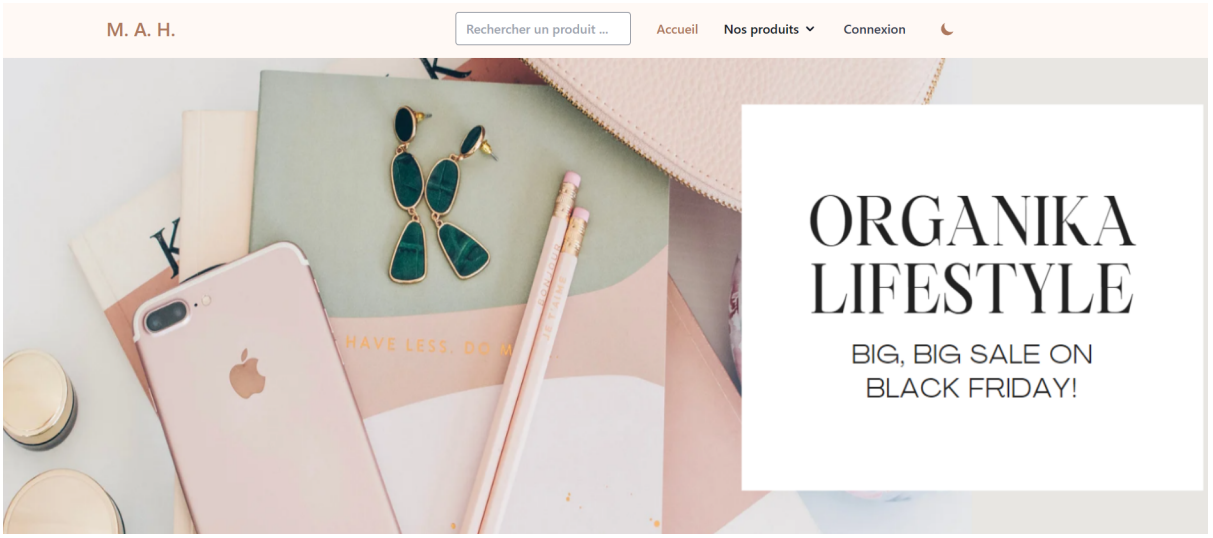
DOSSIER DE PROJET

TITRE PROFESSIONNEL DE DÉVELOPPEUR WEB & WEB MOBILE



Boutique en ligne

Miguel Molia

Table des matières

Remerciements

Je tiens à exprimer ma reconnaissance envers mes collègues de travail qui ont été des partenaires indispensables tout au long de ce projet. Leur collaboration, leur soutien et leur esprit d'équipe ont grandement contribué à la réussite de cette boutique en ligne. Travailler avec eux a été une expérience enrichissante.

Je tiens également à remercier mon école La Plateforme ainsi que mes formateurs, Ruben et Kevin, pour leur accompagnement et leur soutien tout au long de cette année de formation. Leurs conseils éclairés et leurs encouragements m'ont été d'une grande aide pour mener à bien ce projet.

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

Ce projet nous a été demandé par notre école LA PLATEFORME. Il consiste en la vente de vêtements et a été réalisé en groupe, composé de trois personnes: Aurélien GALLEA, Hugo CANOVAS et moi-même.

Pour sa réalisation, nous avons utilisé différentes technologies vues et apprises durant notre année de formation. Le front-end a été développé en utilisant HTML, CSS et JAVASCRIPT. Le back-end a été travaillé grâce à PHP et MySQL. Pour faciliter le développement du CSS, nous avons également eu recours aux frameworks TAILWINDCSS et FLOWBITE.

Pour l'organisation du projet, nous nous sommes servis d'une variété d'outils comme GIT pour le versionning, GITHUB pour l'hébergement, DISCORD pour le collaboratif, VISUAL STUDIO CODE pour l'environnement de développement intégré (IDE) et FIGMA pour s'occuper de la maquette.

Je me suis personnellement chargé de développer la partie catégorie de produit, avec le CRUD. Depuis le panel admin, l'administrateur peut ajouter/insérer, récupérer/afficher, mettre à jour/modifier ou supprimer un ou plusieurs produits, une ou plusieurs catégories. Côté utilisateur, il a fallu créer une barre de recherche, faire une page présentant tous les produits et catégories, un module d'inscription et de connexion, avec un espace dédié à la gestion du profil de l'utilisateur, une gestion pour confirmer son panier. Bien évidemment, une page d'accueil, pour un aspect global, et le responsive pour adapter à tous les écrans.

Ce travail m'a permis d'avoir une approche professionnelle, les contraintes de ce métier, le choix entre les différentes fonctionnalités, l'échange et le partage d'informations avec mes camarades, le travail d'équipe.

Spécifications fonctionnelles

- 1. Description du projet

Ce projet est une boutique en ligne, dont le concept repose sur une gamme variée de produits vestimentaires comprenant des t-shirts, des pulls, des shorts et des pantalons. Nous proposons également une sélection de couleurs et de tailles dans chaque catégorie de produits, afin de répondre aux besoins de chacun.

- 2. Périmètre du projet

Le site sera réalisé en français et ce dernier devra être accessible sur différents supports, à savoir mobile, tablette et ordinateur.

- 3. Cible adressée par le site internet

Le site s'adresse à un public assez large, accueillant hommes et femmes, et offrant une expérience agréable aussi bien par temps chaud que par temps froid.

- 4. Cahier des charges

Voici une liste non exhaustive des fonctionnalités minimales que notre boutique en ligne devait respecter:

- Page d'accueil
- Site responsive
- Barre de recherche / autocomplétion (en JS asynchrone)
- Accès boutique présentant tous les produits, catégories (en JS asynchrone)
- Page détail du produit
- Module de connexion / inscription, gestion profil utilisateur
- Espace administrateur avec gestion de produits, catégories
- Système de validation du panier

- 5. Arborescence du site

L’arborescence du site se présente ainsi :

- Page d’accueil
- Page inscription
- Page connexion
- Page tous les produits
- Page produit
- Page sélection de produit
- Page mon compte
- Page détail commande
- Page panier
- Page choix de la livraison
- Page paiement
- Page de confirmation de commande

Une partie back-office est également prévue afin de permettre la gestion du site.

- 6. Description des fonctionnalités

6.1 Authentification

L’authentification de l’utilisateur doit être effectuée via un formulaire pour l’inscription et la connexion.
Pour l’inscription, il doit fournir son email, mot de passe, prénom et nom.
Pour la connexion, l’email et le mot de passe suffisent.

