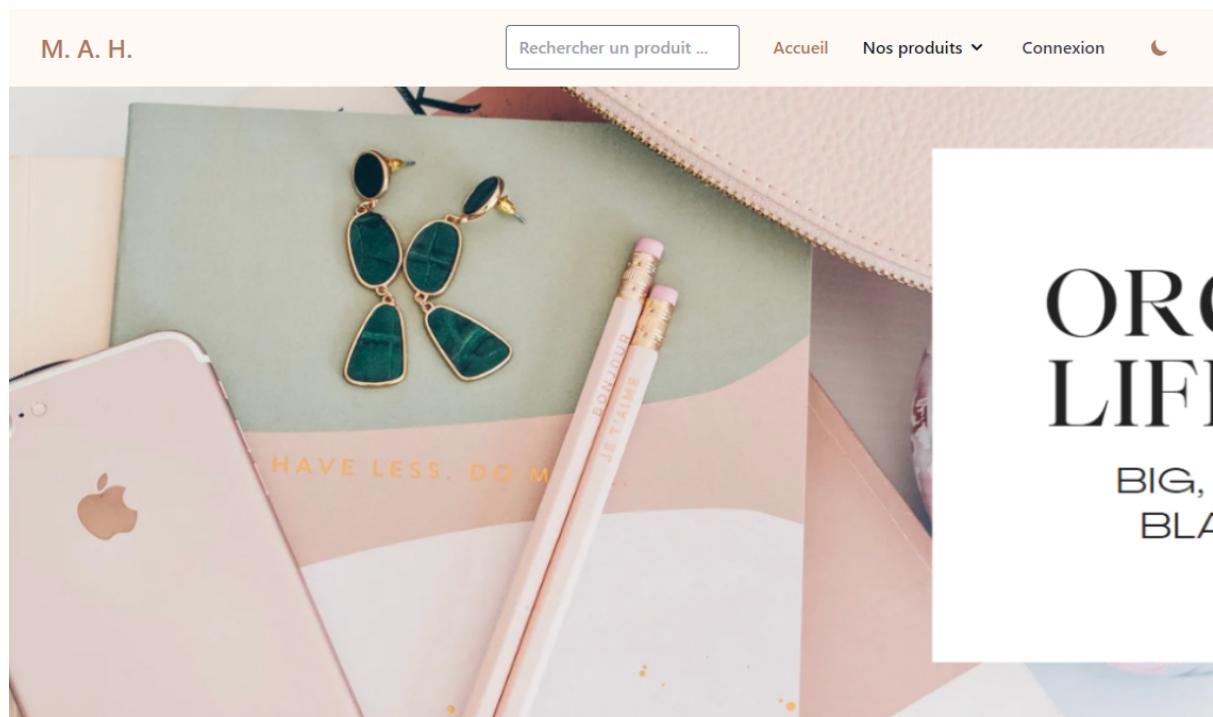


DOSSIER DE PROJET PROFESSIONNEL

Titre : Développeur Web et Web Mobile



Aurélien GALLEA

Table des matières

Compétences du référentiel couvertes par le projet	4
Résumé	4
Spécifications fonctionnelles	5
I. Expression des besoins	5
II. Arborescence du site	5
III. Description des fonctionnalités	5
1. Authentification	5
2. Profil utilisateur	5
3. Catalogue Des produits, filtres et recherches	6
4. Fiche produit	6
5. Panier client	6
6. Livraison	6
7. Récapitulatif de commande et paiement	6
8. Commandes utilisateur	6
9. Back office	6
9.1 Gestion des catégories produits	6
9.2 Gestion des produits	6
9.3 Gestion des transporteurs	7
9.4 Gestion des utilisateurs	7
Specifications techniques	8
Choix techniques et environnements de travail	8
I. Front-end	8
1. Technologies Front-end	8
II. Back-end	8
1. Technologies Back-end	8
2. Base de données	8
3. Arborescence des fichiers	9
III. Autres Outils	10
1. Git et GitHub	
2. IDE	
3. Communication	
Réalisation	11
I. Charte graphique	
II. Maquette	
III. Conception de la base de donnée	12
1. MCD	12
IV. Extraits de code significatifs	15
1. Authentification	15

1.1 Inscription	15
2. Panier	21
2.1 Ajout d'un article	21
2.2 mise à jour et suppression d'un ou plusieurs produits	23
3. Confirmation de commande	25
4. Autocomplétion en JS asynchrone	27
V. Veille sur les vulnérabilités de sécurité	29
1. Faille XSS	29
1.1 Protection contre les failles XSS	29
2. Faille CSRF	29
1. Prévention contre les failles CSRF	29
3. Eviter les mots de passe en clair :	30
4. Eviter les injections SQL	30
5. Contrôler les accès	30
6. Se tenir informé	31
VI. Recherche effectué sur un site anglophone	32
Axe d'améliorations	34
Remerciements	35
Annexes	36
I. Maquette	36
II. Les Breakpoints (media queries de TailwindCSS)	37
III. Modèle logique de données	37
IV. Modèle physique de données	38
V. Le fichier de création automatique de la base de données	39

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 a 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. À 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

