Titre RNCP :

développeur web & web mobile.

Dossier de Projet Pro :

Création d'une boutique en ligne

Machin Max

2021 / 2022

La Plateforme_

Table des matières

Liste des compétences du référentiel couvertes par le projet

Les compétences du référentiel couvertes par ce projet sont les suivantes :

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 statique et adaptable
- ➢ Réaliser une interface utilisateur web dynamique
- ➢ Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce

<u>Activité 2.</u> ***“Développer la partie back-end d’une application web et 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 & web mobile
- ➢ Élaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce.

Résumé du projet

Ce projet **“Boutique en ligne”** a été développé en groupe avec **Laura Savickaite & Laura Sconamiglio** dans le cadre de ma première année de formation à la Plateforme.

La finalité du projet était de réaliser une boutique en ligne comprenant au minimum les éléments suivants :

- ✓ Page d’accueil attractive
- ✓ Mise en avant des produits phares / derniers produits mis en ligne
- ✓ Chaque produit doit avoir une page complète générée dynamiquement (nom, image, prix, description, ajouter au panier...)
- ✓ Création de comptes d’utilisateurs
- ✓ Gestion du profil utilisateur (informations, historique d’achat, consultation du panier...)
- ✓ Gestion des produits à l’aide de back office pour les admins
- ✓ Gestion des catégories et des sous catégories de produits
- ✓ Barre de recherche de produits
- ✓ Validation du panier (simulation du paiement)
- ✓ Design contemporain et respectant la charte graphique de votre entreprise

Le projet a été réalisé en autonomie en **groupe.**

Il a repris une grande partie des étapes de conceptualisation d’un vrai projet professionnel telles que :

- une arborescence
- le maquettage graphique
- les modèles de données
- une organisation / répartition du travail à l’aide de **Trello**.

Nous avons décidé d'optimiser son accessibilité sur les différents supports en adoptant une approche design : **Mobile First**.

"L'entreprise" étant fictive, il a donc fallu *créer l'identité visuelle* de cette dernière et la développer. Nous avons donc créé **Everglow**, une entreprise de beauté & soins ciblant les femmes de tout âge, ayant tous types de peaux.

Nous avons décidé de développer du **back-end / front-end chacun** afin d'acquérir un maximum de compétences et connaissances requises à l'obtention du titre.

Sur le site, les utilisateurs ont la possibilité de **s'inscrire / se connecter**, à l'aide de leur *adresse mail*, d**'ajouter des produits au panier** puis **passer une commande**. Une fois inscrit, ils possèderont un compte, pourront visualiser ce dernier mais également **modifier leurs données personnelles**, obtenir un **aperçu des dernières commandes** qu'ils ont effectuées, **supprimer les commentaires postés** etc..
Une barre de recherche est présente permettant de trouver rapidement un produit.

Des fonctionnalités supplémentaires ont été ajoutées dans le but d'avoir un *rendu final plus pertinent*, d'acquérir d*avantage de capacités* et d'avoir un *site plus fonctionnel.*

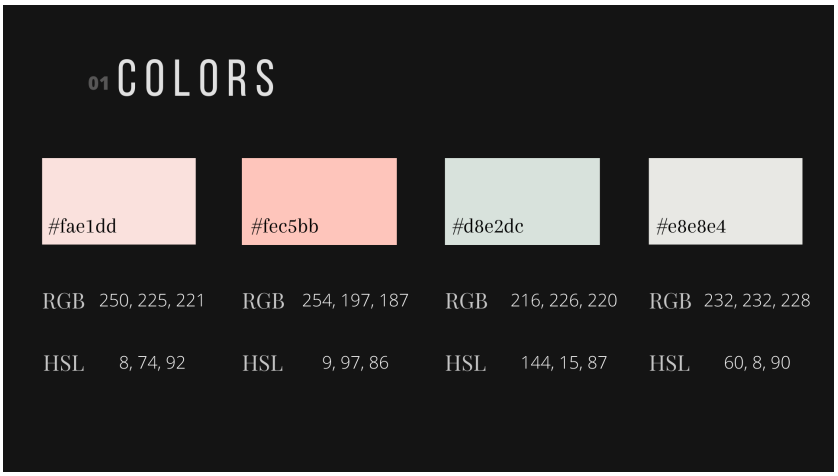
Spécifications fonctionnelles

1 . Conceptualisation graphique

1-1 . Identité graphique

Une des premières étapes de la conception graphique du projet est la mise en place de la *charte graphique.* (car non existante)

Nous avons donc décidé d'une palette de 4 couleurs à l'aide de l'outil **Coolors** . Cet aperçu est disponible dans un dossier *'charte graphique'* du projet.



Palette de couleurs définissant l'identité visuelle.

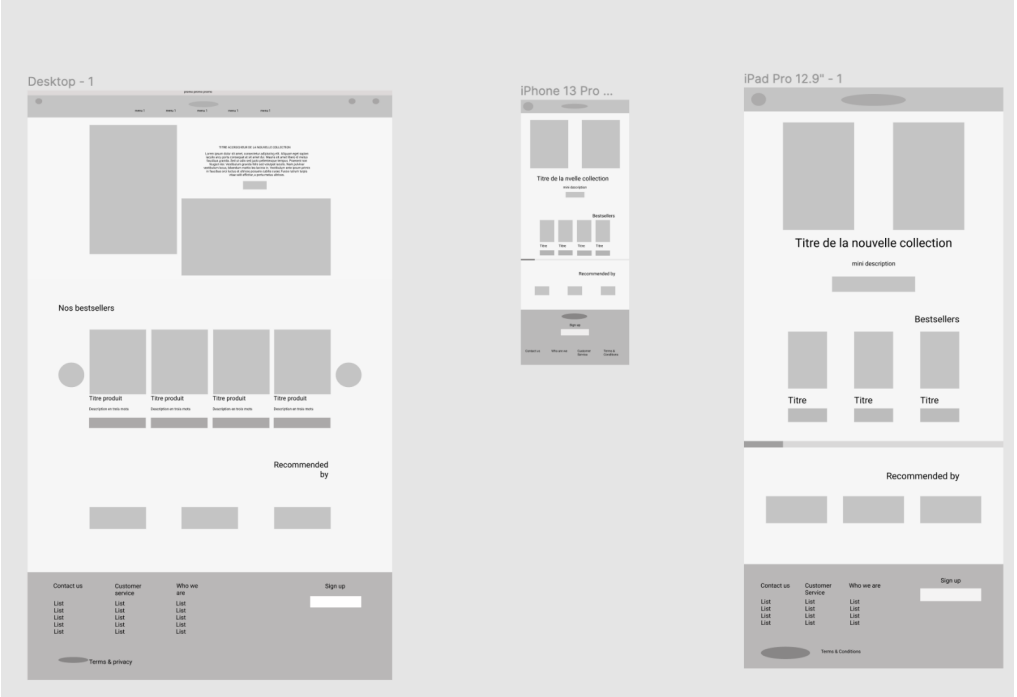
Une *palette plus large* a également été définie apportant *une flexibilité* lors du développement front-end du site en cas de besoin.

Les *Fonts* utilisées ont également été insérées au projet dans un dossier spécifique.

Dans une *étiquette* **Trello**, plusieurs éléments de la charte graphique étaient disponibles, ainsi que des photos déterminant les aspects visuels attendus sur le site.

