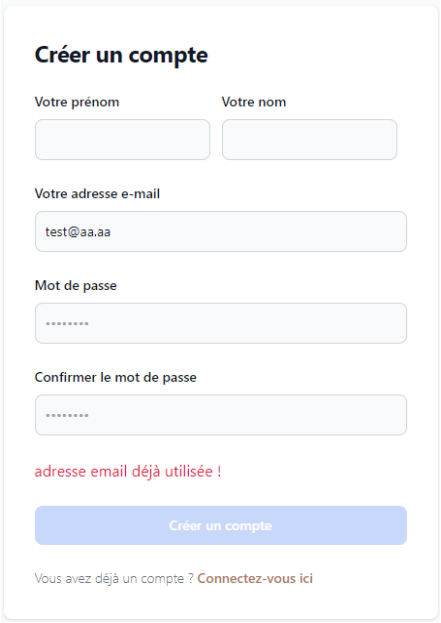
1. Authentification

1.1 Inscription



- Méthode Javascript + PHP

Pour s'inscrire l'utilisateur doit entrer son email, celui-ci doit être unique, nous avons mis plusieurs sécurités en place pour cela.



notions importantes:

- **DOM (Document Object Model) :** C'est le document chargé par le navigateur, le terrain de jeu du front-end, via sa manipulation, on peut Créer, personnaliser, faire plein de choses de façons visuel, pour faire comprendre une information à notre utilisateur.
- **FETCH** : API (Application Programming Interface) permettant d'effectuer des requêtes HTTP, pour récupérer/passer des données de façons asynchrones à l'aide des promesses en javascript.

Ici on se sert de l'événement "keyup" et de email.value (la valeur de l'input email) pour envoyer un fetch : POST, qui permettra de récupérer une réponse du serveur, en fonction de cette réponse, on désactive/active le bouton du formulaire et on met un message à l'écran pour prévenir l'utilisateur. On utilise le **DOM** pour modifier dynamiquement le visuel de la page.

```
email.addEventListener('keyup', () => {

    fetch(`${keyPath}checkAvailable.php`,{
    method: "POST",

    body: 'email=' + email.value,
    headers: {
      "Content-Type": "application/x-www-form-urlencoded",
    },
  })
    .then((response) => {
      if (response.ok) {
        response.text().then(status => {

            // on utilise trim() pour éviter les caractères d'échappements
            if (status.trim() === "used") {

                errorMsg.textContent = "adresse email déjà utilisée !"
                btnOff();
            }else {
                errorMsg.textContent = "";
                btnOn();
            }
        });
      } else {
        alert("problème detecté merci de contacter un administrateur.");
      }
    })
    .catch(function (error) {
      console.log(error);
    });
```

Pour vérifier si l'email est déjà utilisé, on attend une réponse du serveur via la fonction "availableEmail". Si la réponse est supérieure à 0, l'email est déjà utilisé, j'affiche un message d'erreur à mon nouvel utilisateur. Sinon, je permet à mon utilisateur d'utiliser cet email et active le bouton de soumission du formulaire.

```
$email = $_POST['email'];
$myUser = new User();
$usedEmail = $myUser->availableEmail($email);

if ($usedEmail > 0) {
    echo("used");
} else {
    echo("available");
}
```

La méthode de la classe User qui vérifie la présence de l'email dans la table "user".

```
public function availableEmail($email) {
  file_exists('../DB/DBManager.php')? require('../DB/DBManager.php'):require('./php/DB/DBManager.php');
  $request = $bdd->prepare("SELECT * FROM ".$this::TABLE_NAME." WHERE email = ? ");
  $request->execute([$email]);
  return $request->rowCount();
}
```

Ici on crée de l'interaction grâce à la manipulation du DOM, à l'aide de la fonction "passChecker" on affiche/cache un message d'erreur en fonction de la différence de valeur entre le "mot de passe" et la "confirmation de mot de passe". On bloque le bouton du formulaire grâce à l'événement "keyup" si les mdp ne correspondent pas. Afin de **DRY** (Don't Repeat Yourself) le code, cette fonction a été créé.

```
function passChecker(inputPass) {
    inputPass.addEventListener('keyup', () => {
        if (password.value !== confirmPassword.value)  {
            errorMsg.textContent = "Les mots de passes ne correspondent pas !";
            btnOff();
        } else {
            errorMsg.textContent = "";
            btnOn();
        }
    });
}
passChecker(password);
passChecker(confirmPassword);
```

- Méthode PHP

Ici on empêche l'injection de scripts dans les inputs via la méthode htmlspecialchars() en **PHP**.

```
$firstname = htmlspecialchars($_POST['firstname']);
$lastname = htmlspecialchars($_POST['lastname']);
$email = htmlspecialchars($_POST['email']);
$pass1 = htmlspecialchars($_POST['password']);
$pass2 = htmlspecialchars($_POST['confirm-password']);
```

On utilise également la classe Verify(), une classe avec une méthode statique, qui permet de vérifier la syntaxe du mail, au cas où un utilisateur malveillant voudrait changer le type de l'input dans le **HTML**. On utilise pour cela la fonction **PHP** filter_var($email, FILTER_VALIDATE_EMAIL).

```
class Verify {

        public static function verifySyntax($email) {
            if(!filter_var($email, FILTER_VALIDATE_EMAIL)) {
                return false;
            } else {
                return true;
            }
        }
    }
```

Pour le mot de passe on utilise un algorithme de hachage BCRYPT avec la fonction password_hash(), afin de sécuriser les mots de passe et qu'ils ne soient pas enregistrés en

clair dans la base de données. BCRYPT Permet de se protéger contre les attaques par dictionnaire et par force brute en les ralentissant, grâce à son changement de clé lent..

```php
$password = password_hash($pass1, PASSWORD_BCRYPT);
```

Ci-dessous le déroulé global de la gestion du formulaire :

- Encapsulation par condition : chaque input est rempli ? (Ils ont tous l'attribut “required” en HTML)

```php
<form id="form" class="space-y-4 md:space-y-6" method="post" action="">
    <div class="flex gap-3">

        <div>
            <label for="firstname" class=" block mb-2 text-sm font-medium text-gray-900 dark:text-white">Votre prénom</label>
            <input type="text" name="firstname" id="firstname" required class="bg-gray-50 border border-gray-300 text-gray-900 sm:t
        </div>
        <div>
            <label for="lastname" class="block mb-2 text-sm font-medium text-gray-900 dark:text-white">Votre nom</label>
            <input type="text" name="lastname" id="lastname" required class="bg-gray-50 border border-gray-300 text-gray-900 sm:tex
        </div>
    </div>
    <div>
        <label for="email" class="block mb-2 text-sm font-medium text-gray-900 dark:text-white">Votre adresse e-mail</label>
        <input type="email" name="email" id="email" required class="bg-gray-50 border border-gray-300 text-gray-900 sm:text-sm roun
    </div>
    <div>
        <label for="password" class="block mb-2 text-sm font-medium text-gray-900 dark:text-white">Mot de passe</label>
        <input type="password" name="password" id="password" required placeholder="••••••••" class="bg-gray-50 border border-gray-3
    </div>
    <div>
        <label for="confirm-password" class="block mb-2 text-sm font-medium text-gray-900 dark:text-white">Confirmer le mot de pass
        <input type="password" name="confirm-password" id="confirm-password" required placeholder="••••••••" class="bg-gray-50 bord
    </div>

    <div><span id="errorMsg" class="text-rose-600">
    <?php
    if (isset($_GET['error']) && !empty($_GET['message'])) {
         echo htmlspecialchars($_GET['message']);
        }
        ?>
        </span></div>
    <button type="submit" id="signUp" name="signUp" class="disabled:opacity-25 w-full text-white bg-blue-600 hover:bg-blue-700 focu
    <p class="text-sm font-light text-gray-500 dark:text-gray-400">
        Vous avez déjà un compte ? <a href="./signIn.php" class="font-medium text-primary-600 hover:underline dark:text-primary-500
    </p>
</form>
```

```php
if (isset($_POST['signUp'])) {
    if (!empty($_POST['firstname']) && !empty($_POST['lastname']) && !empty($_POST['email']) &&
!empty($_POST['password']) && !empty($_POST['confirm-password'])) {
```