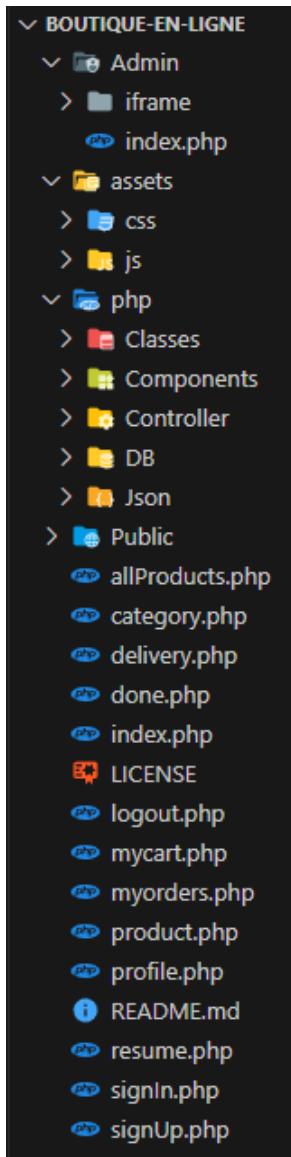
La base de données choisie est en langage SQL, et le Système de Base de Données (SGBD) est MySQL. Nous avons également utilisé phpMyAdmin, application Web de gestion pour les systèmes de gestion de base de données MySQL et MariaDB.

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 p. 36).



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 **MVC¹**, bien que largement utilisé, n'a pas été utilisé ici. Les classes PHP sont chargées via leur méthodes d'effectuer nos requêtes **SQL**, nous en reparlerons dans les extraits de codes significatifs.



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 servis 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

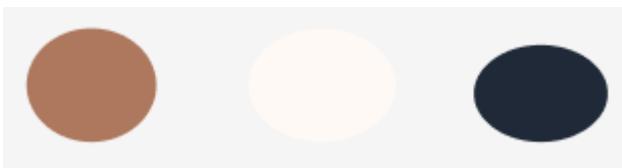
1. Logo

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;
```

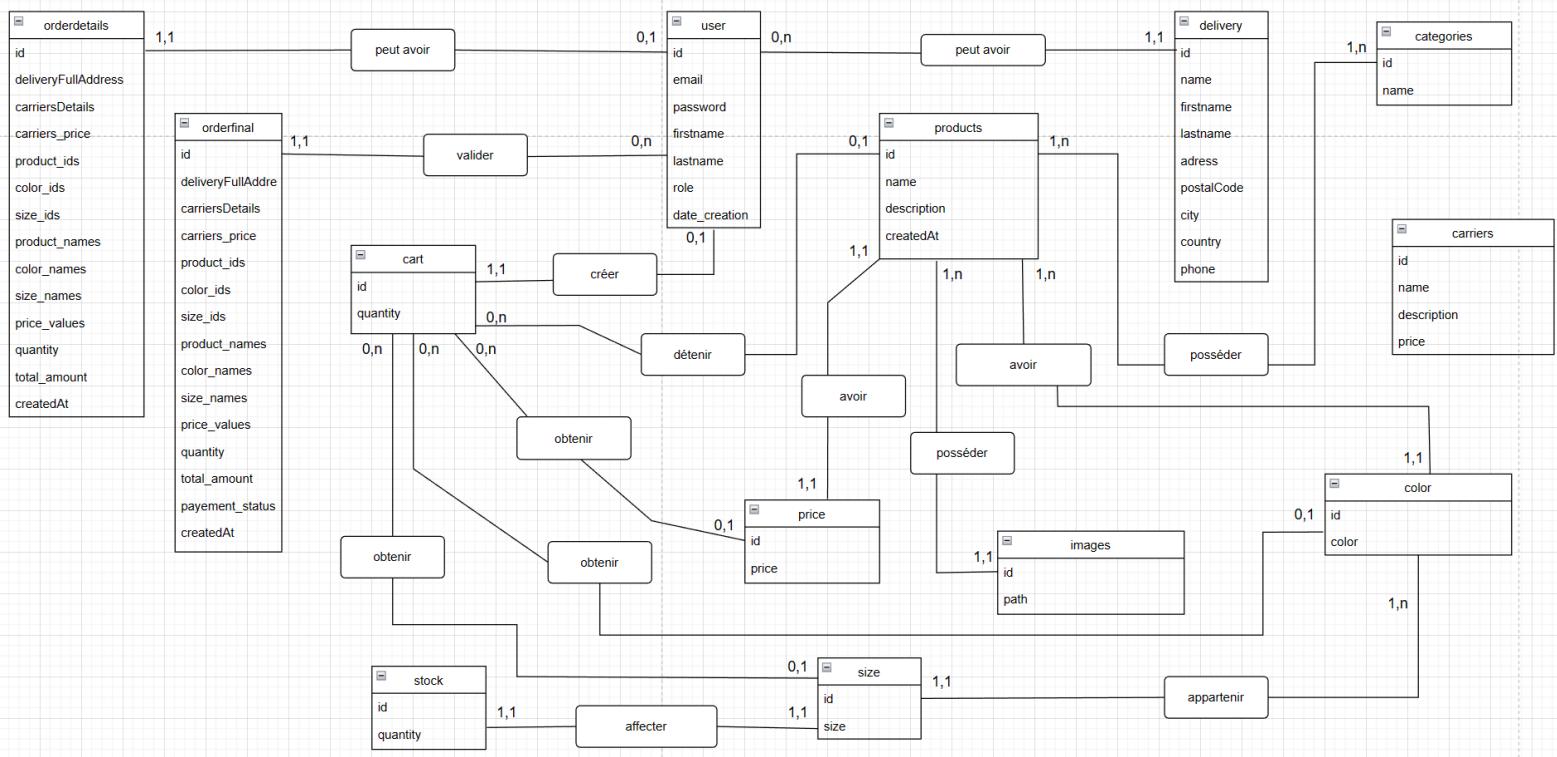
II. Maquette

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).

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 media queries(voir annexes p 34).

III. Conception de la base de donnée

1. Modèle Conceptuel de Données (MCD)



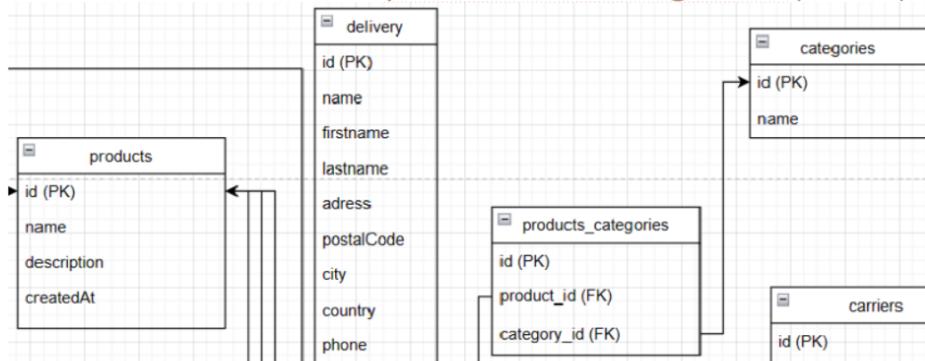
Comme illustré ci dessus, on peut voir que la base de données s'articule autour de 3 tables principales :

- 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)

table de liaison products_categories(MLD)

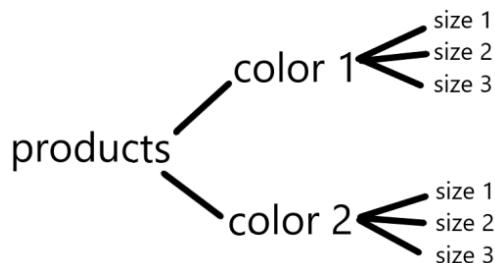


→ cart