6.2 Espace client

Dans l’espace client, l’utilisateur a la possibilité de modifier ses informations personnelles, comme par exemple son prénom ou bien son nom. Il doit confirmer son mot de passe pour effectuer ce changement, par mesure de sécurité. Il peut également consulter ses commandes passées, voir son panier ou tout simplement se déconnecter.

6.3 Catalogue produits, catégories et barre de recherche

Pour les produits, une page est dédiée à leur affichage, montrant toute la collection du site disponible. De plus, un système de filtrage par catégories est disponible si besoin, dans la barre de navigation. Enfin, une barre de recherche est à la disposition de l’utilisateur, permettant un résultat selon le nom du produit.

6.4 Page de détails du produit

Sur la page détaillée du produit, le client peut voir le nom du produit, les images correspondantes, le prix et une description. Il peut également choisir la couleur disponible, la taille, et la quantité voulue.

6.5 Panier client

Dans son panier, l’utilisateur peut voir tous les produits choisis ainsi que le montant à payer. Il peut à tout moment décider de modifier la quantité, la taille choisie, supprimer l’article du panier ou tout simplement valider ses achats.

6.6 Livraison

Pour la livraison, le client à le choix d’ajouter une nouvelle adresse ou tout simplement d’utiliser une adresse déjà enregistrée. Par la suite, il doit choisir le service de livraison qu’il souhaite.

6.7 Détail commande et paiement

Le détail de la commande montre à l'utilisateur un récapitulatif des articles choisis, du prix, de la quantité, du montant de la livraison et donc du prix total. Lorsque la commande est validée, une simulation de paiement se lance.

6.8 Commandes passées client

L'utilisateur a la possibilité d'accéder à son historique de commandes, depuis qu'il est inscrit sur le site, que ce soit les dernières commandes ou celles d'avant.

6.9 Back office

6.9.1 Gestion des catégories

Côté back, l'administrateur peut créer, afficher, mettre à jour ou supprimer une catégorie de produit, qui peut par la suite être utilisée lors de la création d'un produit.

6.9.2 Gestion des produits

L'administrateur a également la possibilité de créer, d'afficher, de mettre à jour ou bien de supprimer des produits. Pour la création de l'article, il doit remplir successivement le nom du produit, sa description, télécharger une image du produit, indiquer son prix, la couleur, la taille, la quantité, et pour finir la collection c'est-à-dire la catégorie correspondante à ce produit.

6.9.3 Gestion des utilisateurs

En ce qui concerne les utilisateurs, l'administrateur a accès à la liste complète de ceux inscrits sur le site. Il peut parfaitement supprimer un utilisateur s'il le souhaite.

6.9.4 Gestion du service de livraison

Pour le service de livraison, l'administrateur peut ajouter, mettre à jour ou supprimer celui qu'il souhaite.

Spécifications techniques

- 1. Choix techniques et environnements de travail

Technologies utilisées pour la partie front-end:

- HTML
- CSS
- Javascript
- Tailwindcss
- Flowbite

Technologies utilisées pour la partie back-end :

- PHP
- MySQL

L'environnement de développement est le suivant:

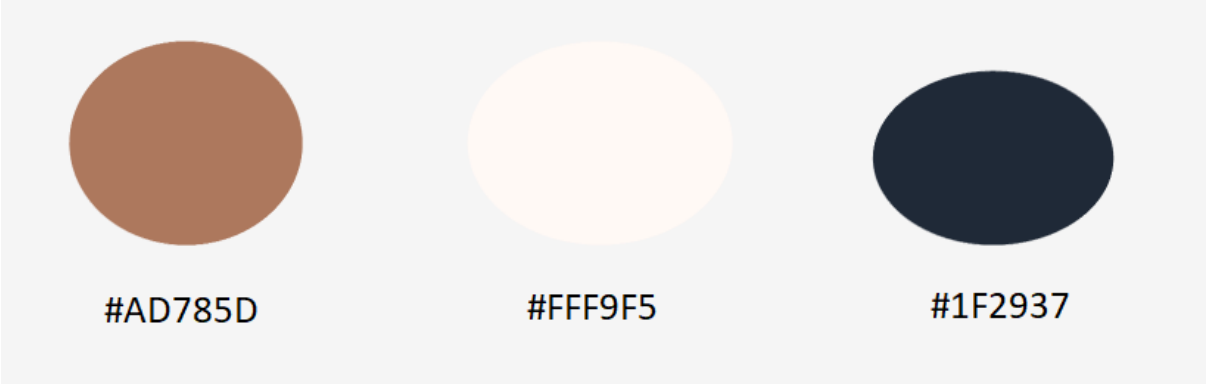
➢ Outil de versioning: GIT
➢ Hébergement: Github
➢ Collaboratif: Discord
➢ Editeur de code: Visual Studio Code
➢ Maquettage: Figma

- 2. Réalisation

2.1 Charte graphique

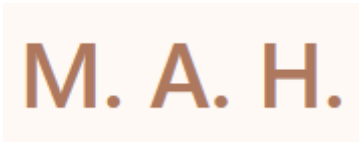
Police d’écriture: ui-sans-serif, system-ui, -apple-system;

Palette de couleurs:



Logo:

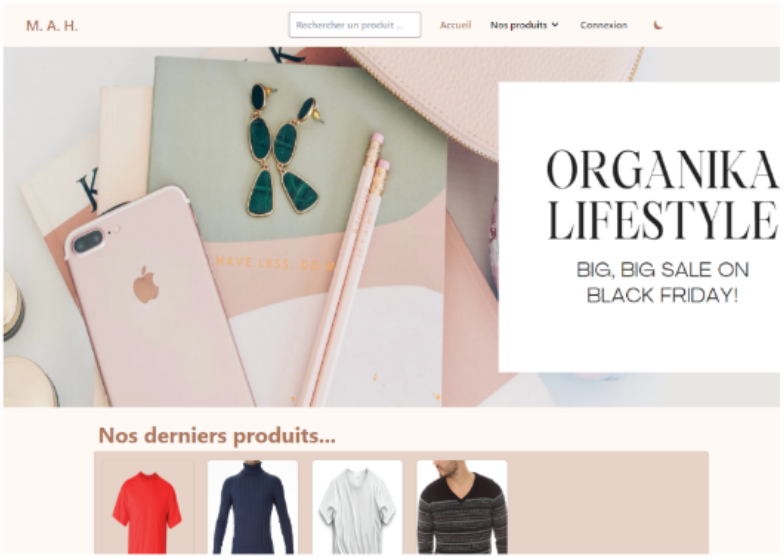
- Thème clair:

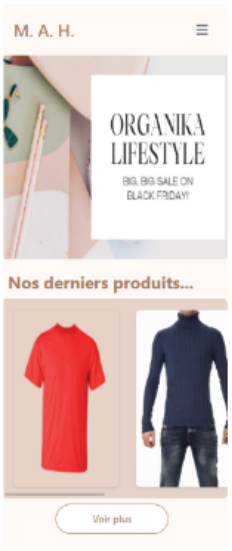


- Thème sombre:

2.2 Maquette

La maquette a été réalisée grâce à Figma, pour mettre en place le design selon la taille des écrans , les couleurs, le style.