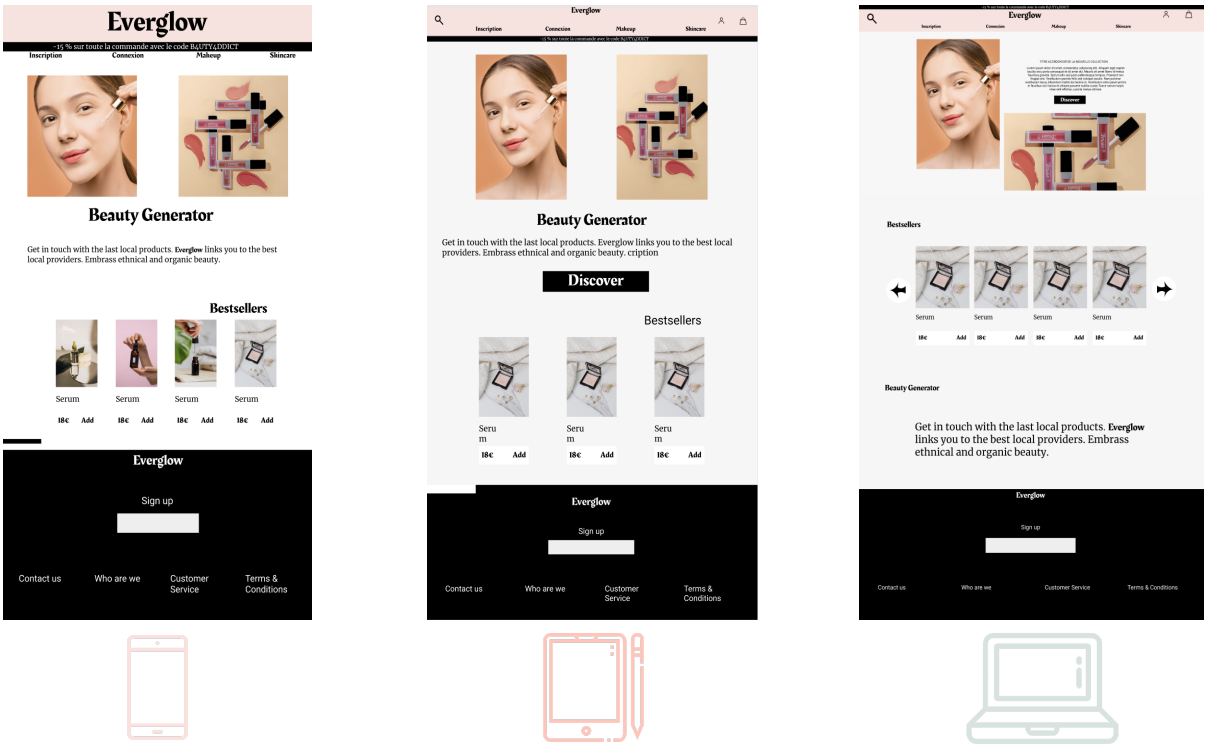
1-2 . Zoning / Wireframe

Une fois la *charte graphique* de l'entreprise établie, nous avons pu développer le *maquettage* du projet.
A l'aide de **FIGMA** nous avons donc commencé par le **zoning.**
Le but du **zoning** étant d'avoir une première approche visuelle du site permettant de réfléchir aux différentes zones de contenus, aux fonctionnalités etc…
afin de *faciliter l'intégration des éléments visuels dans le Wireframe* par la suite.



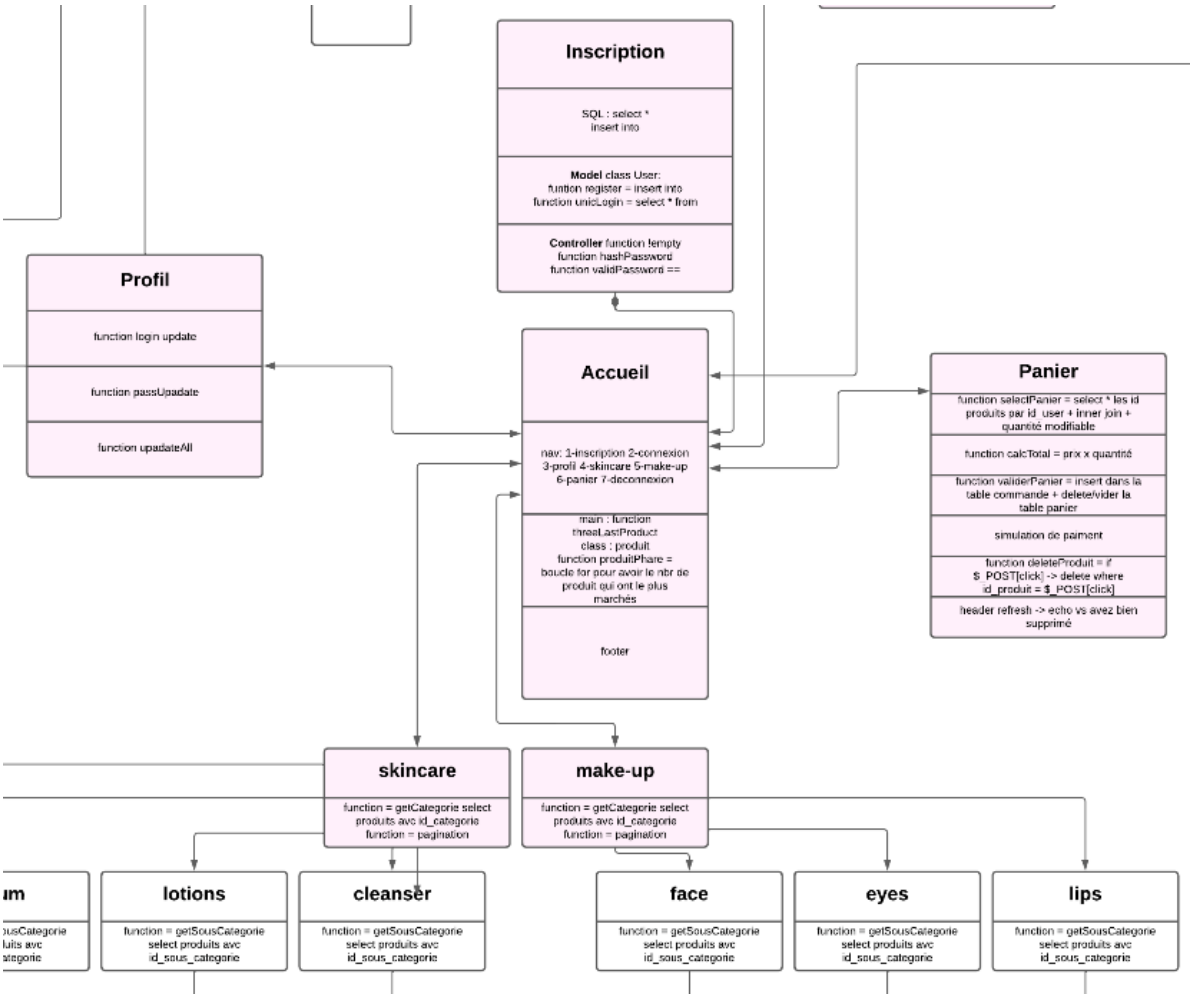
Zoning de la page d'accueil sur les différents supports (mobile, tablette, desktop)

Une fois le **zoning** en place, nous avons continué le maquettage par le **Wireframe** : Le **Wireframe** étant complémentaire au **zoning**, il permet de mieux *visualiser l'agencement des pages*, *l'architecture de l'information* et *d'implémenter les différents éléments de la charte graphique* (logo, couleurs ..).



2 . Arborescence du site

L'arborescence du site a été développée à l'avance, à l'aide d'un schéma réalisé sur **LucidChart.**
Dans ce schéma nous avons pensé aux différentes pages et avons essayé d'y intégrer en amont les appels possibles à la base de données (requêtes).
Cela nous a permis à l'aide de nos modèles de données de pouvoir définir les *fonctionnalités back-end à mettre en place*.