```
'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;',

'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;'
```

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 'color' dont celle-ci a une option enfant 'size'. Cela permet d'avoir plusieurs tailles rattachées à une seule couleur et une 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 payment_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,
    `payment_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 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 34) le Modèle Logique de Données, (p 35) le Modèle Physique de Données et (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. (dont unique en base de données)

```
'user' => 'CREATE TABLE `user` (
    `id` INT AUTO_INCREMENT PRIMARY KEY,
    `email` varchar(255) NOT NULL UNIQUE,
```

The image displays two side-by-side screenshots of a web application's user registration page. Both screenshots have a header 'Créer un compte'.
Left Screenshot (Successful Submission):

- Votre prénom: [empty input]
- Votre nom: [empty input]
- Votre adresse e-mail: name@company.com
- Mot de passe: [empty input]
- Confirmer le mot de passe: [empty input]

A blue button at the bottom says 'Créer un compte'. Below it, a link says 'Vous avez déjà un compte ? Connectez-vous ici'.
Right Screenshot (Validation Error):

- Votre prénom: [empty input]
- Votre nom: [empty input]
- Votre adresse e-mail: test@aa.aa
- Mot de passe: [empty input]
- Confirmer le mot de passe: [empty input]

An error message 'adresse email déjà utilisée !' (Email address already in use!) is displayed below the 'Votre adresse e-mail' field. A blue button at the bottom says 'Créer un compte'. Below it, a link says 'Vous avez déjà un compte ? Connectez-vous ici'.

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 du visuel, des interactions, afin de transmettre 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. Elle possède 2 méthodes:
 - GET : pour récupérer des données depuis le serveur.
 - POST : pour passer des données (ajout/suppression/modification) au serveur.
- **CRUD (Create Read Update Delete)** : le terme CRUD résume les fonctions qu'un utilisateur a besoin d'utiliser pour créer et gérer des données.

CRUD-Operation	SQL	RESTful HTTP	XQuery
Create	INSERT	POST, PUT	insert
Read	SELECT	GET, HEAD	copy/modify/return
Update	UPDATE	PUT, PATCH	replace, rename
Delete	DELETE	DELETE	delete

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 détecté 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.

Créer un compte

Votre prénom	Votre nom
<input type="text"/>	<input type="text"/>
Votre adresse e-mail	
<input type="text" value="testeur@aa.aa"/>	
Mot de passe	
<input type="password"/>	
Confirmer le mot de passe	
<input type="password"/>	
<input style="background-color: #0072BD; color: white; border-radius: 5px; padding: 5px;" type="button" value="Créer un compte"/>	
Vous avez déjà un compte ? Connectez-vous ici	

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

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

La méthode “availableEmail(mon email)” de la classe User qui vérifie la présence de l'email dans la table “user” à l'aide la fonction **rowCount()** qui compte le nombre de lignes correspondantes dans cette table.