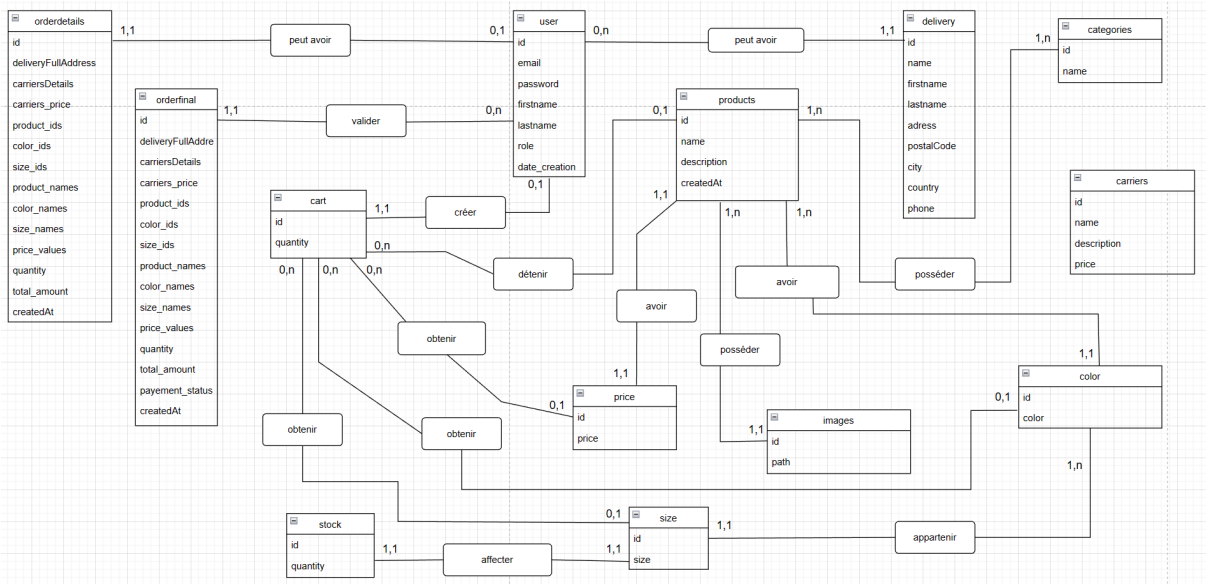




Vous trouverez la maquette dans les annexes du dossier (p.).

2.3 Conception de la base de données

2.3.1 Modèle conceptuel de données



Voici la base de données en MCD (modèle conceptuel de données). Vous trouverez également le MLD (modèle logique de données) et le MPD (modèle physique de données) dans les annexes du dossier (p.).

Comme illustré ci dessus, on peut voir que la base de données s'articule autour de 3 entités principales (entités qui deviennent des tables en MLD et MPD):

- L'entité user, correspondant à l'utilisateur (le client) ou l'administrateur, le rôle définit cela. Cette entité user est reliée à différentes entités dont l'entité delivery (la livraison), l'entité cart (le panier client), l'entité orderdetails (la commande en cours non payée) et l'entité orderfinal (la commande passée).

- l'entité products, correspondant au produit. Cette entité est reliée à l'entité price (le prix), l'entité images (les images), l'entité color (la couleur), l'entité categories (les catégories de produits, avec une table de liaison product_categories) et l'entité cart.

- l'entité cart, correspondant donc au panier client. Cette entité est quant à elle reliée à l'entité products, color, size, price et user.

Concernant l'entité products, elle est reliée à l'entité color, qui elle-même est reliée à l'entité size. Ce qui veut dire que le produit a une option parent color, et color a une option enfant size. Cela permet d'avoir plusieurs tailles pour une couleur, plusieurs couleurs pour un produit.

Pour ce qui est des entités orderdetails et orderfinal, les données y sont rentrées en brut. Nous avons fait cela pour livrer le client dans tous les cas, même si son compte est supprimé après avoir passé une commande.

2.4 Extraits de code significatifs

2.4.1 Authentification

Inscription:

Au niveau de l'inscription, l'utilisateur doit indiquer son prénom, son nom, son e-mail et son mot de passe ainsi qu'une confirmation de ce mot de passe. L'e-mail doit être unique, nous avons donc fait des vérifications en Javascript côté front-end.

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

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

    body: 'email=' + email.value,
    headers: {
      "Content-Type": "application/x-www-form-urlencoded",
    },
  })
```

Lorsque l'utilisateur relâche une touche du clavier dans l'input email, l'événement "keyup" est déclenché. Le code envoie une requête HTTP au serveur en utilisant la fonction fetch() avec la méthode POST. La requête contient l'adresse e-mail saisie par l'utilisateur dans le corps de la requête.

```php
$email = $_POST['email'];

$myUser = new User();

$usedEmail = $myUser->availableEmail($email);
```

Voici la page de traitement qui récupère la valeur de l'adresse e-mail à partir de la requête HTTP POST. On instancie un nouvel objet de la classe user pour accéder aux informations des utilisateurs du système. On appelle la méthode availableEmail sur l'objet user, en passant l'adresse e-mail récupérée en paramètre. La méthode availableEmail vérifie si l'adresse e-mail est déjà utilisée dans le système.

```php
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();
}
```

La méthode commence par inclure le fichier DBManager.php qui permet d'accéder à la base de données. Elle prépare une requête SQL pour sélectionner toutes les colonnes de la table user où l'adresse e-mail correspond à un marqueur de paramètre.
Ensuite, elle exécute la requête en passant l'adresse e-mail fournie en tant que paramètre. La méthode renvoie le nombre de lignes correspondant à l'adresse e-mail dans la base de données.

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

On vérifie si le résultat obtenu est supérieur à zéro. Si c'est le cas, cela signifie que l'adresse e-mail est déjà utilisée par un utilisateur existant dans la base de données. Si l'adresse e-mail est déjà utilisée, le code renvoie "used" comme résultat. Si l'adresse e-mail n'est pas utilisée, le code renvoie "available" comme résultat.