L'arborescence du site s'organise comme cela :

- Page d'accueil
- Page d'inscription
- Page de connexion
- Page mot de passe oublié
- Page Produits (contenant tous les produits)
- Page catégorie (contenant les produits relatifs à une catégorie)
- Page sous-catégorie
- Page détail du produit

- Page du profil utilisateur
- Page panier
- Page confirmation de commande
- Page paiement
- Page validation de commande
- Page concours (inscription)
- Page de résultat de recherche
- Page découvrez votre ‘peau’ (questionnaire)

Un panel administrateur et une *partie back-office* sont également présents sur le site.

3 . Description des fonctionnalités

3-1 . Authentification

Les visiteurs de la boutique ont la possibilité de **créer un compte / se connecter** à l’aide de formulaire. Des *sécurités et vérifications* en base de données sont effectuées à chaque *envoi de formulaire*.

Exemple : Pour l'inscription, une vérification est effectuée au cas où l'adresse e-mail soit déjà utilisée.

3-2 . Oubli de mot de passe

Nous avons implémenté une fonctionnalité de **récupération de mot de passe oublié** à l’aide de **PHPMailer.**
Dès qu’un utilisateur a renseigné son adresse e-mail, un nouveau mot de passe est généré avec la fonction **PHP** *uniqid()* avant de lui être envoyé.
Une adresse e-mail de ‘contact’ de l’entreprise a donc été créée.

3-3 . Catalogue de produits

Sous la barre de navigation du site, l’utilisateur pourra retrouver la *liste des catégories principales de produits.*

Il pourra également se rendre sur la page **“Nos produits”**, page sur laquelle il retrouvera la *totalité des produits disponibles* dans la boutique.

Après avoir choisi une catégorie, une liste contenant les *sous catégories liées* apparaîtra ainsi que tous les *produits de cette catégorie*.
Même logique pour les sous-catégories qui affichent les *produits* qui leurs sont liés.

Concernant l'affichage des produits dans les différentes listes, il est possible d'y retrouver :

- Le nom du produit
- Une photo
- Le prix
- Une redirection vers la "fiche produit"

3-4 . Fiche produit

C'est depuis la **fiche produit** qu'il est possible d'**ajouter un produit à son panier** et de *consulter plusieurs informations* :

<u>Si l'utilisateur n'est pas connecté</u>

- Le nom du produit
- Une galerie photo du produit (minimum 2 photos)
- Une description du produit
- Le prix du produit
- Les commentaires / notes du produit
- Une redirection vers l'inscription / connexion

<u>Si l'utilisateur est connecté</u>

- Un bouton d'ajout au panier
- Un bouton d'ajout aux favoris
- Un choix de la quantité du produit
- Un formulaire d'ajout de commentaire / note

<u>Si le produit possède des couleurs</u>

- Un choix parmis les couleurs proposé pour le produit

Depuis cette page l'utilisateur connecté peut **ajouter le produit à son panier, choisir une quantité, une couleur (si proposée)**.

Il peut également **noter et commenter** un produit à l'aide d'un formulaire.

Enfin, des *produits en rapport* sont proposés plus bas dans la page.

3-5 . Recherche de produits

L'utilisateur a la possibilité de **rechercher un produit** depuis une *barre de recherche*.
L'outil de recherche est présent dans la *barre de navigation* du site, permettant de l'utiliser depuis toutes les pages.

3-6 . Profil utilisateur

Une fois inscrit sur le site, l'utilisateur aura accès à son **espace utilisateur.**
Depuis cet espace, l'utilisateur sera en capacité **visualiser** et de **modifier ses informations** :

- Nom / prénom
- Adresse e-mail
- Adresse de livraison
- Mot de passe

Pour *modifier son mot de passe*, il devra *renseigner le dernier mot de passe connu* en base de données pour son compte.

Il pourra depuis cette page, *consulter les dernières commandes* qu'il a réalisé.

Il aura également la possibilité de **visualiser les commentaires et notes** qu'il a posté sur les différents produits et pourra les **supprimer**.

3-7 . Back office administrateur

Panel administrateur

Un panel administrateur a été créé afin d'y retrouver rapidement des *informations importantes* et de *manières organisées*.

Il est possible pour l'administrateur de *visualiser le nombre d'utilisateurs inscrits, le nombre de commandes réalisées, le chiffre d'affaires ainsi qu'un graphique des meilleures ventes de produits*.

Ajout d'un produit

L'administrateur a la possibilité **d'ajouter un produit à la boutique** à l'aide d'un formulaire.
Il devra renseigner les informations nécessaires et ajouter des photos afin de créer l'affichage du produit sur le site :

- nom
- prix
- description
- Photos
- catégories / sous-catégories

Gestion des produits

L'administrateur a également la possibilité **d'ajouter une / des photos** à un produit.

Il peut aussi **mettre à jour des informations** du produit telles que son prix, sa description…

Enfin, il a la possibilité de **supprimer un produit** de la boutique, ce dernier verra son affichage disparaître.

Promotions

L'administrateur peut **définir un code promotionnel**. Ce code sera visible depuis la totalité des pages du site permettant aux utilisateurs de le *renseigner lors de la validation de la commande*, et de *bénéficier de réduction*.
Un bandeau de défilement affichant le code est visible au-dessus de la barre de navigation.

3-8 . Panier client

L'utilisateur sera en mesure d'ajouter un produit à son panier avant de valider sa commande. Une fois effectué, certaines informations relatives au produit seront disponible depuis le panier :

- Son nom
- Son prix unitaire + total
- Une photo du produit
- La couleur choisie (si proposée)

- Un bouton permettant de modifier la quantité du produit
- La quantité choisie
- Un bouton de redirection vers la confirmation de commande

Le prix total du panier sera visible en dessous du *récapitulatif des produits* et sera *calculé dynamiquement* à chaque mise à jour du panier.

L'utilisateur sera en mesure de **modifier la quantité** d'un produit, mais également de le **supprimer** de son panier.

3-9 . Validation de commande

Il s'agit de l'étape précédent le paiement / validation de la commande. L'utilisateur devra choisir parmi *trois types de livraisons*. (les frais de livraison sont offerts à partir d'un certain prix de commande)
Il devra *renseigner des informations* concernant la livraison à l'aide d'un formulaire et pourra bénéficier d'une réduction sur la commande en entrant un code promotionnel visible sur le site.

3-10 . Solution de paiement

Nous avons choisi d'implémenter **Stripe** dans le but de proposer un service de paiement par carte bancaire fiable et sécurisé.

3-11 . Questionnaire - Trouvez votre look

Une fonctionnalité a été implémentée sous forme de quiz, dans le but de *proposer aux clients des produits* qui pourraient potentiellement les concerner. A l'aide des informations recueillies lors du questionnaire, un tri est effectué parmi les produits afin d'avoir *une proposition pertinente*.

3-12 . Implémentation de l'API instagram

Un aperçu du compte Instagram de l'entreprise est visible depuis la page d'accueil du site. Il a été ajouté à l'aide de l'**API Instagram** et de **Fetch / Javascript**.

3-13 . Jeux concours

Les utilisateurs ont la possibilité de *participer à un jeu concours* en remplissant un formulaire de participation. *Un décompte* est visible permettant de voir le *temps restant* avant le tirage au sort. Un gagnant est choisi aléatoirement parmi les utilisateurs connus en base de données.

Spécifications techniques

4 . Choix techniques et environnement de travail

4-1 . Gestion / organisation du projet

Nous avons choisi d'utiliser **Trello** afin d'*optimiser l'organisation* durant toute l'avancée du projet.
Nous y trouvons notamment :

- La liste des différentes *tâches à effectuer*, permettant ainsi de *répartir le travail de manière équitable*.
- *Les deadlines*, dans le but de *respecter les délais imposés* et donc *avancer efficacement*.
- Des *ressources diverses*. (charte graphique, photos, inspirations)

4-2 . Outils de conceptualisation

Pour le maquettage du site, nous avons décidé d'utiliser **Figma**. Il permet de travailler en groupe sur un même document depuis plusieurs ordinateurs.