- le mot de passe correspond à la confirmation de mot de passe ?

```php
if ($_POST['password'] === $_POST['confirm-password']) {
```

  - si non, redirection + message d’erreur à l’aide du $_GET

```php
header('location:signUp.php?error=1&message=Les mots de passes ne correspondent pas !');
exit();
```

  - Si oui, on récupère toutes les informations des superglobales $_POST() (la valeur de chaque input au moment de la soumission du formulaire) en le transformant via htmlspecialchars().

```php
$firstname = htmlspecialchars($_POST['firstname']);
$lastname = htmlspecialchars($_POST['lastname']);
$email = htmlspecialchars($_POST['email']);
$pass1 = htmlspecialchars($_POST['password']);
$pass2 = htmlspecialchars($_POST['confirm-password']);
```

- On vérifie la syntaxe de l’email, comme expliqué plus haut :

```php
// verifications du mail (au cas où l'utilisateur change le type de l'input)
if (!Verify::verifySyntax($email)) {
    header('location:signUp.php?error=1&message=merci de rentrer un email valide !');
    exit();
}
$user = new User();
```

- si non, redirection et affichage d'un message d'erreur
- si oui, on instancie l'objet User().

- On vérifie encore une fois que l'email n'existe pas déjà :

```php
// doublon mail
if ($user->availableEmail($email) !== 0) {
    header('location:signUp.php?error=1&message=adresse email déjà utilisée !');
    exit();
}
```

- s'il existe, redirection et affichage d'un message d'erreur
- sinon on continue

- On hache le mot de passe et on redirige vers le formulaire de connexion

```php
  $password = password_hash($pass1, PASSWORD_BCRYPT);
header('location:./signIn.php');
  exit();
```

Le code complet :

```php
if (isset($_POST['signUp'])) {
    if (!empty($_POST['firstname']) && !empty($_POST['lastname']) && !empty($_POST['email']) &&
!empty($_POST['password']) && !empty($_POST['confirm-password'])) {
        if ($_POST['password'] === $_POST['confirm-password']) {

            $firstname = htmlspecialchars($_POST['firstname']);
            $lastname = htmlspecialchars($_POST['lastname']);
            $email = htmlspecialchars($_POST['email']);
            $pass1 = htmlspecialchars($_POST['password']);
            $pass2 = htmlspecialchars($_POST['confirm-password']);


            // verifications du mail (au cas où l'utilisateur change le type de l'input)
            if (!Verify::verifySyntax($email)) {
                header('location:signUp.php?error=1&message=merci de rentrer un email valide !');
                exit();
            }
            $user = new User();

            // doublon mail
            if ($user->availableEmail($email) !== 0) {
                header('location:signUp.php?error=1&message=adresse email déjà utilisée !');
                exit();
            }

            $password = password_hash($pass1, PASSWORD_BCRYPT);
          header('location:./signIn.php');
            exit();
        } else {
            header('location:signUp.php?error=1&message=Les mots de passes ne correspondent pas !');
            exit();
        }
    }
}
```

On affiche un message via notre redirection en cas d'erreurs.

```php
<div>
  <span id="errorMsg" class="text-rose-600">
  <?php
    if (isset($_GET['error']) && !empty($_GET['message'])) {
        echo htmlspecialchars($_GET['message']);
      }
      ?>
  </span>
</div>
```

2. Panier

2.1 Ajout d'un article (via page product)

Pour la gestion du panier, à l'heure actuelle l'utilisateur doit être connecté.

Une fois sur la fiche produit, l'utilisateur a la possibilité d'ajouter au panier une fois la taille et la couleur sélectionnée.

Suite à ça on envoie un **fetch** avec la méthode **POST**
Important : on transforme via Number() nos valeurs pour quantity et size_id, pour être certain de récupérer un type **INT** pour notre base de données, sinon la requête **SQL** échouera.

Une fois soumis, on réinitialise la valeur de l'input quantité à 1 et on ré-associe la variable myQty à inputQty.value pour faire correspondre la vue au traitement ultérieur, au cas où l'utilisateur voudrait ajouter le même produit.

```javascript
btnAddToCart.addEventListener('click', function (){
  if(mySize !== null & myQty > 0){

    fetch('./php/Controller/verifAddToCart.php', {
        method: "POST",

        body: JSON.stringify({
            product_id: product.id,
            color: colorChoice,
            color_id: myColorId,
            size: mySize,
            size_id: Number(mySizeId),
            quantity: Number(myQty),
            price: price[0].price,
            price_id: price[0].id,
        }),
        headers: {
            "Content-Type": "application/json",
        },
    })
    .then(function (response){
        inputQty.value = 1;
        myQty = inputQty.value;

        // on fait un toast de notification
```

Côté serveur :

Comme toutes nos requêtes SQL, sont utilisées par des méthodes dans des classes. On instancie les classes : Cart et Orderdetails pour avoir accès à leurs méthodes respectives, et coder notre logique.

```php
$myCart = new Cart();
$myOrderdetails = new Orderdetails();
```

- On incrémente si le produit correspond parfaitement (couleur et taille).

```php
// on incrémente l'ancienne valeur déjà présente en base de données
$myCart->updateQuantity($quantity + $oldQty, $oldCartId);
```

- On ajoute l'article au panier s'il n'est pas présent.

```php
} else {
    $myCart->addNew($productId, $colorId, $sizeId, $quantity, $priceId, $userId);
```

- On supprime la commande en cours, si elle existe (non finalisé, avec le panier précédent).

```php
$rowOrder = $myOrderdetails->alreadyAdded($userId);
if ($rowOrder !== 0) $myOrderdetails->deleteRow($userId);
```

- On met à jour le panier.