```
    .then((response) => {
      if (response.ok) {
        response.text().then(status => {


            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);
    });

});
```

Le serveur traite la requête et renvoie une réponse indiquant si l'adresse e-mail est déjà utilisée ou non. Le code traite la réponse de la requête :

- Si l'adresse e-mail est déjà utilisée ("status" est "used"), il affiche un message d'erreur et désactive le bouton du formulaire.

- Si l'adresse e-mail n'est pas utilisée ("status" n'est pas "used"), il efface le message d'erreur et active le bouton.

En cas d'échec de la requête (par exemple, une erreur réseau ou un problème du serveur), une alerte est affichée.

```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']);
```

Après l'étape où l'on vérifie si l'e-mail est unique et donc disponible, nous poursuivons l'inscription avec les autres champs. On vérifie si le formulaire d'inscription a été soumis, si les champs firstname, lastname, email, password et confirm-password ne sont pas vides (c'est-à-dire qu'ils ont été remplis par l'utilisateur).
Si tous les champs requis sont remplis, on vérifie ensuite si les champs password et confirm-password ont la même valeur. Cela permet de s'assurer que l'utilisateur a correctement confirmé son mot de passe.
Si le mot de passe est correctement confirmé, on utilise la fonction htmlspecialchars pour échapper les caractères spéciaux potentiellement dangereux (comme les balises HTML) dans les valeurs des champs firstname, lastname, email, password, et confirm-password. Cela améliore la sécurité de l'application en évitant les attaques d'injection de code.

```php
if (!Verify::verifySyntax($email)) {
    header('location:signUp.php?error=1&message=merci de rentrer un email valide !');
    exit();
}
$user = new User();
```

On vérifie ensuite la syntaxe de l'adresse e-mail saisie par l'utilisateur en utilisant la méthode statique verifySyntax de la classe Verify.
Si l'adresse e-mail n'est pas valide, l'utilisateur est redirigé vers la page signUp.php avec un message d'erreur indiquant qu'il doit saisir une adresse e-mail valide.
Sinon, on continue le traitement avec une instance d'un nouvel objet de la classe User.

```php
class Verify {

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

}
```

La méthode statique verifySyntax($email) effectue une validation de la syntaxe de l'adresse e-mail passée en paramètre ($email). Elle utilise la fonction filter_var() avec le filtre FILTER_VALIDATE_EMAIL pour vérifier si l'adresse e-mail est valide.
Si l'adresse e-mail est valide (c'est-à-dire qu'elle correspond à un format d'e-mail correct), la méthode retourne true. Sinon, si l'adresse e-mail est invalide ou ne correspond pas au format d'e-mail attendu, la méthode retourne false.

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

On appelle la méthode availableEmail de l'objet $user pour vérifier si l'adresse e-mail existe déjà dans la base de données. La méthode retourne le nombre de lignes correspondant à l'adresse e-mail dans la base de données.

Cette deuxième vérification de l'e-mail disponible ou non (en back-end cette fois) est effectuée lorsque l'utilisateur soumet le formulaire d'inscription. Elle agit comme une mesure de sécurité supplémentaire pour s'assurer que l'adresse e-mail n'est pas déjà utilisée au moment de l'inscription réelle.
La vérification JavaScript (front-end) faite précédemment est effectuée au fur et à mesure que l'utilisateur saisit l'adresse e-mail, offrant ainsi un retour instantané sans recharger la page. Cela améliore l'expérience utilisateur en évitant de soumettre le formulaire pour recevoir un message d'erreur.

L'utilisation des deux vérifications est complémentaire, car elles offrent une meilleure expérience utilisateur en fournissant un retour instantané et en évitant des requêtes inutiles au serveur lorsque l'utilisateur saisit l'adresse e-mail, tandis que la vérification PHP (back-end) garantit l'intégrité des données et protège contre les manipulations malveillantes.

On vérifie donc si le nombre de lignes renvoyées par la méthode availableEmail est différent de zéro. Si le nombre de lignes est différent de zéro, cela signifie que l'adresse e-mail est déjà utilisée par un utilisateur existant dans la base de données.
On redirige l'utilisateur vers la page de formulaire d'inscription signUp.php avec un message d'erreur sur la page de formulaire d'inscription pour informer l'utilisateur que l'adresse e-mail qu'il a saisie est déjà enregistrée dans le système.

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

On utilise password_hash pour transformer le mot de passe ($pass1) en une forme sécurisée et illisible appelée "hachage".
bcrypt en paramètre est utilisé pour saler son mot de passe, rendant le hachage plus résistant et complexe.

Connexion:

Concernant la connexion, l'utilisateur doit remplir un formulaire et le valider:

```php
if (isset($_POST['signIn'])) {
    if (!empty($_POST['email']) && !empty($_POST['password'])) {

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

On vérifie si le bouton de soumission du formulaire de connexion a été cliqué et que le formulaire a été soumis.
On vérifie que les champs d'e-mail et de mot de passe ne sont pas vides.
Si les deux conditions sont satisfaites, on utilise à nouveau la fonction htmlspecialchars.

```php
$user = new User();

$hachedPassword = $user->passVerify($email, $pass);
```

On instancie un nouvel objet $user de la classe User. On appelle la méthode passVerify de l'objet $user, en lui passant en paramètre l'e-mail et le mot de passe saisis par l'utilisateur.

```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 = ?");
    $request->execute([$email]);
    $row = $request->fetch();

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

On inclut le fichier DBManager.php qui est responsable de la gestion de la connexion à la base de données. On vérifie d'abord si le fichier DBManager.php existe dans le répertoire parent (../) et on l'inclut s'il est trouvé. Sinon, on inclut le fichier DBManager.php à partir du répertoire php/DB/ actuel.

On prépare une requête SQL pour sélectionner le mot de passe associé à l'e-mail donné. La requête est préparée en utilisant des marqueurs de paramètre, représentés par des points d'interrogation .
On exécute la requête SQL préparée en remplaçant le point d'interrogation par la valeur réelle de l'e-mail.

On récupère la première ligne de résultat de la requête exécutée dans une variable $row. Le résultat est récupéré sous forme de tableau associatif contenant les colonnes de la requête.
On utilise ensuite une expression ternaire, qui contient 3 éléments avec une condition. C'est une structure similaire à if-else, mais avec une syntaxe plus concise.

La condition $row && password_verify($pass, $row['password']) est évaluée en premier. Si cette condition est vraie (c'est-à-dire que $row n'est pas vide et que le mot de passe saisi ($pass) correspond au mot de passe haché dans la base de données), alors l'expression renverra $row['password'], c'est-à-dire le mot de passe haché associé à l'e-mail donné.
Si la condition est fausse (c'est-à-dire que $row est vide ou que le mot de passe ne correspond pas), alors l'expression renverra false.