Concernant les modèles de données (MCD / MLD) nous les avons réalisés à l'aide de l'outil en ligne : **LucidChart.**
Il permet tout comme Figma de travailler en groupe sur un même document.

4-2 . Environnement de développement

L'environnement de développement utilisé est le suivant :

- Système d'exploitation : **Windows 7**
- Editeur de Code : **Visual Studio Code**
- Outil de versionning : **Gitbash ,Github**

La base de données **Mysql** est hébergée sur **Apache**.
Les différents tests sur le site ont été effectués depuis le serveur local **Wamp**.
Le site a été développé sur les navigateurs suivants : **Google Chrome, Safari**.

4-3 . Choix des technologies

4-3-1 . Front-end

HTML

Le site a été développé à l'aide du *langage de balisage* **HTML5**.

CSS

Des *feuilles de style* ont été appliquées aux balises HTML à l'aide de **CSS3**. Des styles communs ont été développés permettant de rapidement mettre en forme un nouvel élément / fonctionnalité récemment ajouté.
Le responsive design a été géré à l'aide de CSS3.

Javascript

Des éléments du site ont été stylisés à l'aide de **Javascript** permettant ainsi d'obtenir un *rendu dynamique* dans certaines sections / éléments et *améliorer l'expérience utilisateur.*

4-3-2 . Back-end

PHP

La quasi-totalité des fonctionnalités présentes sur le site ont été développées en utilisant la notion de *programmation orientée obje*t à l'aide du langage **PHP** (Hypertext Preprocessor).

MySQL

Nous avons développé notre base de données à l'aide du système de gestion **MySQL**. Ce dernier étant conçu pour fonctionner avec **PHP**, permettant de *manipuler la base de données* et *diriger l'accès à son contenu*.

Javascript

Des requêtes *Ajax* (*HTTP*) ont été initiées à l'aide de la fonction native de **Javascript** : *Fetch*.
Fetch permet de faire des requêtes vers le serveur sans rechargement de page afin de recevoir et travailler avec des données de manière dynamique.

5 . Architecture logicielle

5-1 . MVC

Concernant *l'architecture logicielle*, nous avons mis en place le *design pattern* **MVC** (Model-View-Controller).
Cette architecture a pour principe de séparer les données des méthodes qui les utilisent. Elle permet notamment de faire collaborer le développement front-end et back-end tout en maintenant un code organisé et réorganisable rapidement.

Le MVC se compose de 3 modules distincts :

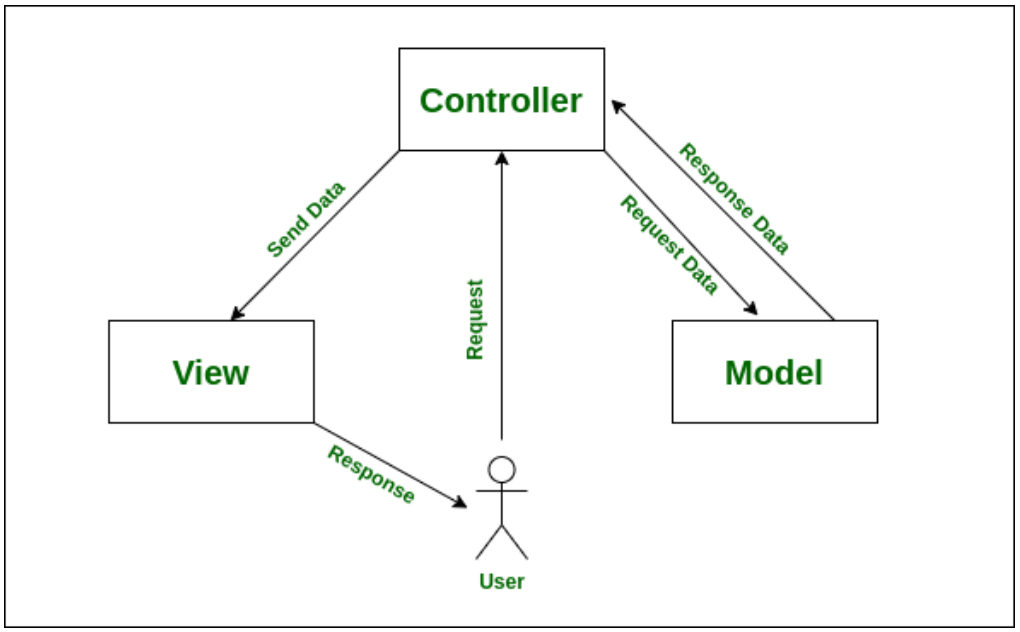
- Modèle : Le modèle contient *des données* ainsi que *la logique en rapport* avec ces données. Il permet de les *modifier* (suppression, ajout, modification) mais également de les *récupérer*.

- Vue : Une vue *contient des éléments visuels* ainsi que la *logique nécessaire* pour *afficher* les données provenant du *modèle*.

- Contrôleur: Le contrôleur est un module qui permet de *traiter les actions de l'utilisateur*. Il est responsable de la *logique du code* et sert *d'intermédiaire entre le modèle et la vue*. Il pourra communiquer avec les *modèles* afin d'obtenir des *informations* qu'il transmettra à la *vue*.

En résumé, lorsqu'un client envoie une requête à l'application :

- La requête envoyée depuis la **vue** est analysée par le **contrôleur**
- Le **contrôleur** demande au **modèle** approprié d'effectuer les traitements et notifie à la **vue** que la requête est traitée.
- La **vue** notifiée fait une requête au **modèle** pour se mettre à jour (par exemple affiche le résultat du traitement via le modèle).