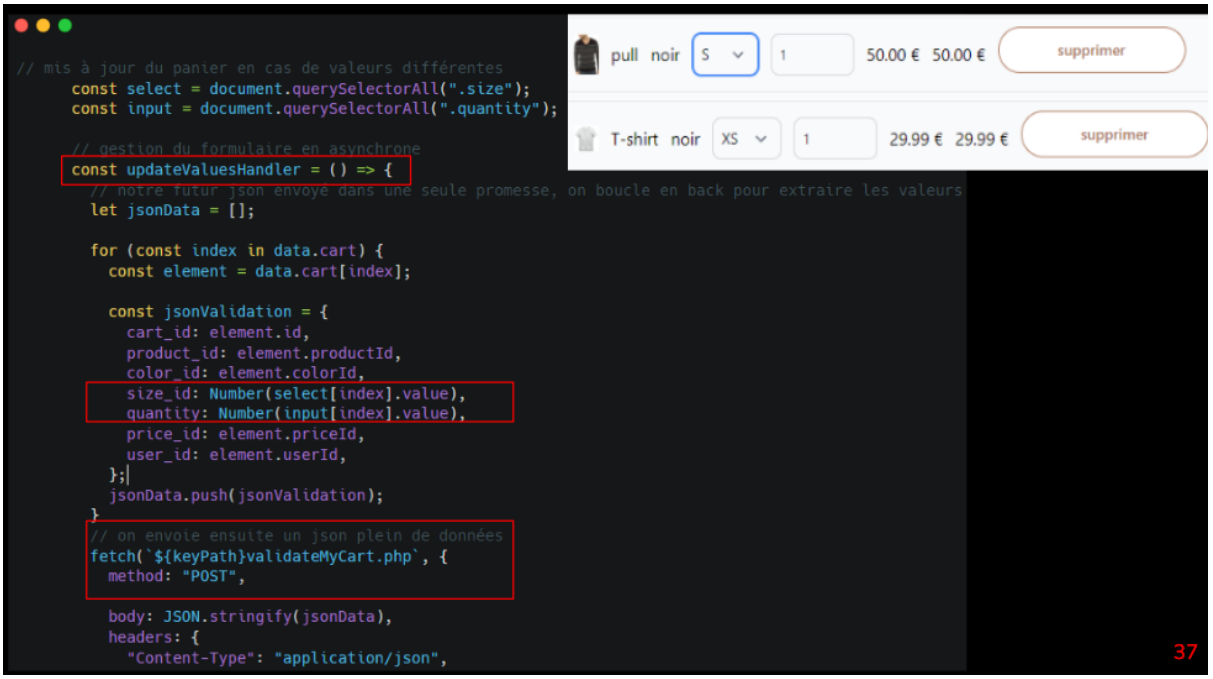
ici le code complet:

```php
$data = json_decode(file_get_contents('php://input'), true);

$productId = $data['product_id'];
$colorName = $data['color'];
$colorId = $data['color_id'];
$sizeName = $data['size'];
$sizeId = $data['size_id'];
$quantity = $data['quantity'];
$price = $data['price'];
$priceId = $data['price_id'];

spl_autoload_register(function($classes) {
    require_once('../' .$classes. '.php');
});
use Classes\Cart;
use classes\Orderdetails;
$myCart = new Cart();
$myOrderdetails = new Orderdetails();

$row = $myCart->productAlreadyAdded($productId, $colorId, $sizeId, $userId);
$rowOrder = $myOrderdetails->alreadyAdded($userId);
if ($rowOrder !== 0) $myOrderdetails->deleteRow($userId);
if ($row !== 0) {
  $oldCart = $myCart->getQtyByRow($productId, $colorId, $sizeId, $userId);
  $oldQty = $oldCart['quantity'];
  $oldCartId  = $oldCart['id'];
  // on incrémente l'ancienne valeur déjà présente en base de données
  $myCart->updateQuantity($quantity + $oldQty, $oldCartId);
} else {
    $myCart->addNew($productId, $colorId, $sizeId, $quantity, $priceId, $userId);
```

2.2 mise à jour/ suppression d'un/plusieurs produits

Une fois sur son panier l'utilisateur à la possibilité de mettre à jour par produit choisi en **JS asynchrone** (fetch post):

- taille
- quantité



Coté serveur:

On va boucler pour mettre à jour le panier

- on met à jour les produits qui ont été modifiés
- on supprime si le produit du panier a une quantité égale à 0

```php
$data = json_decode(file_get_contents('php://input'), true);

$myCart = new Cart();

foreach ($data as $key => $element) {

    $cartId = $element['cart_id'];
    $productId = $element['product_id'];
    $colorId = $element['color_id'];
    $sizeId = $element['size_id'];
    $quantity = $element['quantity'];
    $priceId = $element['price_id'];

    $cart = $myCart->getById($cartId); // <-- ici on récupère les valeurs de la bdd qu'on va
    // condition pour update
    if ($cart['id'] === $cartId && $cart['product_id'] === $productId) {

        if ($quantity === 0) {
            $myCart->deleteRow($cartId);
        } else {
            if ($cart['quantity'] != $quantity) $myCart->updateQuantity($quantity, $cartId);
            if ($cart['size_id'] != $sizeId) $myCart->updateSize_id($sizeId, $cartId);
```

3. Confirmation de commande

Une fois le panier, le choix de la livraison et le transporteur sont validés. Nous récupérons toutes les données en dur, pour la table orderdetails.

Nous sommes parti du principe que :

- toutes les lignes de paniers (un produit par ligne, en table SQL) qui correspondent à un utilisateur.
- et qu'il ne peut y avoir qu'une seule commande en cours (non payé/finalisé) par utilisateur.

APRÈS que la table orderfinal (copie de orderdetails avec la colonne payement_status en plus) soit incrémenté d'une ligne concernant l'utilisateur en cours. Les lignes de la table cart et la ligne de la table orderdetails seront supprimées.

Dans cet exemple de code nous simulons l'acceptation de la banque pour le paiement de la commande. et nous envoyons l'objet JSON.stringify(). A l'intérieur nous aurons le payement_status.

```
// récupération de la commande en cours
  fetch(`${keyPath}checkMyOrder.php`) // a remplacer plus tard par une API : paypal /stripe
    .then((response) => response.json())
    .then((bank) => {
      // lorsque la banque accepte le paiement on envoie les données à la table orderfinal pour
terminer l'achat
      const bankResponse = true; // simulation de la variable récupéré
      const jsonOrderData = { payement_status: bankResponse };

    //   on remplit la table orderfinal
      fetch(`${keyPath}addToOrderfinal.php`, {
        method: "POST",

        body: JSON.stringify(jsonOrderData),
        headers: {
          "Content-Type": "application/json",
        },
      })
        .then((response) => {
          if (response.ok) {
              window.location.href = "./myorders.php";
          } else {
            alert("problème detecté merci de contacter un administrateur.");
          }
        })
        .catch(function (error) {
          console.log(error);
        });
    });
```

côté serveur:

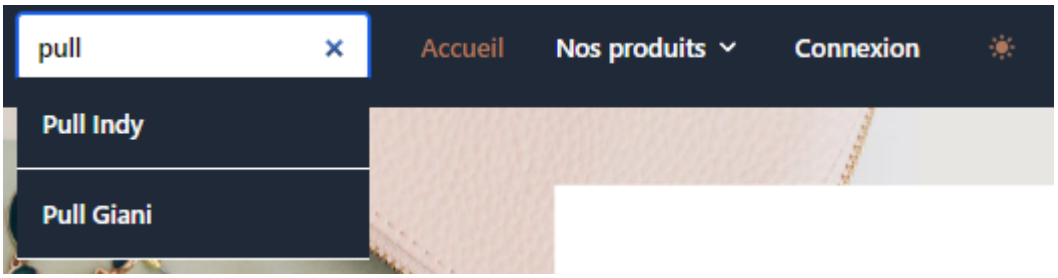
Nous faisons une double vérification:

- Si la table orderdetails existe on récupère les données et on exécute la méthode addNew() de la classe Orderfinal.
- Si la requête de création de la ligne (addNew()) dans la table orderfinal à été un succès, alors à ce moment là on supprime la ligne de la orderdetails et les lignes cart correspondantes à l'utilisateur en question.