```
if ($hachedPassword !== false) {
    $response = $user->connection($email, $hachedPassword);
```

On vérifie si la variable $hachedPassword n'est pas égale à false. Si cette condition est vraie (c'est-à-dire que le mot de passe haché est valide), alors le bloc de code à l'intérieur du if sera exécuté.
On appelle la méthode connection de l'objet $user. Les valeurs de l'e-mail ($email) et du mot de passe haché ($hachedPassword) sont passées en tant que paramètres à cette méthode.

```php
public function connection($email, $hachedPassword) {
    file_exists('../DB/DBManager.php') ? require('../DB/DBManager.php') : require('./php/DB/DBManager.php');
    $request = $bdd->prepare("SELECT * FROM ".$this::TABLE_NAME." WHERE email = ? AND password = ?");
    $request->execute([$email, $hachedPassword]);
    $response = $request->fetch();
    return $response;

}
```

On va récupérer toutes les autres informations de l'utilisateur. On prépare une requête SQL pour sélectionner tous les champs (*) d'une table de la table user, où l'e-mail et le mot de passe correspondent aux valeurs passées en paramètres. La requête est préparée en utilisant des marqueurs de paramètre, représentés par des points d'interrogation.
On exécute la requête SQL préparée en remplaçant les points d'interrogation par les valeurs des paramètres $email et $hashedPassword.
On récupère le résultat de la requête exécutée. La méthode fetch récupère la première ligne du résultat de la requête sous la forme d'un tableau associatif.
On renvoie le résultat de la requête.

```php
    $_SESSION["userId"] = $response["id"];
    $_SESSION["email"] = $response["email"];
    $_SESSION["firstname"] = $response["firstname"];
    $_SESSION["lastname"] = $response["lastname"];
    $_SESSION["role"] = $response["role"];

    header("location:./");
    exit();
} else {
    header('location:./signIn.php?error=1&message=Nom d\'utilisateur ou mot de passe incorrect. Veuillez réessayer.');
    exit();
}
```

On stocke toutes les informations récupérées de l'utilisateur dans des variables de session. L'utilisateur est redirigé vers la page d'accueil en utilisant header("location:./").
Si la vérification du mot de passe échoue (c'est-à-dire que l'utilisateur a fourni un nom d'utilisateur ou un mot de passe incorrect), l'utilisateur est redirigé vers la page de connexion (signIn.php) avec un message d'erreur.

2.4.2 Gestion du CRUD pour les catégories

le CRUD (acronyme de Create, Read, Update et Delete) représente les opérations de base que l'on peut effectuer sur les données, il est largement utilisé dans le développement d'applications pour créer, lire, mettre à jour et supprimer des informations dans une base de données ou une API.

- Créer une catégorie

```
<form class="flex flex-col w-11/12 md:w-5/12 h-full space-y-5" action="../../php/Controller/verifNewCategory.php" method="POST">
    <div class="flex flex-col shadow-md bg-white rounded-lg px-8 py-4 space-y-3 dark:bg-gray-800 dark:border">
        <h2 class="text-md font-medium dark:text-white">Collection</h2>
        <div>
            <label for="name" class="block mb-2 text-sm font-normal text-gray-900 dark:text-white">Nom</label>
            <input type="text" id="name" name="name" class="block w-full p-2 text-sm text-gray-900 border border-gray-300 rounded-
        </div>
    </div>
    <button name="valider" type="submit" class="text-white bg-blue-700 mt-5 hover:bg-blue-800 focus:ring-4 focus:outline-none focu
</form>
```

La balise <form> est la principale pour définir un formulaire HTML. L'attribut action spécifie l'URL ou le chemin vers lequel les données du formulaire seront envoyées pour le traitement. L'attribut method spécifie la méthode HTTP utilisée pour envoyer les données, qui est ici "POST". Cela signifie que les données seront envoyées dans le corps de la requête HTTP.

La balise <input> est un champ de saisie de type text qui permet à l'administrateur de saisir le nom de la nouvelle catégorie. L'attribut id est utilisé pour lier cet élément d'entrée à l'étiquette associée (for="name"). Le nom de cet élément d'entrée est "name", ce qui signifie que sa valeur saisie sera accessible côté serveur avec la clé "name".

La balise <button> est un bouton de type "submit" qui est utilisé pour soumettre le formulaire. Lorsque l'administrateur clique sur ce bouton, les données du formulaire seront envoyées à l'URL spécifiée dans l'attribut action. Le nom de ce bouton est name="valider", ce qui signifie que sa valeur sera utilisée pour identifier ce bouton côté serveur lorsque le formulaire est soumis.

```php
require_once('../Classes/Categories.php');

use Classes\Categories;



    if(empty($_POST['name'])) {
        echo "Il faut rentrer un nom de catégorie";
    }else{
        $newCategory = new Categories();
        $newCategory->add($_POST['name']);
        header('location: '.$destination.'/admin/iframe/allCategories.php');
    }
```

Avec require_once('../Classes/Categories.php'), on inclut le fichier Categories.php, qui contient la classe Categories.

Le require_once assure que le fichier n'est inclus qu'une seule fois pour éviter les doublons.
Avec use Classes\Categories, on utilise la déclaration use pour importer le namespace Classes\Categories. Cela permet d'utiliser directement Categories plutôt que de devoir écrire le nom complet de la classe à chaque fois qu'on l'utilise.

On vérifie si le champ "name" du formulaire est vide (pas de nom de catégorie saisi).
Si le champ "name" est vide, on affiche le message d'erreur "Il faut rentrer un nom de catégorie".
Sinon (si le champ "name" est rempli) :
On crée une nouvelle instance de la classe Categories en utilisant $newCategory = new Categories().
On appelle la méthode add de la classe Categories et on lui passe le nom de la nouvelle catégorie ($_POST['name']) en paramètre.

Après avoir ajouté la nouvelle catégorie, on redirige l'utilisateur vers la page allCategories.php en utilisant header('location: '.$destination.'/admin/iframe/allCategories.php'). La variable $destination contient l'URL de base du site.