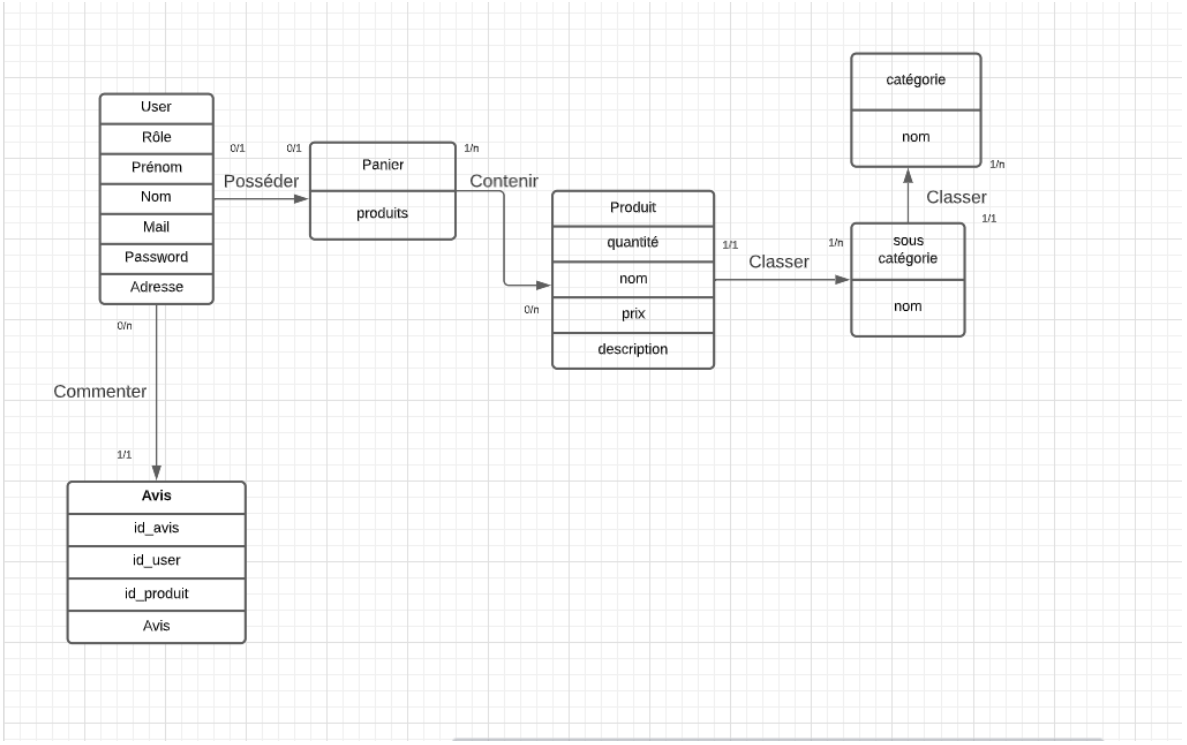


6 . Conception de la base de données

A l'aide de la méthodologie **Merise** et de l'outil en ligne **Lucid Chart**, nous avons conçu les modèles de données.

6-1 . MCD

La première étape consiste à mettre en place le **modèle conceptuel de données** (**MCD**).
Le modèle conceptuel de données est une représentation *logique de l'organisation* des informations et de leurs *relations*, permettant de décrire le *système d'information* à l'aide *d'entités*.



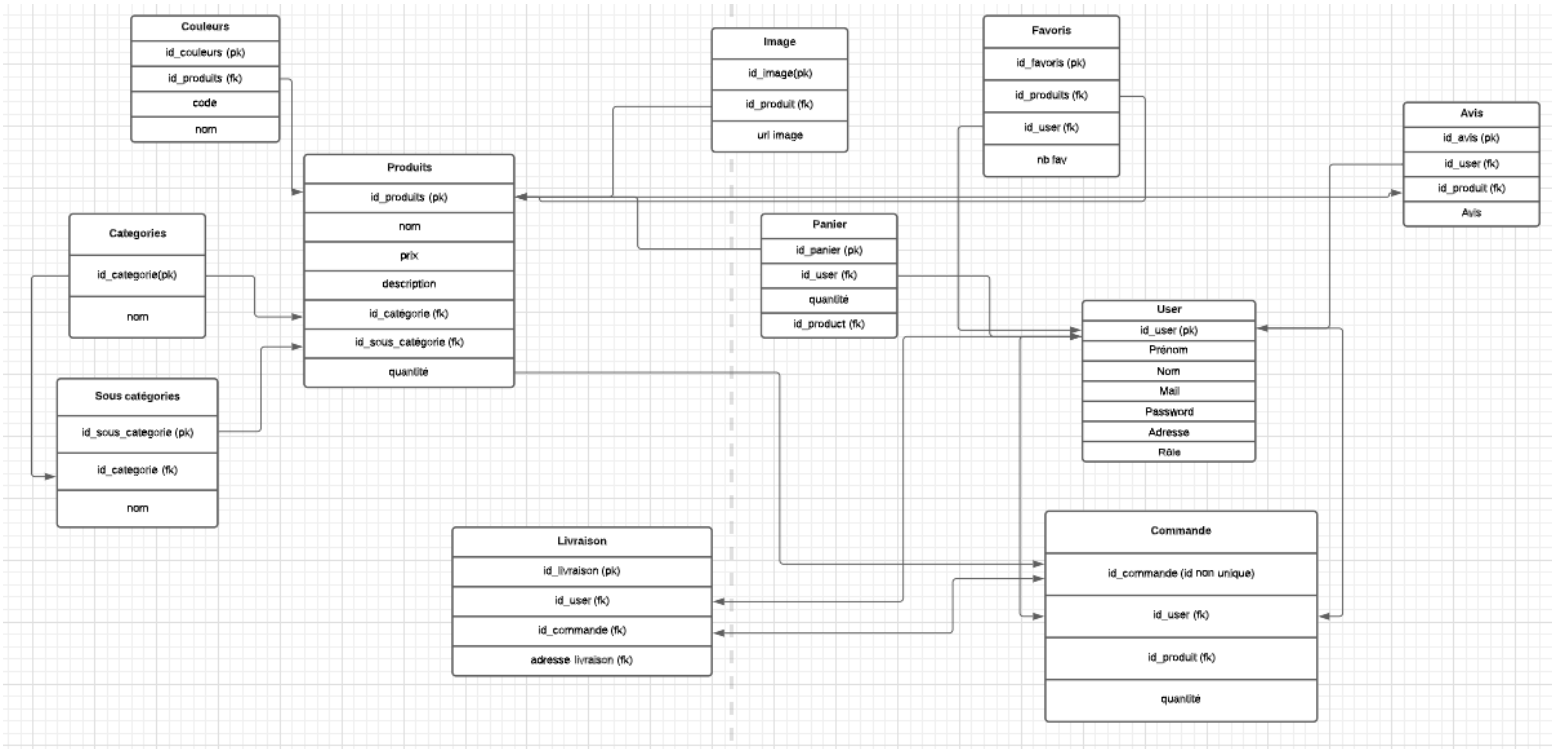
Modèle conceptuel de données (MCD)

6-2 . MLD

Le **modèle logique de données** (**MLD**) est la seconde étape de conception de la base de données.
Le MCD (Modèle Conceptuel de Données) ne peut pas être *implanté dans une base de données* sans modification.

Il est obligatoire de transformer ce modèle. On dit qu'on effectue un passage du modèle conceptuel de données vers le modèle logique de données.

Le MLD pourra être *implanté dans une base de données relationnelle*.
Il est la *représentation en ligne* du schéma représentant la *structure de la base de données.*



Modèle logique de données (MLD)

La base de données s'articule principalement autour des tables **user**, **produits** et **commande**.
La table **user** permet l'*identification des utilisateurs*. Elle est liée avec les tables suivantes :

- Avis
- Favoris
- Commande
- Livraison
- Panier

Permettant ainsi d'identifier à qui *appartient une commande* ou bien *quel utilisateur à ajouter quel produit à ses favoris.*

La table product est elle liée avec les tables :

- Couleurs
- Catégories
- Sous-catégories
- Images
- Commandes
- Panier

Ces différentes liaisons permettent de déterminer les *déclinaisons de couleurs d'un produit, d'afficher des images de ce produit, à quelle catégorie / sous-catégorie il appartient..*

La table commande elle récupère des informations des tables :

- User
- Product
- Livraison

Elle permet de regrouper la totalité des informations relatives à une commande passée par un utilisateur.

Jeu d'essai

Après avoir renseigné ses *informations personnelles* lors de **l'inscription**, l'utilisateur a accès à de *nouvelles fonctionnalités* sur le site. Ses informations sont alors entrées en base de données et il peut se **connecter**. Si l'utilisateur n'a pas rempli le formulaire correctement, une *erreur* lui indiquera la problème. Il ne sera donc pas inscrit.
Après sa **connexion**, il a la possibilité :

- De se rendre sur sa page **profil**. Depuis cette page il pourra *visualiser ses informations* mais pourra aussi les *modifier*. Une sécurité est ajoutée en cas de demande de *modification de mot de passe* nécessitant de renseigner l'ancien mot de passe connu en base de données pour ce compte.

- Il peut **ajouter un produit à son panier** en se rendant sur la **fiche produit**. Depuis la **fiche produit** il peut choisir un *quantité pour ce produit*, il peut *choisir une couleur* si le produit en possède plusieurs. 4 produits en rapport avec celui consulté par l'utilisateur seront proposés.