```
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();
}
```

Comme vu ci-dessus pour l'email, à l'aide de la fonction **passChecker(“mon input”)** pour la vérification du mot de passe, on applique la même méthode. Afin de **DRY** (Don't Repeat Yourself) le code, cette fonction a été créée.

```
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 et sécurités

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..

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

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

- Encapsulation par condition : est-ce que chaque input est rempli ? (Ils ont tous un attribut "required" en HTML)

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

```

<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:rounded-md" value="John" />
    </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:rounded-md" value="Doe" />
    </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:rounded-md" value="john.doe@example.com" />
  </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-300 text-gray-900 sm:rounded-md" value="password123" />
  </div>
  <div>
    <label for="confirm-password" class="block mb-2 text-sm font-medium text-gray-900 dark:text-white">Confirmer le mot de passe</label>
    <input type="password" name="confirm-password" id="confirm-password" required placeholder="*****" class="bg-gray-50 border border-gray-300 text-gray-900 sm:rounded-md" value="password123" />
  </div>
  <div><span id="errorMsg" class="text-rose-600"></span></div>
  <?php
  if (isset($_GET['error']) && !empty($_GET['message'])) {
    echo htmlspecialchars($_GET['message']);
  }
  ?>
  </div>
  <button type="submit" id="signUp" name="signUp" class="disabled:opacity-25 w-full text-white bg-blue-600 hover:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-blue-500 focus:ring-offset-2">
    Vous avez déjà un compte ? <a href="./signIn.php" class="font-medium text-primary-600 hover:underline dark:text-primary-500">Connexion</a>
  </button>
</form>

```

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

```

if ($_POST['password'] === $_POST['confirm-password']) {
  // si oui, on récupère toutes les informations des superglobales $_POST() (la valeur de chaque input au moment de la soumission du formulaire) en les transformant via htmlspecialchars().
  $firstname = htmlspecialchars($_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 :

```

// vérifications 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 :

```
// 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

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

Le code complet :

```
● ● ●

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 grâce au `$_GET` (superglobale) via notre redirection en cas d'erreurs.

```
<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 au panier

Pour la gestion du panier, 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.



Une fois que l'utilisateur appuie sur le bouton "Ajouter au panier", on envoie une requête **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.

```
size_id: Number(mySizeId),  
quantity: Number(myQty),
```

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 en appuyant sur le bouton plusieurs fois.

```
● ● ●  
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  
        })  
    }  
})
```

The code block shows the JavaScript logic for adding a product to the cart. It includes a `fetch` call to a PHP controller, setting the `Content-Type` header to `application/json`, and handling the response by setting the `inputQty.value` back to 1 and updating the `myQty` variable. Red arrows point from the highlighted code segments in the first snippet to the corresponding elements in the second snippet:

- A red arrow points from the `size_id: Number(mySizeId),` line to the `size_id: Number(mySizeId),` line in the code block.
- A red arrow points from the `quantity: Number(myQty),` line to the `quantity: Number(myQty),` line in the code block.
- A red arrow points from the `size_id: Number(mySizeId),` line to the `size_id: Number(mySizeId)` option in the `<select>` element.
- A red arrow points from the `quantity: Number(myQty),` line to the `type="number" value="1" max="10" id="quantity"` line in the code block.

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.

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

- On incrémente si le produit ajouté au panier est déjà présent dans ce dernier et correspond parfaitement (couleur et taille).

```
// 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. Une ligne dans la table "cart" par produit.

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

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

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

- On met à jour le panier.

ici le code complet:

```
● ● ●

$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 et suppression d'un ou 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

Chemise classique Blanc M 1 35.99 € 35.99 € supprimer

T-shirt casual Rouge M 1 17.99 € 17.99 € supprimer

prix hors livraison : 53.98 €

Valider mon panier

En cas de changement, le prix total est automatiquement recalculé et on le fait savoir à l'utilisateur. `arrayPrice` contient à chaque index (prix du produit * quantité), on utilise la fonction `.reduce()` pour additionner le tout.

```
// dès que la quantité est changé tous les calculs sont mis à jour
child4.addEventListener("change", () => {
    // on empêche une valeur négative
    if (child4.value < 0) child4.value = 0;

    // child4.value = child4.textContent
    totalPrice = item.price * child4.value;

    child6.textContent = totalPrice + "€";

    // on met à jour notre tableau des prix totaux et le visuel
    arrayPrice[key] = totalPrice;
    child8.textContent = `prix hors livraison : ${finalPrice()} €`;
});

const initialValue = 0; // on peut changer cette variable si un bon d'achat / code promo par exemple
const finalPrice = () =>
    arrayPrice
        .reduce(
            (accumulator, currentValue) => accumulator + currentValue,
            initialValue
        )
        .toFixed(2);
```

ici on récupère les valeurs de nos inputs une fois que le bouton “Valider mon panier” est cliqué. On envoie tout dans `jsonData` grâce à une boucle for in et on ajoute (`.push()`) à chaque index les informations concernant le produit.