```php
public function add($name) {
    require('../DB/DBManager.php');
    $request = $bdd->prepare("INSERT INTO ".$this::TABLE_NAME." (name) VALUES (?)");
    $request->execute([$name]);

}
```

La méthode add prend un paramètre $name, qui représente le nom de la nouvelle catégorie à ajouter.
Elle inclut le fichier DBManager.php, qui contient la connexion à la base de données. Ensuite, elle prépare une requête SQL pour insérer une nouvelle ligne dans la table correspondante ($this::TABLE_NAME) avec la valeur du nom de la catégorie.
La requête est exécutée en passant le nom de la catégorie en tant que paramètre.
Ainsi, une nouvelle catégorie est ajoutée à la base de données avec le nom fourni.

- Afficher une (ou plusieurs) catégorie(s)

```php
<?php
session_start();

require_once('../Classes/Categories.php');

use Classes\Categories;

$cat = new Categories();
$categories = $cat->getAll();

echo json_encode($categories);
```

Même principe que précédemment, on inclut avec require_once('../Classes/Categories.php') le fichier Categories.php qui contient la classe Categories.

On utilise le namespace Classes\Categories pour pouvoir créer une instance de la classe Categories sans avoir à écrire le nom complet de la classe à chaque fois. On crée une nouvelle instance de la classe Categories. On appelle la méthode getAll de la classe Categories.

Une fois que toutes les catégories sont récupérées, elles sont converties en format JSON à l'aide de la fonction json_encode. Le résultat JSON est ensuite affiché en sortie avec la fonction echo.

```php
public function getAll() {
    require('../DB/DBManager.php');
    $request = $bdd->prepare("SELECT * FROM ".$this::TABLE_NAME);
    $request->execute();
    $response = $request->fetchAll(PDO::FETCH_ASSOC);
    return $response;

}
```

On prépare une requête SQL pour sélectionner toutes les lignes de la table correspondante, sans condition particulière. La requête est exécutée, récupérant ainsi toutes les catégories de la base de données. Les résultats sont récupérés sous forme d'un tableau associatif. Chaque ligne du tableau représente une catégorie avec ses colonnes en tant que clés et les valeurs des colonnes en tant que valeurs associées.
On nous renvoie le tableau contenant toutes les catégories récupérées depuis la base de données.

```javascript
fetch('../../php/Json/allCategories.php')
    .then(response => response.json())
    .then(data => {

        let categories = data.categories;
```

On envoie une requête HTTP GET au fichier allCategories.php.
On traite la réponse de la requête.

La fonction .then est une méthode des promesses JavaScript qui permet d'exécuter une action une fois que la promesse (la requête) est résolue (lorsque la réponse est reçue). Ici, elle utilise la méthode .json pour extraire le corps de la réponse sous forme de données JSON.
Une fois les données JSON extraites de la réponse, elles sont stockées dans la variable data. On cible ensuite les categories des données JSON (data) et on les stocke dans la variable categories. Cela nous servira pour les afficher sur la page.

- Mettre à jour une catégorie

```php
<?php
    session_start();

    $_SESSION['catID'] = $_GET['id'];

?>

    <main class="sm:pl-64 pt-[64px] min-h-screen bg-[#FFF9F5] dark:bg-gray-700">
        <div class="flex pt-4 justify-center w-full pb-5">
            <form class="flex flex-col w-11/12 md:w-5/12 h-full space-y-5" action="../../php/Controller/updateCategory.php" method="POST">
                <div class="flex flex-col shadow-md bg-white rounded-lg px-8 py-4 space-y-3 dark:bg-gray-800 dark:border">
                    <h2 class="text-md font-medium dark:text-white">Collection</h2>

                </div>
                <button name="valider" type="submit" class="text-white bg-blue-700 mt-5 hover:bg-blue-800 focus:ring-4 focus:outline-none
            </form>
```

On démarre une session PHP avec session_start, permettant de stocker des variables de session pour une utilisation ultérieure.
On stocke la valeur de $_GET['id'] dans la variable de session $_SESSION['catID']. Cela signifie que la valeur de $_GET['id'] est maintenant accessible dans d'autres pages ou scripts à travers $_SESSION['catID'].
Le formulaire affiché contient un bouton et est destiné à être utilisé pour mettre à jour une catégorie.

L'attribut action spécifie l'URL de l'endroit où les données du formulaire seront envoyées lorsque l'utilisateur cliquera sur le bouton Dans ce cas, le formulaire enverra les données à updateCategory.php
La méthode POST est utilisée, ce qui signifie que les données du formulaire seront envoyées dans le corps de la requête HTTP plutôt que dans l'URL.

```php
<?php
session_start();

require_once('../Classes/Categories.php');

use Classes\Categories;


    if (empty($_POST['name'])) {
        echo 'Vous devez rentrer une catégorie';
    } else {
        $myCat = new Categories();
        $myCat->updateName($_POST['name'], $_SESSION['catID']);
        header('location: ' . $destination . '/admin/iframe/allCategories.php');
    }
```

Toujours pareil, on vérifie si le champ du formulaire avec le nom name est vide à l'aide de empty($_POST['name']). Si le champ est vide, il affiche le message "Vous devez rentrer une catégorie". Si le champ n'est pas vide, cela signifie que l'utilisateur a saisi une nouvelle valeur pour le nom de la catégorie à mettre à jour. Dans ce cas, on crée un nouvel objet de la classe Categories. On appelle la méthode updateName de la classe Categories avec les paramètres $_POST['name'] (la nouvelle valeur du nom de la catégorie) et $_SESSION['catID'] (l'identifiant de la catégorie à mettre à jour).
Après la mise à jour, le code redirige l'utilisateur vers une autre page (allCategories.php) à l'aide du header('location: ' . $destination . '/admin/iframe/allCategories.php').

```php
public function updateName($name,$id) {
    require('../DB/DBManager.php');
    $request = $bdd->prepare("UPDATE ".$this::TABLE_NAME." SET name = ?  WHERE id = ? ");
    $request->execute([$name, $id]);
```

La fonction updateName inclut le fichier DBManager.php, qui contient la connexion à la base de données.
Ensuite, on prépare une requête SQL pour mettre à jour le champ name dans la table correspondante ($this::TABLE_NAME) en utilisant la valeur $name, où l'identifiant de la catégorie est égal à la valeur $id.
La requête est exécutée mettant ainsi à jour le nom de la catégorie dans la base de données.

- Supprimer une catégorie

```
buttonAgree.addEventListener('click', function () {
    fetch('../../php/Controller/deleteCategory.php', {
        method: "POST",
        body: JSON.stringify({
            category_id: category.id

        }),
        headers: {
            'Content-Type': "application/json",
        },
```

On ajoute un événement de clic sur le bouton (buttonAgree). Lorsque le bouton est cliqué, il effectue une requête HTTP vers le fichier deleteCategory.php à l'aide de la méthode fetch().
Cette requête est de type POST. Les données à envoyer avec la requête sont spécifiées dans l'option body. Dans ce cas, les données sont envoyées sous forme de JSON en utilisant JSON.stringify(). L'objet envoyé contient une propriété category_id dont la valeur est l'identifiant d'une catégorie (category.id).

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

$category_id = $data['category_id'];

require_once('../Classes/Categories.php');

use Classes\Categories;


$myCat = new Categories();

$deleteCat = $myCat->deleteRow($category_id);

header('location: '.$destination.'/admin/iframe/allCategories.php');
```

On commence par récupérer les données JSON envoyées avec la requête à l'aide de file_get_contents('php://input'). Ensuite, on décode ces données JSON en utilisant json_decode pour les convertir en un tableau PHP associatif, stocké dans la variable $data.