Le panier est donc vidé et il n'y a plus de commande en cours car elle est à présent payée.

```php
<?php
session_start();

$userId = $_SESSION["userId"];

// Récupérer les données envoyées depuis la requête fetch POST
$data = json_decode(file_get_contents('php://input'), true);

$payement_status      = $data['payement_status'];

require_once('../Classes/Orderdetails.php');
require_once('../Classes/orderfinal.php');
require_once('../Classes/Cart.php');

use Classes\Orderfinal;
use Classes\Cart;
use Classes\Orderdetails;

$myOrderfinal = new Orderfinal();
$myCart = new Cart();
$myOrderdetails = new Orderdetails();
$result = $myOrderdetails->getByUserId($userId);
// si la commande existe existe alors on peut continuer le tunnel d'achat
if ($result) {

    $deliveryFullAddress  = $result['deliveryFullAddress'];
    $carriersDetails      = $result['carriersDetails'];
    $carriers_price       = $result['carriers_price'];
    $product_ids          = $result['product_ids'];
    $color_ids            = $result['color_ids'];
    $size_ids             = $result['size_ids'];
    $product_names        = $result['product_names'];
    $color_names          = $result['color_names'];
    $size_names           = $result['size_names'];
    $price_values         = $result['price_values'];
    $quantity             = $result['quantity'];
    $total_amount         = $result['total_amount'];

    $orderSuccess = $myOrderfinal->addNew($deliveryFullAddress, $carriersDetails, $carriers_price,
$product_ids, $color_ids, $size_ids,$product_names,$color_names,$size_names,$price_values, $quantity,
$total_amount, $payement_status, $userId);
    // si la table orderfinal est rempli alors on vide les tables cart et orderdetails qui contiennent
cet id_user
    if ($orderSuccess) {
        $myCart->deleteRowByUserId($userId);
        $myOrderdetails->deleteRowByUserId($userId);
    }
}

// Répondre avec une réponse JSON indiquant le succès ou l'échec de l'opération
$response = array('success' => true);
echo json_encode($response);
?>
```

4. Autocomplétion en JS asynchrone



Le but de la recherche avec autocomplétion est d'afficher à l'utilisateur tous les produits existant qui comportent le résultat de sa recherche sous forme de liste SANS rechargement de page.

Ici nous avons un input de type search auquel nous avons rattaché, un événement keyup et un fetch post, qui ira chercher la réponse du serveur à un requête SQL LIKE %"input.value"%

- l'événement Javascript + fetch post

```
searchBar.addEventListener("keyup", () => {
  results.innerHTML = "";
  const jsonValue = { name: searchBar.value };

  if (searchBar.value.length > 0) {
    fetch(`${keyPath}searchProducts.php`, {
      method: "POST",

      body: JSON.stringify(jsonValue),
```

- la méthode getSearchLike("ma recherche dans l'input") de la classe Products

```
public function getSearchLike($value) {
    file_exists('../DB/DBManager.php') ? require('../DB/DBManager.php') : require('./php/DB/DBManager.php');
    $request = $bdd->prepare("SELECT * FROM ".$this::TABLE_NAME." WHERE `name` LIKE ?");
    $request->execute(['%' .$value. '%']);
    $response = $request->fetchAll(PDO::FETCH_ASSOC);
    return $response;
}
```

- Le back-end prêt à renvoyer la réponse à la requête SQL LIKE

```
// Récupérer les données envoyées depuis la requête fetch POST
$data = json_decode(file_get_contents('php://input'), true);

$inputValue = $data['name'];

$myProducts = new Products();

$mySearch = $myProducts->getSearchLike($inputValue);

// Répondre avec une réponse JSON indiquant le succès ou l'échec de l'opération
$response = array($mySearch);
echo json_encode($response);
```

Une fois la réponse du serveur transmise et grâce aux chaînage des fonctions callback ".then()" nous récupérons un array de produits avec toutes les colonnes SQL (id, name, description) associées, nous allons nous en servir pour créer une liste de liens, qui permettent de rediriger l'utilisateur vers la page du produit concerné.

```
import { addElement } from "../modules/addElement.js";
import { keyPath } from "../modules/key.js";

const searchBar = document.querySelector("#search");
const results = document.querySelector("#results");

searchBar.addEventListener("keyup", () => {
  results.innerHTML = "";
  const jsonValue = { name: searchBar.value };

  if (searchBar.value.length > 0) {
    fetch(`${keyPath}searchProducts.php`, {
      method: "POST",

      body: JSON.stringify(jsonValue),
      headers: {
        "Content-Type": "application/json",
      },
    })
      .then((response) => {
        if (response.ok) {
          response.text().then((result) => {
            const items = JSON.parse(result);
            for (const key in items[0]) {
              const nameProduct = items[0][key].name;
              const idProduct = items[0][key].id;
              console.log(nameProduct);

              const link = addElement(
                "a",
                ["block", "bg-white", "p-4", "dark:bg-gray-800", "dark:text-white", "border-b", "border-white"],
                { href: `./product.php?name=${nameProduct}&id=${idProduct}` },
                `${nameProduct}`
              );
              results.append(link);
            }
          });
        } else {
          alert("problème détecté merci de contacter un administrateur.");
        }
      })
      .catch(function (error) {
        console.log(error);
```

NB: "keypath" est juste un chemin d'accès, ça nous évitait de le taper à chaque fois, "addElement" est une fonction qui permet de créer et personnaliser un élément dans le **DOM** rapidement.

Elle se décompose de la façon suivante:

- addElement(argument 1 = type ,argument 2 = ce sont les classes, argument 3 = attributs, argument 4 = textContent)
    - arg1 : c'est l'équivalent de la fonction createElement("ma balise html")
    - arg2: c'est l'équivalent de classlist.add(["toutes mes classes"])
    - arg3: c'est l'équivalent de setAttribute({mes attributs : "sa valeur" })
    - arg4: c'est l'équivalent de textContent = "le text que je veux afficher"

V. Veille sur les vulnérabilités de sécurité

Protection des données sensibles

La sécurité web vise à protéger les informations sensibles: données personnelles, informations financières, mots de passe, contre les cyberattaques.

Afin de protéger ces données sensibles plusieurs bonnes pratiques sont à mettre en place, dans ce projet nous avons accordé une importance particulière à cela. Voici quelques exemples :

1. Faille XSS

La faille XSS (Cross Site Scripting) est une faille de sécurité permettant d'injecter du contenu dans une page. L'attaque peut être faite en utilisant tous les langages pris en charge par le navigateur tel que le JavaScript. Il suffit pour cela d'injecter un code JavaScript dans l'URL ou dans un champ de formulaire.

1.1 Protection contre les failles XSS

Il existe plusieurs techniques pour se prémunir des failles XSS notamment en PHP :

- Utiliser la fonction htmlspecialchars() qui filtre les chevrons (< et >).
- Utiliser la fonction htmlentities() qui filtre les caractères équivalents au codage HTML ou JS.

```php
$email = htmlspecialchars($_POST['email']);
$pass1 = htmlspecialchars($_POST['password']);
```

2. Faille CSRF

La faille CSRF (Cross Site Request Forgery) est une faille où un utilisateur malveillant fait en sorte que vos utilisateurs soumettent sans le savoir des données qu'ils n'ont pas l'intention de soumettre.