- Il pourra également **commenter** le produit à l'aide d'un formulaire prévu à cet effet, et **noter** ce dernier sur 5 étoiles. (1 étoile étant le plus mauvais et 5 étoiles le meilleur avis)

- Il peut **ajouter un produit à ses favoris**. Une fois connu en base de données comme favoris, l'affichage du bouton se verra modifié et *apparaîtra en rouge*.

- Son panier sera visible depuis la page concernée. Un *récapitulatif* du panier sera présent ainsi que la possibilité de **modifier la quantité du produit** voir de le **supprimer**. Dès qu'une modification est effectuée depuis la page panier, la base de données est également mise à jour.

- Une fois le panier validé, il sera demandé de remplir des informations *relatives à la livraison* de la commande. Il pourra par la suite renseigner ses informations bancaires dans le module de **Stripe**.

- Lorsque les informations de paiement sont *renseignées et valides*, la table **panier** transfère les *données relatives* à la table **commande** en plus de la date, de l'identité du 'client' concerné et d'autres informations.

- Une fois la commande passée, l'utilisateur peut la *visualiser* depuis son **espace client**.

- Il a également la possibilité de se **déconnecter** depuis la barre de navigation.

Extrait de code

9 . Authentification

Ci-dessous, vous trouverez le *parcours utilisateur* ainsi que la *transition des données* durant **l'inscription** / **connexion** de ce dernier :

Les premières étapes consistent à *vérifier* que toutes les informations nécessaires à la création d'un nouveau compte ont bien été renseignées lors de l'envoi du formulaire :

```php
// Validation du formulaire
if ( isset ($_POST['submit']))
{
    // Vérification des champs
    if ( !empty($_POST['email']))
    {
        // Vérification des champs
        if ( !empty ($_POST['surname']))
        {
```

(code extrait de la page UserController.php)

Si un champ n'est pas *correctement renseigné*, un message d'erreur apparaît permettant à l'utilisateur de résoudre le problème.

Une vérification est effectuée entre le **mot de passe** et la **confirmation** de ce dernier. Ci les deux correspondent, le parcours peut continuer.

La prochaine étape consiste à **vérifier l'unicité des adresses e-mail** en base de données afin d'optimiser le **confort utilisateur** et la **sécurité**. Lors de l'envoi du formulaire **d'inscription**, si le mail renseigné est déjà enregistré en base de données, un message le fera savoir à l'utilisateur.

On vérifie donc la présence de l'adresse e-mail renseignée à l'aide de la fonction **findBy.** (*fonction qui sera expliquée plus bas dans cette section*)

```php
// Vérification de l'unicité des Emails en base de données
$model = new UsersModel();
$valid_email = $model->findBy(['email' => $email]);
```

En fonction du retour, *deux cas* sont possibles :

- L'**adresse email est déjà connue** en base de données, dans ce cas on le fait savoir à l'utilisateur de part un message d'erreur.

```php
} else {
    $error_email = "E-mail déjà utilisé";
}
```

- L'*adresse email n'est pas connue* en base de données. Dans ce cas, les informations renseignées dans le formulaire d'inscription sont **sécurisées** à l'aide d'une fonction crée précédemment : **valid data()**

```php
// Validation des données
function valid_data($données)
{
    //trim permet de supprimer les espaces inutiles
    $données = trim($données);
    //stripslashes supprimes les antishlashs
    $données = stripslashes($données);
    //htmlspecialchars permet d'échapper certains caractéres spéciaux et les transforment en entité HTML
    $données = htmlspecialchars($données);
    return $données;
}
```

Sécurisation des données renseignées lors de l'envoi du formulaire :

```php
// Sécurisation des données du formulaire
$email = valid_data($_POST['email']);
$surname = valid_data($_POST['surname']);
$name = valid_data($_POST['name']);
```

Une fois les étapes précédentes passées avec succès et les données **sécurisées**, le mot de passe renseigné par l'utilisateur est haché à l'aide de la fonction **password_hash()** de **PHP**.
Depuis le **UserController** on instancie un nouveau **model User**, permettant de **créer** en base de données un nouvel utilisateur avec les *données renseignées.*
L'utilisateur est inscrit à l'aide de la fonction **create()**.

```php
// Hashage du mot de passe avant insertion en base de données
$password_hash = password_hash($password, PASSWORD_DEFAULT);


// On crée un nouveau UserModel
$model = new UsersModel();

// On récupère les informations entrées en formulaire
$user_data = $model
->setNom($name)
->setPrenom($surname)
->setEmail($email)
->setPassword($password_hash);

// On inscrit l'utilisateur en base de données
$user_data->create($model);
```

10 . Les méthodes génériques

10-1 . La méthode create

Suite au passage de toutes les *étapes d'authentification*, on peut voir que l'utilisateur est *inscrit en base de données* à l'aide de la fonction **create()**.

Cette fonction provient du **modèle mère** dans lequel sont créées toutes les *requêtes génériques* aux autres modèles. Cette méthode est accessible grâce à **l'héritage**.

Le *modèle User* lui hérite des méthodes présentes dans le *modèle mère* permettant d'utiliser par exemple la fonction **create()**.

La fonction **create()** prend en compte un *paramètre* : *un modèle*.

```php
public function create(model $model)
{
```

Ce modèle est en fait le *tableau de données* recueillies lors du remplissage du formulaire et transmis lors de l'appel de la fonction **create()**.

```php
// On récupère les informations entrées en formulaire
$user_data = $model
->setNom($name)
->setPrenom($surname)
->setEmail($email)
->setPassword($password_hash);
```

3 tableaux sont créés dans la fonction **create()**.

```php
$champs = [];
$inter = [];
$valeurs = [];
```

Ces tableaux permettront d'écrire la requête *dynamiquement* en fonction des *paramètres récupérés*.

On boucle ensuite sur le tableau contenant les informations saisies afin de les *insérer* dans les tableaux précedemment créer :

```php
// On boucle pour éclater le tableau
foreach($this as $champ => $valeur){
    // INSERT INTO annonces (titre, description, actif) values (?,?,?)

    if($valeur != null && $champ != 'database' && $champ != 'table'){
        $champs[] = $champ;
        $inter[] = "?";
        $valeurs[] = $valeur;
    }

}
```

- On insère dans le tableau **champs[ ]** : les “**clefs**” du tableau qui permettront de préciser la **‘colonne’** concernée en base de données.

- On insère dans le tableau **inter[ ]** : autant de **“?”** qu’il y a de *champs à remplir / de clefs* dans le tableau.

- Et enfin dans le tableau **valeurs[ ]** : les valeurs récupérées dans le modèle.

Cela mis en place, les requêtes pourront avoir un nombre de paramètres différents, sans avoir à les réécrire.

La fonction **implode()** de **PHP** a été utilisée afin de transformer les tableaux champs et inter en chaîne de caractères séparés par des **‘, ’** :

```
//transformer la tableau champs en chaine de caractére
$liste_champs = implode(', ', $champs);
$liste_inter = implode(', ', $inter);
```

Une fois les deux tableaux transformés en **chaîne de caractères**, il est possible d'exécuter la requête de cette manière.

```
//On éxecute la requête
return $this->requete('INSERT INTO '.$this->table.' ('.$liste_champs.') VALUES ('.$liste_inter.')', $valeurs);
```

La fonction **create()** retourne alors une requête *dynamique* elle même appelée dans une méthode : **requête()**

Exemple avec un utilisateur fictif :

```
$model = new UserModel()
$user_data = $model
        ->setNom(“Machin”)
        ->setPrenom(“Max”);

$user_data->create($model);
```

ce qui donnera une fois retourner par la fonction **create()** :

```
$this->requete(‘INSERT INTO ‘.$this->table.’ ( nom, prenom ) VALUES ( ? , ? ), [“max”, “machin”] )
```

$this->table est précisé dans le constructeur de chaque classe, permettant de ne pas avoir à le préciser dans aucune requête. La table concernée est donc “**users**”.