On extrait l'identifiant de catégorie (category_id) du tableau $data, puis on inclut le fichier Categories.php et on crée un objet de la classe Categories.

Ensuite, on appelle la méthode deleteRow de la classe Categories avec l'identifiant de catégorie ($category_id) comme argument pour supprimer la catégorie correspondante de la base de données.

Enfin, on redirige l'utilisateur vers une autre page (allCategories.php) à l'aide de header('location: '.$destination.'/admin/iframe/allCategories.php').

```php
public function deleteRow($id) {
    require('../DB/DBManager.php');
    $request = $bdd->prepare("DELETE FROM ".$this::TABLE_NAME." WHERE id = ? ");
    $request->execute([$id]);
```

On inclut le fichier DBManager.php, qui contient la connexion à la base de données.
On prépare une requête SQL de type DELETE pour supprimer la ligne de la table correspondante ($this::TABLE_NAME) où l'identifiant (id) est égal à la valeur $id.
La requête est exécutée à l'aide de $request->execute([$id]);, supprimant ainsi la ligne de la base de données qui correspond à l'identifiant donné.

2.5 Veille sur les vulnérabilités de sécurité

2.5.1 Exposition des données sensibles

Lorsque l’on va sur Internet, le navigateur utilise le protocole HTTP (Hypertext Transfer Protocol) pour afficher les pages web.
Les données transitant en HTTP peuvent être interceptées, car elles transitent en clair.

Comment éviter d’exposer les données sensibles en transit ?

- Utiliser le HTTPS pour l'ensemble de votre site, même s'il ne contient pas de données sensibles. C’est la manière sécurisée, Il utilise un protocole de chiffrement pour sécuriser les données échangées entre le navigateur et le serveur.

- Utiliser les requêtes GET pour récupérer les informations et POST pour modifier les informations.

2.5.2 Injections SQL

Cette vulnérabilité permet à un attaquant d’injecter des données non maîtrisées qui seront exécutées par l’application et qui permettent d’effectuer des actions qui ne sont normalement pas autorisées.

Solution:

Utilisation de requêtes préparées : Les requêtes préparées sont des instructions SQL paramétrées qui séparent clairement les données des instructions SQL. Les valeurs des paramètres sont traitées séparément de la requête SQL, ce qui empêche les attaquants d'injecter du code malveillant.

Exemple:

```php
public function add($name) {
    require('../DB/DBManager.php');
    $request = $bdd->prepare("INSERT INTO ".$this::TABLE_NAME." (name) VALUES (?)");
    $request->execute([$name]);

}
```

2.5.3 Failles XSS

Les failles XSS (Cross-Site Scripting) permettent à des attaquants d'injecter du code malveillant (généralement du code JavaScript) dans les pages web consultées par les utilisateurs. Lorsqu'un utilisateur consulte une page vulnérable, le code injecté s'exécute dans le navigateur de l'utilisateur, ce qui peut conduire à des attaques telles que le vol d'informations sensibles ou la modification du contenu de la page.

Solution:

Le htmlspecialchars est une fonction qui permet d'échapper les caractères spéciaux utilisés en HTML, tels que les chevrons ("<" et ">"), les guillemets doubles ("), les guillemets simples ('), etc. En échappant ces caractères, la fonction convertit ces caractères spéciaux en entités HTML, ce qui empêche leur interprétation en tant que code JavaScript malveillant par le navigateur.

Exemple:

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

2.5.4 Authentification faible

L'authentification faible, également appelée authentification légère ou faible sécurité d'authentification, est une méthode d'identification des utilisateurs qui offre un niveau de sécurité relativement bas. Elle se caractérise par des mécanismes d'authentification peu robustes et faciles à contourner, ce qui la rend vulnérable aux attaques et aux compromissions de sécurité.

Solution:

La fonction password_hash est utilisée pour hacher les mots de passe avant de les stocker dans la base de données. Elle utilise un algorithme de hachage fort (tel que bcrypt) pour transformer le mot de passe en une chaîne de caractères illisible et irréversible. Le hachage des mots de passe empêche leur stockage en clair dans la base de données, protégeant ainsi les utilisateurs contre les fuites de données en cas d'atteinte à la sécurité.

La fonction password_verify est utilisée pour vérifier si un mot de passe saisi par l'utilisateur correspond au hash stocké dans la base de données. Lorsqu'un utilisateur tente de se connecter, le mot de passe saisi est haché et comparé avec le hash stocké dans la base de données. Si les deux correspondent, l'authentification est réussie, sinon l'accès est refusé.

Exemple:

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

```php
password_verify($pass, $row['password'])
```

2.6 Recherche effectuée à partir d'un site anglophone

Lors du développement de la boutique en ligne, j'ai décidé d'utiliser Tailwind CSS pour rendre le site responsive et adaptatif sur différentes tailles d'écran. Cependant, j'ai rencontré des difficultés pour passer d'un format à l'autre de manière fluide. Pour résoudre ce problème, j'ai entrepris des recherches dans la documentation officielle de Tailwind CSS, qui était en anglais. J'ai trouvé les informations nécessaires concernant les points d'arrêt (breakpoints) appropriés pour assurer la réactivité du site sur tous les types d'appareils.

There are five breakpoints by default, inspired by common device resolutions:

Breakpoint prefix	Minimum width	CSS
`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) { ... }`

Après avoir donc sélectionné le responsive design sur le site, on m'indique ici qu'il existe 5 points d'arrêt par défaut, correspondant aux résolutions d'écran les plus couramment utilisées.

To add a utility but only have it take effect at a certain breakpoint, all you need to do is prefix the utility with the breakpoint name, followed by the character:`:`

```
<!-- Width of 16 by default, 32 on medium screens, and 48 on large screens -->
<img class="w-16 md:w-32 lg:w-48" src="...">
```

Ici, on nous dit que pour ajouter un utilitaire mais qu'il ne prenne effet qu'à un certain point d'arrêt, tout ce que nous avons besoin de faire c'est de préfixer l'utilitaire avec le nom du point d'arrêt, suivi par le caractère ":".