1. Prévention contre les failles CSRF

Pour se prémunir contre les failles CSRF on peut :

- Demander des confirmations à nos utilisateurs

- Demander des informations antérieures telles qu'un ancien mot de passe pour en changer
- Utiliser des tokens style (JWT)
- Eviter d'utiliser la requête HTTP GET, car les données sont passés en clair

3. Eviter les mots de passe en clair : hachage des mots de passe

Tous les mots de passe lors de l'inscription sont hachés via un algorithme de hachage BCRYPT.

```php
$password = password_hash($pass1, PASSWORD_BCRYPT);
```

Pour la connexion on utilisera password_verify() pour la correspondance des mots de passe.

```php
public function passVerify($email, $pass) {
    file_exists('../DB/DBManager.php') ? require('../DB/DBManager.php') : require('./php/DB/DBManager.php');
    $request = $bdd->prepare("SELECT `password` FROM ".$this::TABLE_NAME." WHERE email = ?");
    $request->execute([$this->setEmail($email)]);
    $row = $request->fetch();

    return $row && password_verify($pass, $row['password']) ? $row['password'] : false;

}
```

4. Eviter les injections SQL : les requêtes préparés

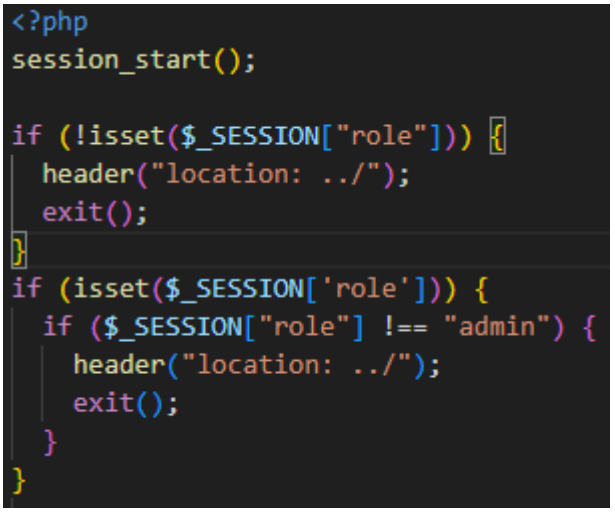
Absolument toutes les requêtes sont préparées sur le site et utilisent toutes l'objet PDO. Les données seront interprétées indépendamment de la requête, pour prévenir des injections SQL. De plus, la majorité des entrées des formulaires sont transformés via htmlspecialchars(). Ceci afin d'éviter que des scripts malveillants soient interprétés via la valeur d'un ou plusieurs champs de formulaire.

Exemple :

```php
// classe parente ne servant qu'à être hérité
class SearchByUser_id {

    const TABLE_NAME = "";

    public function getAllByUserId($userId) {
        file_exists('../DB/DBManager.php') ? require('../DB/DBManager.php') : require('./php/DB/DBManager.php');
        $request = $bdd->prepare("SELECT * FROM ".$this::TABLE_NAME." WHERE user_id = ? ");
        $request->execute([$userId]);
        $response = $request->fetchAll(PDO::FETCH_ASSOC);
        return $response;
    }
```

5. Contrôler les accès

Par exemple, un utilisateur non administrateur ne peut pas avoir accès au panel d'administration. Dans l'exemple ci-dessous nous fermons la porte dans deux conditions, si l'utilisateur n'est pas authentifié et si l'utilisateur n'as pas le rôle "admin".



```php
<?php
session_start();

if (!isset($_SESSION["role"])) {
  header("location: ../");
  exit();
}
if (isset($_SESSION['role'])) {
  if ($_SESSION["role"] !== "admin") {
    header("location: ../");
    exit();
  }
}
```

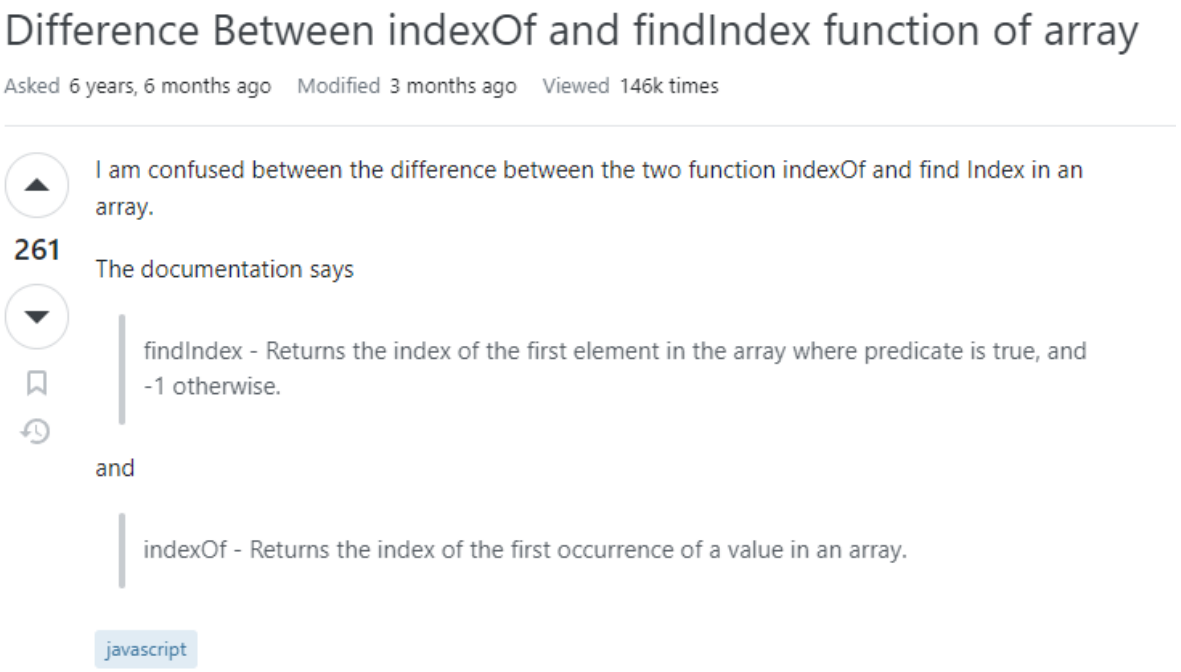
6. Se tenir informé

On peut également lire des articles sur des sites comme :

- owasp.org
- /www.cybersecurity-guide.com

VI. Recherche effectué sur un site anglophone

Pour pouvoir faire une vérification en front avec javascript, j'avais besoin de trouver une méthode pour savoir si le nom de l'adresse de livraison existait déjà dans mon tableau où je stocke tous les nom d'adresses, car ils sont uniques pour chaque utilisateur. Par exemple, "maison" ne peut être utilisé qu'une seule fois par utilisateur, pour éviter les confusions.



Difference Between indexOf and findIndex function of array

Asked 6 years, 6 months ago Modified 3 months ago Viewed 146k times

261

I am confused between the difference between the two function indexOf and find Index in an array.

The documentation says

> findIndex - Returns the index of the first element in the array where predicate is true, and -1 otherwise.

and

> indexOf - Returns the index of the first occurrence of a value in an array.

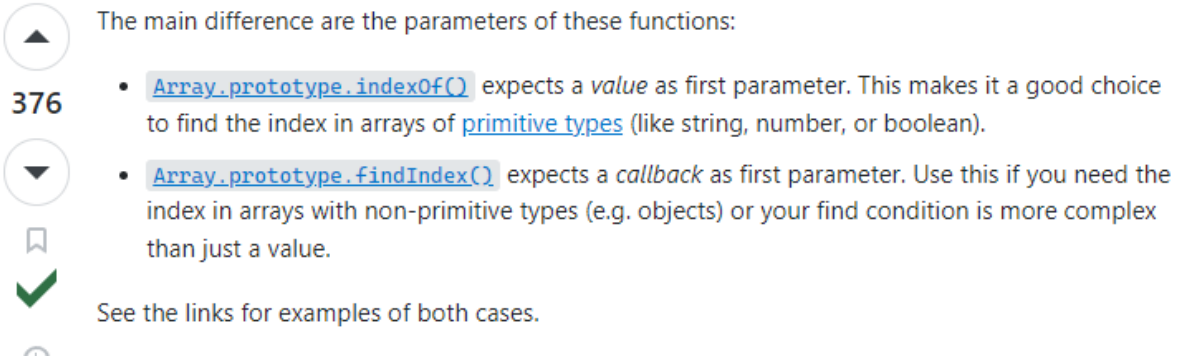
javascript

Ici une personne dit qu'elle est confuse sur la différence entre 2 fonctions à savoir : indexOf et findIndex dans un tableau.
Il dit que la documentation dit :

- findIndex - retourne l'index du premier élément du tableau où le prédicat est vrai et sinon renvoie -1.

et

- indexOf - renvoie l'index de la première occurrence de la valeur dans un tableau.