```

// mis à jour du panier en cas de valeurs différentes
const select = document.querySelectorAll(".size");
const input = document.querySelectorAll(".quantity");

// gestion du formulaire en asynchrone
const updateValuesHandler = () => {
    // notre futur json envoyé dans une seule promesse, on boucle en back pour extraire les valeurs
    let jsonData = [];

    for (const index in data.cart) {
        const element = data.cart[index];

        const jsonValidation = {
            cart_id: element.id,
            product_id: element.productId,
            color_id: element.colorId,
            size_id: Number(select[index].value),
            quantity: Number(input[index].value),
            price_id: element.priceId,
            user_id: element.userId,
        };
        jsonData.push(jsonValidation);
    }
    // on envoie ensuite un json plein de données
    fetch(`${keyPath}validateMyCart.php`, {
        method: "POST",
        body: JSON.stringify(jsonData),
        headers: {
            "Content-Type": "application/json",
        }
    })
}

```

37

Côté 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

```

$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 validés. Nous récupérons toutes les données en dur, pour la table orderdetails.

Nous sommes parti du principe que :

- Un utilisateur ne peut avoir qu'un seul panier en cours (une ligne en table SQL par produit)
- Un utilisateur ne peut y avoir qu'une seule commande en cours (non payée).
- Un utilisateur peut avoir 0 ou N commandes payées.

APRÈS que la table orderfinal (copie de orderdetails avec la colonne payment_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. Nous envoyons l'objet JSON.stringify(), à l'intérieur nous aurons le payment_status.

```
// récupération de la commande en cours
fetch(`${keyPath}checkMyOrder.php`) // à 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 = { payment_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 détecté 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() (ajoute une ligne dans la table orderfinal) 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 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
session_start();

$userId = $_SESSION["userId"];

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

$payment_status      = $data['payment_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;

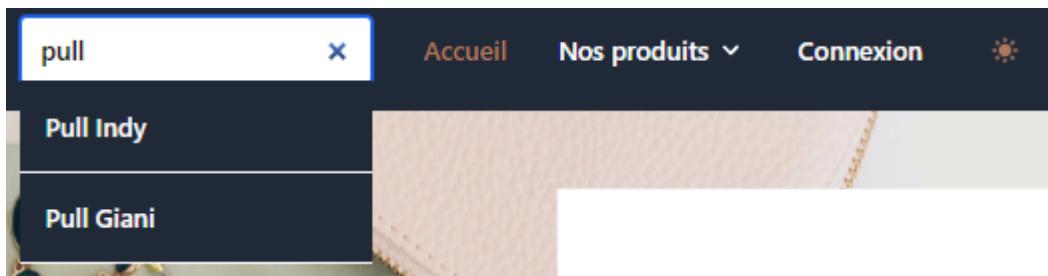
$myOrderfinal = new Orderfinal();
$myCart = new Cart();
$myOrderdetails = new Orderdetails();
$result = $myOrderdetails->getByUserId($userId);
// si la commande existe alors on peut continuer le tunnel d'achat
if ($result) {

    $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, $payment_status, $userId);
    // si la table orderfinal est remplie alors on vide les tables cart et orderdetails qui contiennent
    // cet id user
    if ($orderSuccess) {
        $myCart->deleteRowById($userId);
        $myOrderdetails->deleteRowById($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 écouteur sur l'événement keyup et un fetch post, qui ira chercher la réponse du serveur à une requête SQL LIKE "%input.value%"

- l'événement Javascript + fetch post

```
7 searchBar.addEventListener("keyup", () => {
8   results.innerHTML = "";
9   const jsonValue = { name: searchBar.value };
10
11  if (searchBar.value.length > 0) {
12    fetch(` ${keyPath}searchProducts.php`, {
13      method: "POST",
14
15      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

```
8 // Récupérer les données envoyées depuis la requête fetch POST
9 $data = json_decode(file_get_contents('php://input'), true);
10
11 $inputValue = $data['name'];
12
13 $myProducts = new Products();
14
15 $mySearch = $myProducts->getSearchLike($inputValue);
16
17 // Répondre avec une réponse JSON indiquant le succès ou l'échec de l'opération
18 $response = array($mySearch);
19 echo json_encode($response);
```

Une fois la réponse du serveur transmise et grâce aux chaînages 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.

```

1 import { addElement } from "../modules/addElement.js";
2 import { keyPath } from "../modules/key.js";
3
4 const searchBar = document.querySelector("#search");
5 const results = document.querySelector("#results");
6
7 searchBar.addEventListener("keyup", () => {
8   results.innerHTML = "";
9   const jsonValue = { name: searchBar.value };
10
11  if (searchBar.value.length > 0) {
12    fetch(`${keyPath}searchProducts.php`, {
13      method: "POST",
14
15      body: JSON.stringify(jsonValue),
16      headers: {
17        "Content-Type": "application/json",
18      },
19    })
20    .then((response) => {
21      if (response.ok) {
22        response.text().then((result) => {
23          const items = JSON.parse(result);
24          for (const key in items[0]) {
25            const nameProduct = items[0][key].name;
26            const idProduct = items[0][key].id;
27            console.log(nameProduct);
28
29            const link = addElement(
30              "a",
31              ["block", "bg-white", "p-4", "dark:bg-gray-800", "dark:text-white", "border-b", "border-white"],
32              { href: `./product.php?name=${nameProduct}&id=${idProduct}` },
33              `${nameProduct}`
34            );
35            results.append(link);
36          }
37        });
38      } else {
39        alert("problème détecté merci de contacter un administrateur.");
40      }
41    })
42    .catch(function (error) {
43      console.log(error);
44    });
45  }
46}