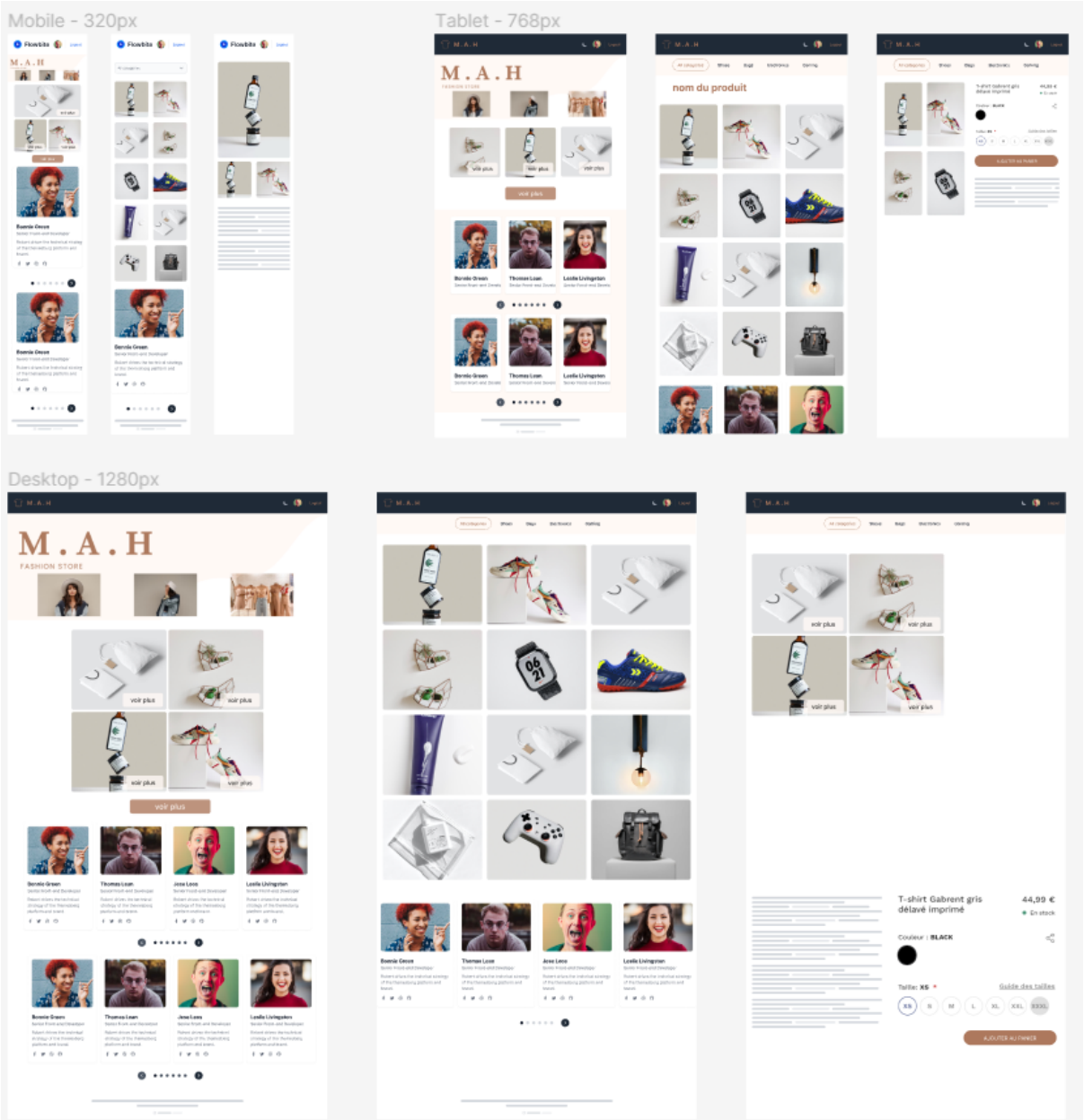
Ensuite, nous avons un exemple pour une largeur de 16 par défaut, de 32 pour les écrans de taille moyenne et 48 pour les écrans larges.

```
<form class="flex flex-col w-11/12 md:w-5/12
```

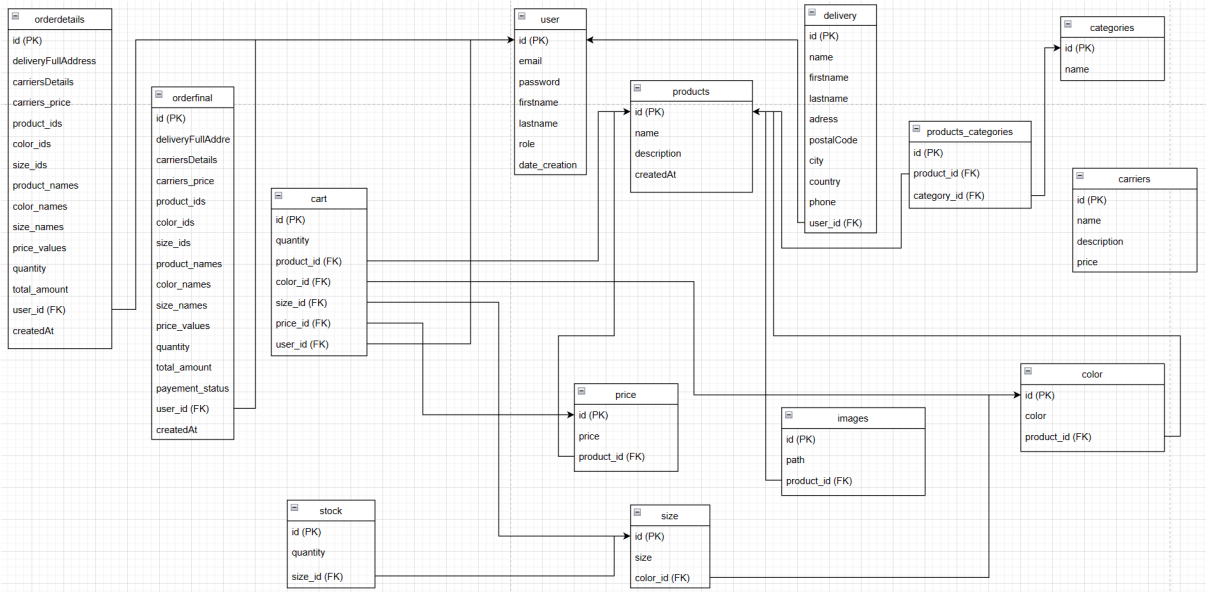
Voilà comment j'ai pu adapter la largeur en fonction des breakpoints vus sur Tailwind CSS, en fonction des différentes tailles d'écran.

Annexes

- 1. Maquette



- 2. Modèle logique de données



- 3. Modèle Physique de données