376

The main difference are the parameters of these functions:

- `Array.prototype.indexOf()` expects a *value* as first parameter. This makes it a good choice to find the index in arrays of primitive types (like string, number, or boolean).
- `Array.prototype.findIndex()` expects a *callback* as first parameter. Use this if you need the index in arrays with non-primitive types (e.g. objects) or your find condition is more complex than just a value.

See the links for examples of both cases.

On lui répond que la principale différence sont les paramètres de cette fonction:

- pour indexOf() : attend une valeur comme premier paramètre. C'est un bon choix pour trouver un index dans des tableaux avec des types primitifs comme des "string" "number" "boolean" (chaîne de caractères, nombres, booléens).
- pour findIndex() : attend une fonction callback en 1er paramètre. Utilisez ceci si vous avez besoin de l'index dans des tableaux avec des types non-primitifs comme les objets ou si votre condition de recherche est plus complexe que juste une valeur.

J’ai pu coder grâce à ça une fonction qui s'exécute si la valeur de retour de indexOf() n’est pas égale à -1 (si la valeur existe dans le tableau).

```
// si le nom est en doublon on bloque le formulaire
nameAddress.addEventListener("keyup", () => {
  if (arrayDeliveryNames.indexOf(nameAddress.value) !== -1) {
    confirmAddress.disabled = true;
    confirmAddress.style.cursor = "not-allowed";
    errorMsg.textContent = "Nom déjà existant !";
  } else {
    confirmAddress.disabled = false;
    confirmAddress.style.cursor = "";
    errorMsg.textContent = "";
  }
});
```

Axe d’améliorations

Ils sont nombreux, il serait difficile de tous les citer. Néanmoins quelque uns peuvent être immédiatement pointé:

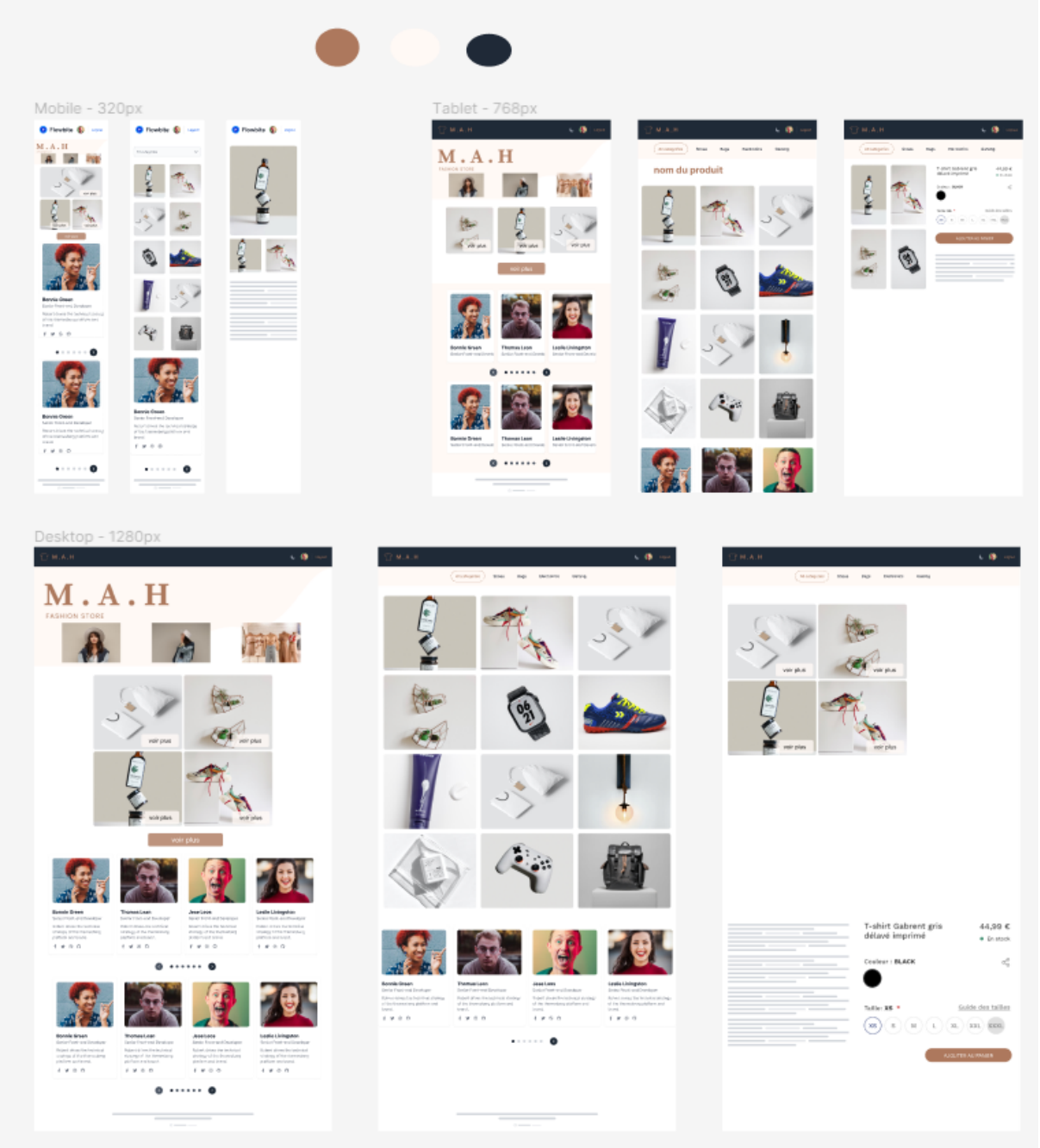
- Stocker dans un $_SESSION /$_COOKIE les produits d’un utilisateur déconnecté, pour ensuite créer un algorithme qui les associera réellement à la table ‘cart’ une fois l’utilisateur connecté.
- Revoir peut-être l’architecture de la BDD en elle-même, notre façon de la concevoir nous a amené à devoir faire des choses compliquées. Notamment pour les relations entre products, color et size. Le fait de devoir gérer 2 options sur un même produit était un challenge et je pense que nous aurions pu voir la chose différemment. Nous avons réussi, mais pas forcément avec la bonne approche.
- Suppression des utilisateurs.
- Ajouter le module de paiement.
- Améliorer le design

Remerciements

Je tenais à remercier mes collègues de l’équipe de développement et La Platefome_ pour ce projet très intéressant tant sur le plan technique que gestion de projet et organisationnel. Il m’a permis d’améliorer mes compétences en SQL, JS asynchrone et algorithmie.