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

10-2 . La méthode requête

Elle permet d’effectuer une *connexion à la base de données* en cas de requête, mais vérifie également s' il est nécessaire de **préparer** une requête ou pas en fonction de ses *attributs* et **retourne le résultat** de cette requête.

La méthode **requête()** attend **2 paramètres** :

- $sql : **Une requête sql** (dans l’exemple précédent, la requête écrite à l’aide de la fonction create)

- $attributs : **Un tableau d’attributs** qui peut être nul.

En premier une instanciation de l’objet **Database** est effectuée afin de permettre une *connexion* à la base de données.

```php
//On récupere l'instance de database
$this->database = DataBase::getPdo();
```

On vérifie par la suite la **présence d’attributs** :

- Si des attributs sont définis alors la requête sera **préparée**
- Sinon il s’agira d’une requête qui ne **nécessite pas d’être préparée**.

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

On **retourne ensuite le résultat**.

10-3 . La méthode findBy

Cette méthode permet de *retrouver un élément* de la base de données en *fonction d'un ou plusieurs critères* exemple :

$user->findBy('id' => 3, 'nom' => "Machin")

Elle prend en paramètre un **tableau de critères**.

```php
public function findBy(array $criteres)
{
```

Comme pour la fonction **create()**, deux *tableaux* sont créer :

- champs[ ]
- valeurs[ ]

On effectue un **foreach** sur le *tableau de critères* afin d'insérer dans les informations dans les tableaux précédemment créé :

```php
// On boucle pour éclater le tableau
foreach($criteres as $champ => $valeur){
    //SELECT * FROM annonces where actif = ? AND signale = 0
    //bindValue(1, valeur);
    $champs[] = "$champ = ?";
    $valeurs[] = $valeur;
}
```

- On insère dans le tableau **champs[ ] :** les différentes clefs récupérées dans le tableau de critère, en ajoutant pour chaque clef **=?** .

- On insère dans le tableau **valeurs[ ] :** les valeurs.

Comme pour la fonction **create()**, on transforme le tableau **champs[ ]** en chaîne de caractères à l'aide de **implode()** en ajoutant **' AND '** entre chaque champ.

```php
// On transforme le tableau champs en chaine de caractére
$liste_champs = implode(' AND ', $champs);
```

Nous pouvons ensuite exécuter la requête dans notre fonction **requête()** :

```php
// On éxecute la requête
return $this->requete('SELECT * FROM '.$this->table.' WHERE '. $liste_champs, $valeurs)->fetchAll();
```

On ajoutera directement un **->fetchAll** aprés la fonction car cette dernière retournera un *résultat de requête* qu'il faudra donc récupérer. Étant donné qu'il est possible de récupérer plusieurs données, un **fetchAll** est nécessaire.

Cette fonction a été utilisée pour l'authentification de l'utilisateur précédemment décrite afin de vérifier **l'unicité** d'une adresse e-mail renseignée à **l'inscription** par exemple.

Exemple avec un utilisateur fictif :

$model = new UserModel()

$valid_user = $model->findBy(['prenom' => "Max", 'nom' => "Machin"])

Cette requête une fois retourner par la fonction findBy sera écrite comme telle :

$this->requete('SELECT * FROM '.$this->table.' WHERE '. prenom = ? AND nom = ?, "Max","Machin")

Liste des fonctions génériques mises en place pour le projet :

- **find(id)** : Trouve un utilisateur / produit selon un id
- **findBy(critères)** : Trouve un utilisateur / produit en fonction de critères.
- **findAll()** : Permet de récupérer toutes les informations d'une table
- **delete By(critères)** : Supprimer une ligne en fonction de critères.
- **delete(id)** : Supprimer une ligne en fonction d'un critères
- **create(model)** : Permet d'insérer en base de données à l'aide d'un modèle.
- **update(model)** : Permet la modification de données à l'aide d'un modèle.

Le développement de ces fonctions dites génériques ont apporté beaucoup d'avantages tout au long de l'avancée du projet :

- Elles ont permis de mettre en place un **CRUD sécurisé** rapidement.

- Elles évitent la **réécriture de requêtes** au cas où le nombre de paramètres puisse changer grâce à leur 'flexibilité'.

- Elles permettent de **développer de nouvelles fonctionnalités** très rapidement. En effet, il est possible de créer, modifier, supprimer des données sans avoir à écrire de requête.

11 . Implémentation de Stripe

Stripe est une solution de paiement offrant aux utilisateurs la possibilité d'effectuer *des paiements par carte bancaire*.

Une fois sa *commande validée*, l'utilisateur aura la possibilité de renseigner ses informations de paiement dans le module **Stripe** présent sur la page **paiement**.

On vérifie d'abord que la commande ne soit pas vide :

```
// Si un prix est définit / différent de vide = commande vide
if ( isset ( $_POST['prix']) && !empty ( $_POST['prix']))
{
```

Si c'est le cas, une variable de type **booléen** est transmise afin de confirmer la commande.

on récupère le prix total de la commande avant d'*instancier un nouvel objet* **Stripe** à l'aide d'une **Api Key** récupérer à **l'inscription** sur **Stripe** :

On crée ensuite une instance de paiement **Stripe** dans laquelle il faudra préciser le *montant* ainsi que la *monnaie de paiemen*t.

```
$intent = \Stripe\PaymentIntent::create([
'amount' => intval($prix)*100,
'currency' => 'eur'
```

Veille technologique

Dans le but d'augmenter ses *connaissances personnelles* et *professionnelles* et de *rester dans l'actualité* de ce qu'il peut se faire de nouveau en termes de *technologie* ou autres, il est nécessaire de faire de la veille technologique.

La première semaine de travail sur ce projet à d'ailleurs été prévue pour cela suite à la décision d'adopter le **MVC**.
En effet, le **MVC** nécessitant l'apprentissage de plusieurs *notions et concepts* nous avons décidé de nous accorder du temps afin d'apprendre à quoi sert le **MVC** et comment il fonctionne.

Nous avons suivi de nombreux tutoriels et explications disponibles sur internet et Youtube par exemple. (le site et la chaîne youtube de Grafikart par exemple)
Nous avons développé chacun de notre côté notre propre **MVC** avant de réunir nos connaissances et décider quel exemple est le plus *pertinent et compréhensible*.

Point sécurité

Un point important concernant le développement d'une application web est la **sécurité**. Des *règles de sécurité* sont à respecter tout au long de l'avancée du projet, dans le but de se prémunir d'éventuelles **failles de sécurité**.

7 . Failles de sécurité

Ci dessous, une liste des principales failles de sécurité et comment les éviter :

7-1 . La faille Include

La **faille include** exploite une mauvaise utilisation de la *fonction include*. Elle est utilisée pour exécuter du code **PHP** présent dans une autre page.
Il est possible qu'un fichier **PHP** soit inclus dans l'*url de manière malintentionnée* faisant alors s'exécuter un script pouvant *dérober des données sensibles*.

Comment l'éviter : Il suffit d'effectuer des tests sur les différentes pages de son site. Essayer d'inclure une page qui n'existe pas. Si l'*url transmis est vulnérable* alors **PHP** retournera un message détaillant le problème.

7-2 . La faille XSS

La faille XSS consiste en l'*injection de code / d'un script* directement interprété par le navigateur Web.
La navigateur ne fera pas la différence entre le code officiel du site et celui du dit hackeur. Il est possible de voir une *modification du code de la page, une redirection vers une page mal intentionnée* ou bien un *vol de cookies.*

Comment l'éviter : Pour ce protéger des **failles XSS**, il est possible d'utiliser les fonctions **htmlentities** ou **htmlspecialchar.**
Grâce à ces fonctions, les caractères pouvant être interprétés comme *des balises html* seront remplacés par leurs *entités*. La navigateur affichera alors mot à mot le caractère et ne cherchera plus à l'interpréter.