```

NB: “keypath” est juste un chemin d'accès, cela nous évitait de le taper à chaque fois.

```
export const keyPath = "./php/Json/";
```

“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 texte 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.

```
$email = htmlspecialchars($_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 passées 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.

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

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

```
public function passVerify($email, $pass) {
    file_exists('../DB/DBManager.php') ? require('../DB/DBManager.php') : require('../php/DB/DBManager.php');
    $bdd = $bdd->prepare("SELECT `password` FROM ".$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 préparées

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 :

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

    const TABLE_NAME = "";

    public function getAllById($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'a pas le rôle "admin".

```
<?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](http://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



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

261

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

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

et

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



The main difference are the parameters of these functions:

376

- [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 affiche un message à l'utilisateur 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 double 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 quelques uns peuvent être immédiatement pointés:

- 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 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 Plateforme_ 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

The image displays a grid of six screenshots illustrating the responsive design of the M.A.H. Fashion Store website across three devices: Mobile (320px), Tablet (768px), and Desktop (1280px). The website features a clean, modern aesthetic with a dark header and footer.

- Mobile - 320px:** Shows a grid of product thumbnails and user profile cards.
- Tablet - 768px:** Shows a larger grid of products and user profiles, with a detailed product view on the right.
- Desktop - 1280px:** Shows a detailed product view with a sidebar for filters and a color palette at the top.

II. Les Breakpoints (media queries de tailwindcss)