ANNEXES

I. Maquette

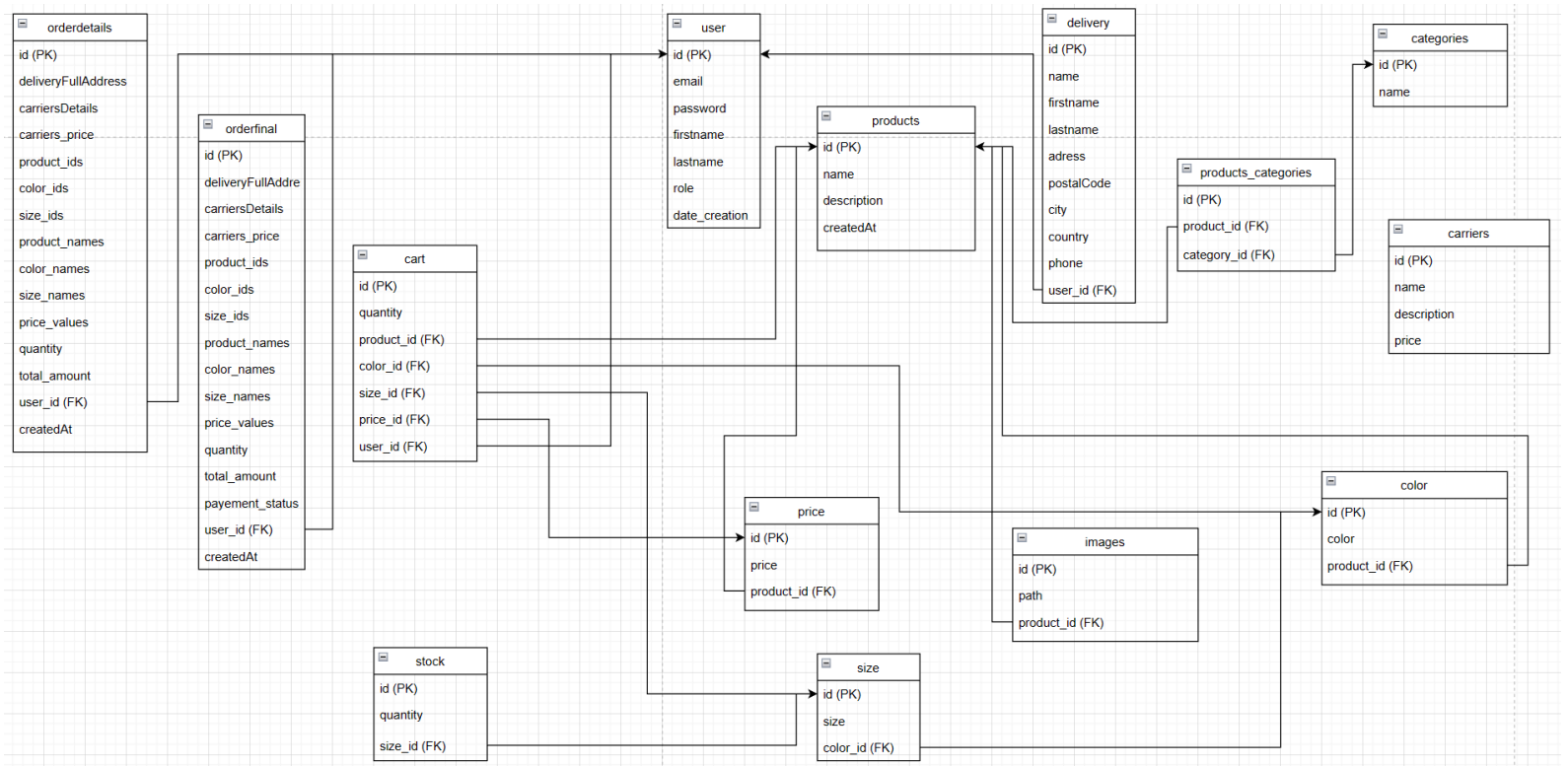


II. Les Breakpoints (media queries de tailwindcss)

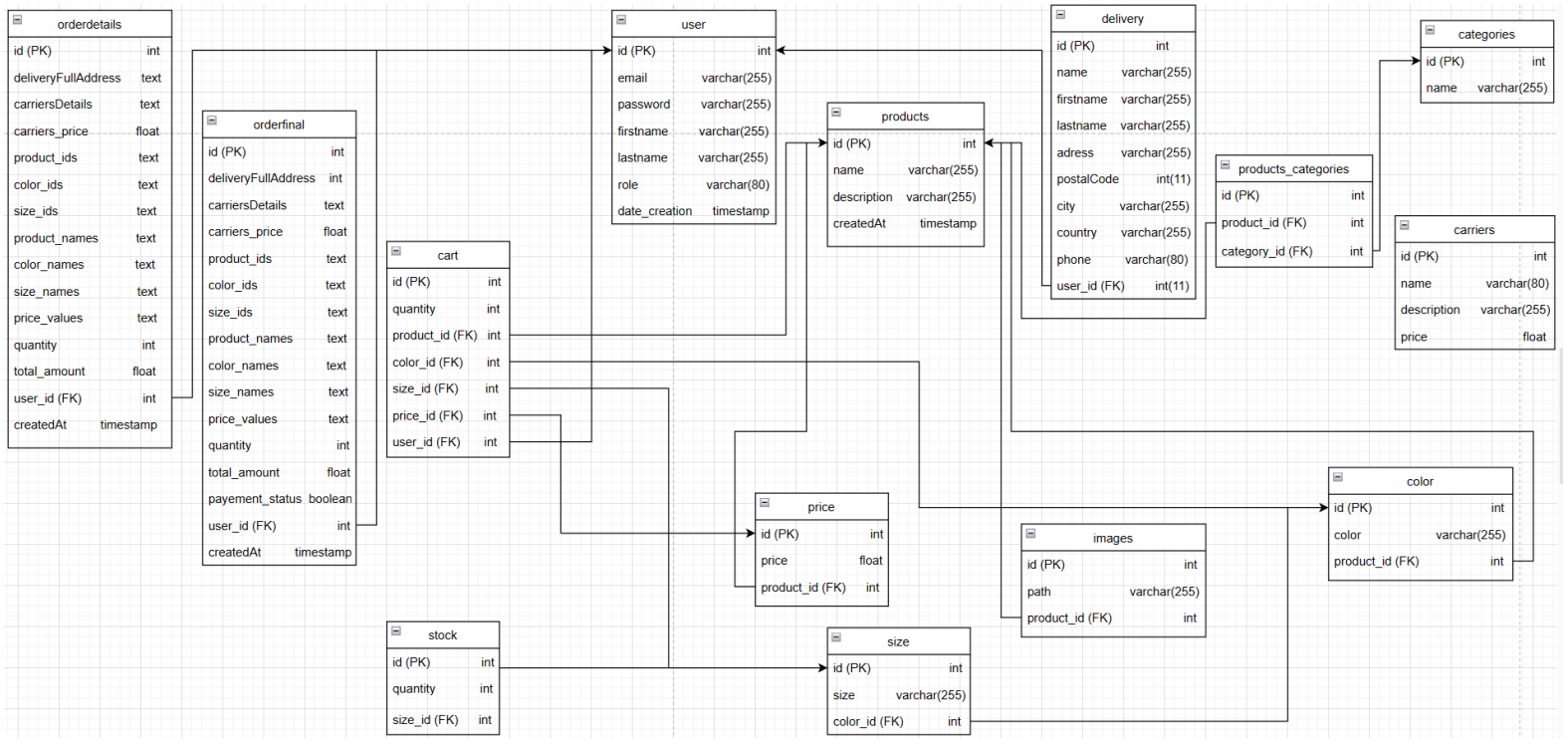
tailwind.config.js

```
module.exports = {
  theme: {
    screens: {
      'sm': '640px',
      // => @media (min-width: 640px) { ... }

      'md': '768px',
      // => @media (min-width: 768px) { ... }

      'lg': '1024px',
      // => @media (min-width: 1024px) { ... }

      'xl': '1280px',
      // => @media (min-width: 1280px) { ... }

      '2xl': '1536px',
      // => @media (min-width: 1536px) { ... }
    }
  }
}
```