7-3 . L'injection SQL

L'injection SQL survient lors de la modification d'une requête **SQL**. Elle consiste à injecter par le biais d'un formulaire, des morceaux de code non filtrés. Dans la finalité, la requête se voit détournée de son but premier afin de subtiliser des données sensibles par exemple.

Comment l'éviter : Ajouter des sécurités et des vérifications lors de l'envoi des formulaires :

- à l'aide de **mysql_real_escape_string()** par exemple : Un caractère tel qu'une **'** sera échappé afin d'éviter la manipulation des données.

Utiliser **PDO** et la notion de **requêtes préparées**.

7-4 . La faille Upload

La faille upload intervient lors de l'*upload de fichiers* via un formulaire présent sur le site :

- Photo de produit
- document pdf
- …

Elle consiste à *mettre en ligne des fichiers* malveillants **PHP** qui laisseront la liberté au hackeur de faire ce qu'il veut sur le site.

Comment l'éviter :

- Éviter de laisser cette fonctionnalité possible aux utilisateurs si elle n'est pas indispensable.
- Interdire l'exécution de code depuis le dossier dans lequel sont stockés les fichiers uploadés.
- Vérifiez et autorisez les extensions des fichiers tolérez via une liste blanche.

7-5 . Attaque par force brute

Cette méthode consiste à **trouver le mot de passe** ou la **clé cryptographique** d'un utilisateur dans le but d'*accéder à un service en ligne, à des données personnelles* voire même *à un ordinateur*.

Comment l'éviter :

- Utiliser des *mots de passe fort* lors de l'inscription des utilisateurs permettant de rendre l'attaque plus complexe.
- Utiliser un *regex* lors de la création de ce mot de passe obligeant l'utilisation de *minuscules*, *majuscules*, *chiffres* et *caractères spéciaux*.
- Renouveler régulièrement les mots de passe.
- Utiliser des mots de passe différents pour chaque site.

8 . Exposition des données sensibles

Lors *du stockage de données sensible* en base de données tel que des *mots de passe*, il est indispensable d'utiliser un système **de hachage de mot de passe**. Ce système crée une **clef** (suite de caractères) ou **empreinte** associés à une *chaîne de caractères* évitant de *stocker les mot de passe en clair* en base de données.

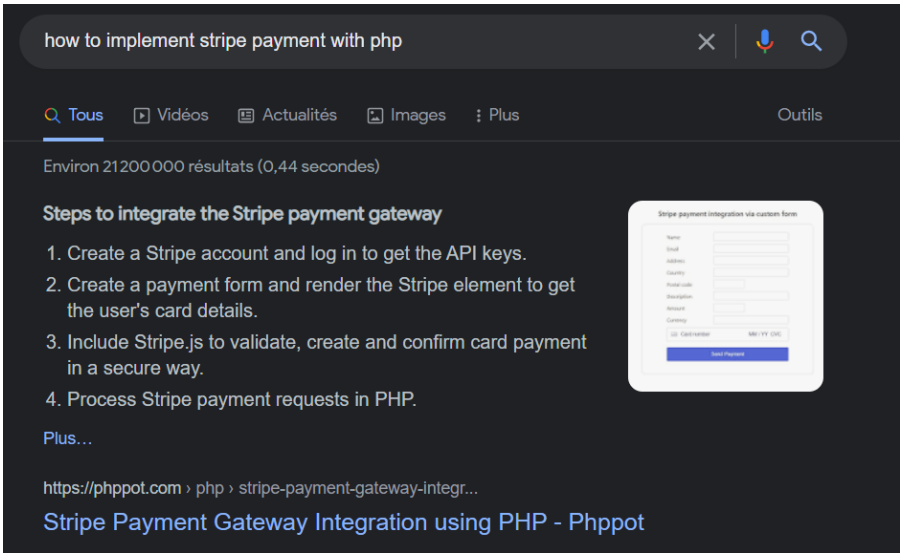
Il est recommandé d'utiliser des algorithmes de hachage sécurisés tels que :

- Argon 5
- Scrypt
- Bcrypt
- PBKDF2.

Recherche anglophone

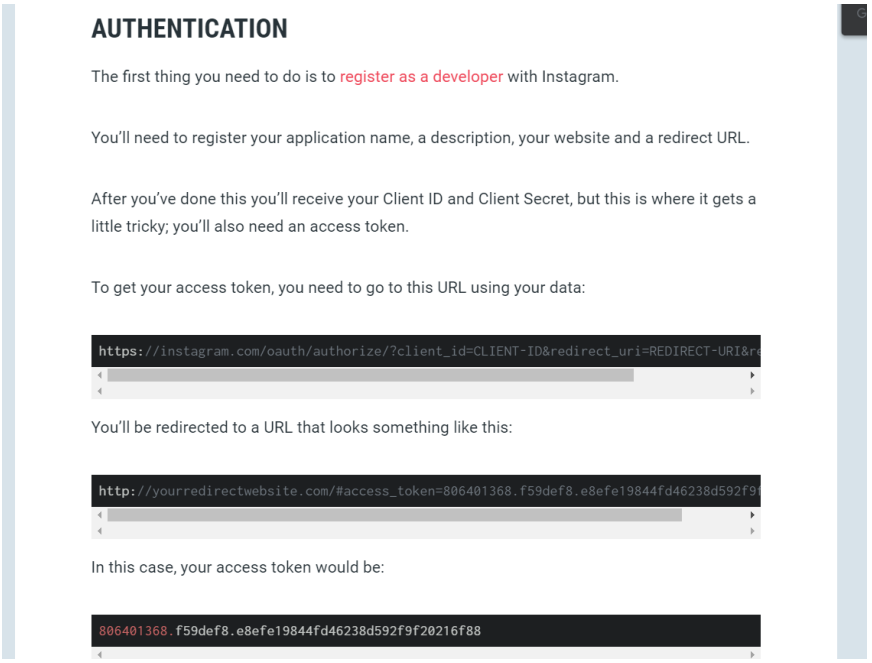
Tout au long de l'avancement du projet, il a été nécessaire de réaliser des recherches en **Anglais**.

Pour de la documentation ou des précisions sur l'implémentation de **Stripe** par exemple.



Traduction du résultat de la recherche effectuée :
1 . Créez un compte Stripe et connectez-vous pour obtenir votre clef d'API.
2 . Créez un formulaire de paiement et ajoutez-y les éléments de Stripe pour récupérer les détails des cartes utilisateurs.
3 . Inclure Stripe.js pour valider, créer et confirmer le paiement par carte de manière sécurisée.
4 . Procédez à une requête de paiement Stripe en PHP.

D'autres recherches ont été effectuées pour l'intégration de l'API Instagram par exemple ou lors de l'apparition d'erreurs durant le développement du projet.



AUTHENTICATION

The first thing you need to do is to register as a developer with Instagram.

You'll need to register your application name, a description, your website and a redirect URL.

After you've done this you'll receive your Client ID and Client Secret, but this is where it gets a little tricky; you'll also need an access token.

To get your access token, you need to go to this URL using your data:

```
https://instagram.com/oauth/authorize/?client_id=CLIENT-ID&redirect_uri=REDIRECT-URI&r
```

You'll be redirected to a URL that looks something like this:

```
http://yourredirectwebsite.com/#access_token=806401368.f59def8.e8efe19844fd46238d592f9
```

In this case, your access token would be:

```
806401368.f59def8.e8efe19844fd46238d592f9f20216f88
```