```
tailwind.config.js
```

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

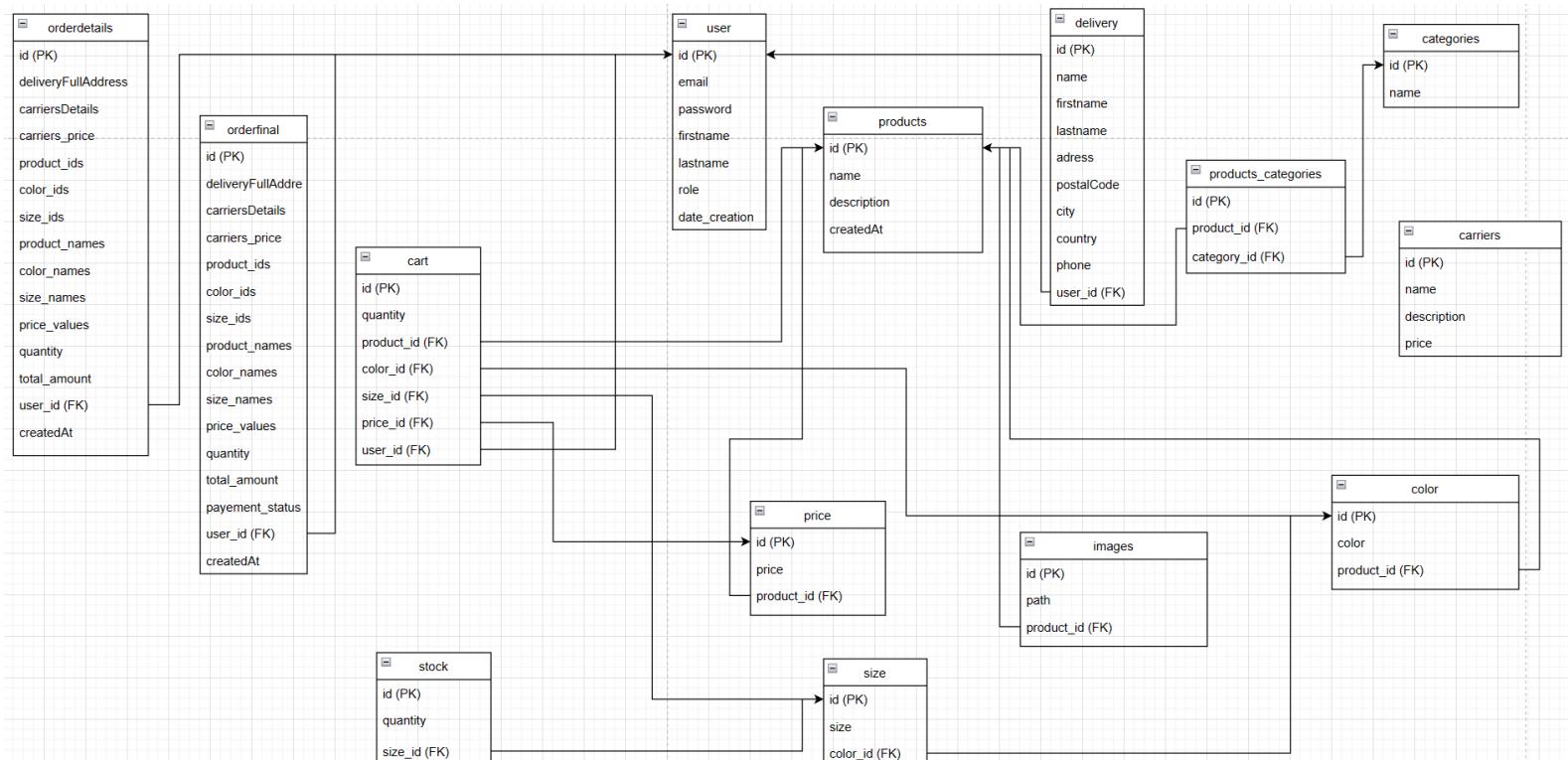
      'md': '768px',
      // => @media (min-width: 768px) { ... }

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

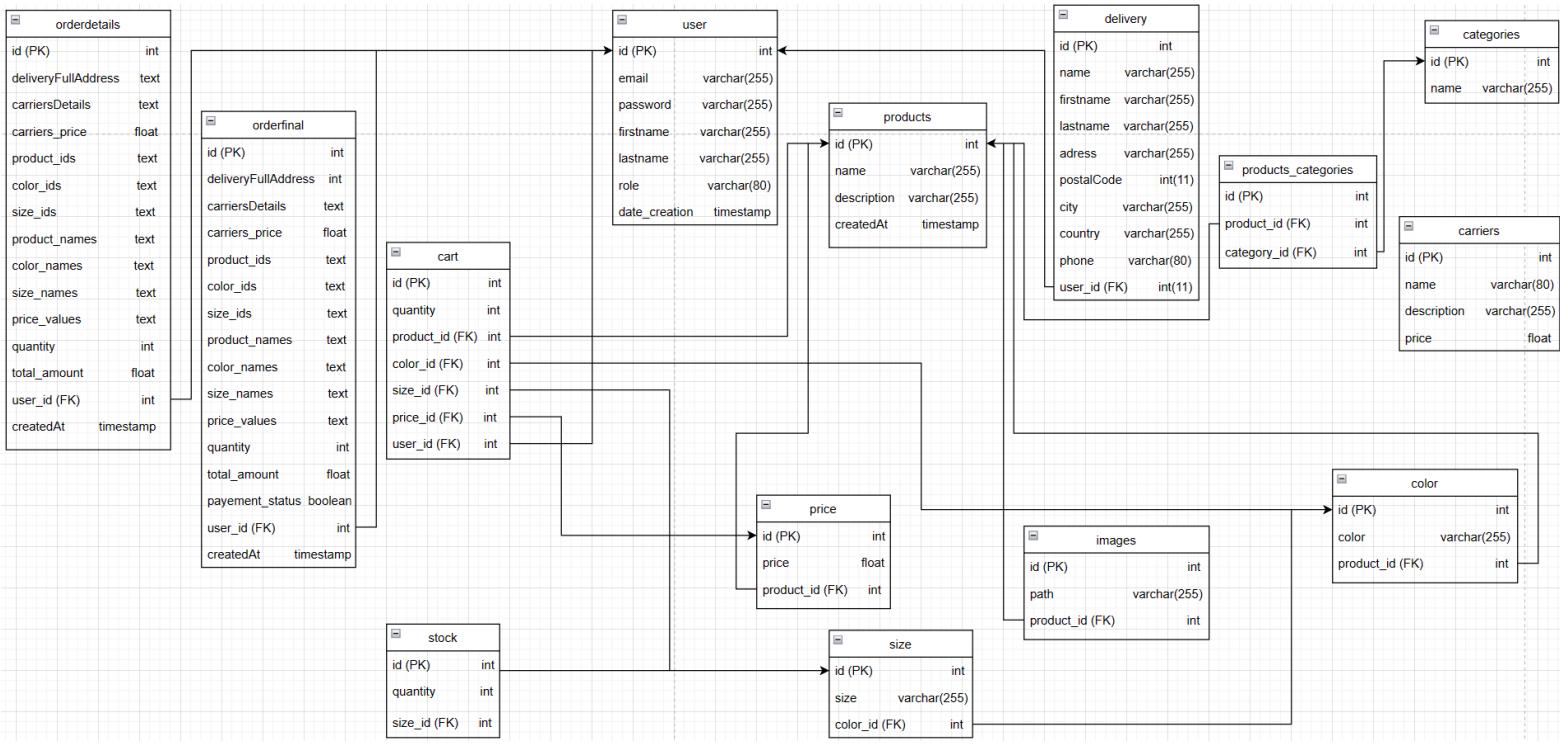
      'xl': '1280px',
      // => @media (min-width: 1280px) { ... }

      '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