III. Modèle logique de données



IV. Modèle physique de données



V. Le fichier de création automatique de la base de données

```php
<?php
require_once('php/DB/DBManager.php');

    $tables = [
        'user' => 'CREATE TABLE `user` (
                    `id` INT AUTO_INCREMENT PRIMARY KEY,
                    `email` varchar(255) NOT NULL UNIQUE,
                    `password` varchar(255) NOT NULL,
                    `firstname` varchar(255) NOT NULL,
                    `lastname` varchar(255) NOT NULL,
                    `role` varchar(80) NOT NULL,
                    `createdAt` timestamp NOT NULL DEFAULT current_timestamp()
                    ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',

        'categories' => 'CREATE TABLE `categories` (
                        `id` INT AUTO_INCREMENT PRIMARY KEY,
                        `name` varchar(255) NOT NULL
                        ) ENGINE=InnoDB DEFAULT CHARSET=utf8;',

        'products' => 'CREATE TABLE `products` (
                        `id` INT AUTO_INCREMENT PRIMARY KEY,
                        `name` varchar(255) NOT NULL,
                        `description` varchar(255) DEFAULT NULL,
                        `createdAt` timestamp NOT NULL DEFAULT current_timestamp()
                        ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',

        'prod_cat' => 'CREATE TABLE `products_categories` (
                        `id` INT AUTO_INCREMENT PRIMARY KEY,
                        `product_id` INT NOT NULL,
                        `category_id` INT NULL,
                        FOREIGN KEY (`product_id`) REFERENCES `products`(`id`),
                        FOREIGN KEY (`category_id`) REFERENCES `categories`(`id`) ON DELETE SET NULL
                        ) ENGINE=InnoDB DEFAULT CHARSET=utf8;',

        'image' => 'CREATE TABLE `images` (
                    `id` INT AUTO_INCREMENT PRIMARY KEY,
                    `path` VARCHAR(255) NOT NULL,
                    `product_id` INT NOT NULL,
                    FOREIGN KEY (`product_id`) REFERENCES `products`(`id`) ON DELETE CASCADE
                    ) ENGINE=InnoDB DEFAULT CHARSET=utf8;',

        'price' => 'CREATE TABLE `price` (
                    `id` INT AUTO_INCREMENT PRIMARY KEY,
                    `price` FLOAT NOT NULL,
                    `product_id` INT NOT NULL,
                    FOREIGN KEY (`product_id`) REFERENCES `products`(`id`) ON DELETE CASCADE
                    ) ENGINE=InnoDB DEFAULT CHARSET=utf8;',

        'color' => 'CREATE TABLE `color` (
                    `id` INT AUTO_INCREMENT PRIMARY KEY,
                    `color` varchar(255) NOT NULL,
                    `product_id` INT NOT NULL,
                    FOREIGN KEY (`product_id`) REFERENCES `products`(`id`) ON DELETE CASCADE
                    ) ENGINE=InnoDB DEFAULT CHARSET=utf8;',

        'size' => 'CREATE TABLE `size` (
                    `id` INT AUTO_INCREMENT PRIMARY KEY,
                    `size` varchar(255) NOT NULL,
                    `color_id` INT NOT NULL,
                    FOREIGN KEY (`color_id`) REFERENCES `color`(`id`) ON DELETE CASCADE
                    ) ENGINE=InnoDB DEFAULT CHARSET=utf8;',

        'stock' => 'CREATE TABLE `stock` (
                    `id` INT AUTO_INCREMENT PRIMARY KEY,
                    `quantity` INT NOT NULL,
                    `size_id` INT NOT NULL,
                    FOREIGN KEY (`size_id`) REFERENCES `size`(`id`) ON DELETE CASCADE
                    ) ENGINE=InnoDB DEFAULT CHARSET=utf8;',
```

```
'carriers' => 'CREATE TABLE `carriers` (
                `id` INT AUTO_INCREMENT PRIMARY KEY,
                `name` varchar(80) NOT NULL,
                `description` varchar(255) NOT NULL,
                `price` float NOT NULL
                ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',

'delivery' => 'CREATE TABLE `delivery` (
                `id` INT AUTO_INCREMENT PRIMARY KEY,
                `name` varchar(255) NOT NULL,
                `firstname` varchar(255) NOT NULL,
                `lastname` varchar(255) NOT NULL,
                `address` varchar(255) NOT NULL,
                `postalCode` int(11) NOT NULL,
                `city` varchar(255) NOT NULL,
                `country` varchar(255) NOT NULL,
                `phone` varchar(80) NOT NULL,
                `user_id` int(11) NOT NULL,
                FOREIGN KEY (`user_id`) REFERENCES `user`(`id`)
                ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',

'cart' => 'CREATE TABLE `cart` (
                `id` INT AUTO_INCREMENT PRIMARY KEY,
                `quantity` INT NOT NULL,
                `product_id` INT NOT NULL,
                `color_id` INT NOT NULL,
                `size_id` INT NOT NULL,
                `price_id` INT NOT NULL,
                `user_id` INT NOT NULL,
                FOREIGN KEY (`user_id`) REFERENCES `user`(`id`),
                FOREIGN KEY (`price_id`) REFERENCES `price`(`id`),
                FOREIGN KEY (`size_id`) REFERENCES `size`(`id`),
                FOREIGN KEY (`product_id`) REFERENCES `products`(`id`),
                FOREIGN KEY (`color_id`) REFERENCES `color`(`id`),
                `createdAt` timestamp NOT NULL DEFAULT current_timestamp()
                ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',

'orderDetails' => 'CREATE TABLE `orderdetails` (
                    `id` INT AUTO_INCREMENT PRIMARY KEY,
                    `deliveryFullAddress` TEXT NOT NULL,
                    `carriersDetails` TEXT NOT NULL,
                    `carriers_price` FLOAT NOT NULL,
                    `product_ids` TEXT NOT NULL,
                    `color_ids` TEXT NOT NULL,
                    `size_ids` TEXT NOT NULL,
                    `product_names` TEXT NOT NULL,
                    `color_names` TEXT NOT NULL,
                    `size_names` TEXT NOT NULL,
                    `price_values` TEXT NOT NULL,
                    `quantity` INT NOT NULL,
                    `total_amount` FLOAT NOT NULL,
                    `user_id` INT NOT NULL,
                    FOREIGN KEY (`user_id`) REFERENCES `user`(`id`),
                    `createdAt` timestamp NOT NULL DEFAULT current_timestamp()
                    ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',
```

```
126         'orderFinal' => 'CREATE TABLE `orderfinal` (
127                             `id` INT AUTO_INCREMENT PRIMARY KEY,
128                             `deliveryFullAddress` TEXT NOT NULL,
129                             `carriersDetails` TEXT NOT NULL,
130                             `carriers_price` FLOAT NOT NULL,
131                             `product_ids` TEXT NOT NULL,
132                             `color_ids` TEXT NOT NULL,
133                             `size_ids` TEXT NOT NULL,
134                             `product_names` TEXT NOT NULL,
135                             `color_names` TEXT NOT NULL,
136                             `size_names` TEXT NOT NULL,
137                             `price_values` TEXT NOT NULL,
138                             `quantity` INT NOT NULL,
139                             `total_amount` FLOAT NOT NULL,
140                             `payement_status` BOOLEAN NOT NULL,
141                             `user_id` INT NOT NULL,
142                             FOREIGN KEY (`user_id`) REFERENCES `user`(`id`),
143                             `createdAt` timestamp NOT NULL DEFAULT current_timestamp()
144                             ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',
145 
146         ];
```