```
1  <?php
2  require_once('php/DB/DBManager.php');
3
4  $tables = [
5      'user' => 'CREATE TABLE `user` (
6          `id` INT AUTO_INCREMENT PRIMARY KEY,
7          `email` varchar(255) NOT NULL UNIQUE,
8          `password` varchar(255) NOT NULL,
9          `firstname` varchar(255) NOT NULL,
10         `lastname` varchar(255) NOT NULL,
11         `role` varchar(80) NOT NULL,
12         `createdAt` timestamp NOT NULL DEFAULT current_timestamp()
13     ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',
14
15     'categories' => 'CREATE TABLE `categories` (
16             `id` INT AUTO_INCREMENT PRIMARY KEY,
17             `name` varchar(255) NOT NULL
18         ) ENGINE=InnoDB DEFAULT CHARSET=utf8;',
19
20     'products' => 'CREATE TABLE `products` (
21             `id` INT AUTO_INCREMENT PRIMARY KEY,
22             `name` varchar(255) NOT NULL,
23             `description` varchar(255) DEFAULT NULL,
24             `createdAt` timestamp NOT NULL DEFAULT current_timestamp()
25         ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',
26
27     'prod_cat' => 'CREATE TABLE `products_categories` (
28             `id` INT AUTO_INCREMENT PRIMARY KEY,
29             `product_id` INT NOT NULL,
30             `category_id` INT NULL,
31             FOREIGN KEY (`product_id`) REFERENCES `products`(`id`),
32             FOREIGN KEY (`category_id`) REFERENCES `categories`(`id`) ON DELETE SET NULL
33         ) ENGINE=InnoDB DEFAULT CHARSET=utf8;',
34
35     'image' => 'CREATE TABLE `images` (
36             `id` INT AUTO_INCREMENT PRIMARY KEY,
37             `path` VARCHAR(255) NOT NULL,
38             `product_id` INT NOT NULL,
39             FOREIGN KEY (`product_id`) REFERENCES `products`(`id`) ON DELETE CASCADE
40         ) ENGINE=InnoDB DEFAULT CHARSET=utf8;',
41
42     'price' => 'CREATE TABLE `price` (
43             `id` INT AUTO_INCREMENT PRIMARY KEY,
44             `price` FLOAT NOT NULL,
45             `product_id` INT NOT NULL,
46             FOREIGN KEY (`product_id`) REFERENCES `products`(`id`) ON DELETE CASCADE
47         ) ENGINE=InnoDB DEFAULT CHARSET=utf8;',
48
49     'color' => 'CREATE TABLE `color` (
50             `id` INT AUTO_INCREMENT PRIMARY KEY,
51             `color` varchar(255) NOT NULL,
52             `product_id` INT NOT NULL,
53             FOREIGN KEY (`product_id`) REFERENCES `products`(`id`) ON DELETE CASCADE
54         ) ENGINE=InnoDB DEFAULT CHARSET=utf8;',
55
56     'size' => 'CREATE TABLE `size` (
57             `id` INT AUTO_INCREMENT PRIMARY KEY,
58             `size` varchar(255) NOT NULL,
59             `color_id` INT NOT NULL,
60             FOREIGN KEY (`color_id`) REFERENCES `color`(`id`) ON DELETE CASCADE
61         ) ENGINE=InnoDB DEFAULT CHARSET=utf8;',
62
63     'stock' => 'CREATE TABLE `stock` (
64             `id` INT AUTO_INCREMENT PRIMARY KEY,
65             `quantity` INT NOT NULL,
66             `size_id` INT NOT NULL,
67             FOREIGN KEY (`size_id`) REFERENCES `size`(`id`) ON DELETE CASCADE
68         ) ENGINE=InnoDB DEFAULT CHARSET=utf8;',
69 ]
```

```

70     'carriers' => 'CREATE TABLE `carriers` (
71         `id` INT AUTO_INCREMENT PRIMARY KEY,
72         `name` varchar(80) NOT NULL,
73         `description` varchar(255) NOT NULL,
74         `price` float NOT NULL
75     ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',
76
77     'delivery' => 'CREATE TABLE `delivery` (
78         `id` INT AUTO_INCREMENT PRIMARY KEY,
79         `name` varchar(255) NOT NULL,
80         `firstname` varchar(255) NOT NULL,
81         `lastname` varchar(255) NOT NULL,
82         `address` varchar(255) NOT NULL,
83         `postalCode` int(11) NOT NULL,
84         `city` varchar(255) NOT NULL,
85         `country` varchar(255) NOT NULL,
86         `phone` varchar(80) NOT NULL,
87         `user_id` int(11) NOT NULL,
88         FOREIGN KEY (`user_id`) REFERENCES `user`(`id`)
89     ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',
90
91     'cart' => 'CREATE TABLE `cart` (
92         `id` INT AUTO_INCREMENT PRIMARY KEY,
93         `quantity` INT NOT NULL,
94         `product_id` INT NOT NULL,
95         `color_id` INT NOT NULL,
96         `size_id` INT NOT NULL,
97         `price_id` INT NOT NULL,
98         `user_id` INT NOT NULL,
99         FOREIGN KEY (`user_id`) REFERENCES `user`(`id`),
100        FOREIGN KEY (`price_id`) REFERENCES `price`(`id`),
101        FOREIGN KEY (`size_id`) REFERENCES `size`(`id`),
102        FOREIGN KEY (`product_id`) REFERENCES `products`(`id`),
103        FOREIGN KEY (`color_id`) REFERENCES `color`(`id`),
104        `createdAt` timestamp NOT NULL DEFAULT current_timestamp()
105     ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',
106
107    'orderDetails' => 'CREATE TABLE `orderdetails` (
108        `id` INT AUTO_INCREMENT PRIMARY KEY,
109        `deliveryFullAddress` TEXT NOT NULL,
110        `carriersDetails` TEXT NOT NULL,
111        `carriers_price` FLOAT NOT NULL,
112        `product_ids` TEXT NOT NULL,
113        `color_ids` TEXT NOT NULL,
114        `size_ids` TEXT NOT NULL,
115        `product_names` TEXT NOT NULL,
116        `color_names` TEXT NOT NULL,
117        `size_names` TEXT NOT NULL,
118        `price_values` TEXT NOT NULL,
119        `quantity` INT NOT NULL,
120        `total_amount` FLOAT NOT NULL,
121        `user_id` INT NOT NULL,
122        FOREIGN KEY (`user_id`) REFERENCES `user`(`id`),
123        `createdAt` timestamp NOT NULL DEFAULT current_timestamp()
124    ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;',
125

```

```
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     ];
147
```