

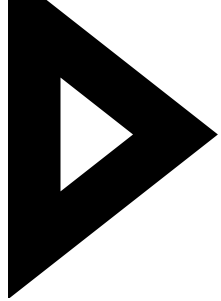
Dossier de Projet - Boutique en Ligne

iLighter

Présenté par Thomas Serdjebi

TITRE RNCP Développeur Web et Web Mobile





SOMMAIRE

1. Introduction	01
1.1 Résumé du dossier de projet.	01
1.2 Présentation personnelle.	01
1.3 Compétences attendues.	02
2. Préparation du projet.	03
2.1 Résumé du projet.	03
2.2 Spécifications fonctionnelles.	04
2.3 Planification et organisation	04
2.4 Gestion du versionning.	05
3. Développement de la partie back-end.	06
3.1 La base de données : modélisation, création, composants d'accès.	06
3.2 L'architecture MVC et le router.	08
3.3 Développement des fichiers de classes.	10
3.4 Développement des fichiers de traitements.	11
3.5 Développement des mesures de sécurité.	13
4. Développement de la partie front-end	16
4.1 Maquettage du site Web.	16
4.2 Le header et le footer.	17
4.3 Développement de la mise en page et du responsive.	18
4.4 Développement des affichages dynamiques.	20
5. Parcours utilisateur : le panier.	23
6. Veille et recherche documentaire.	29
6.1 Veille sur les failles de sécurité en anglais.	29
6.2 Extrait et traduction de la documentation en anglais.	29
Conclusion.	30

Annexes

Annexe 1 : Modèle Conceptuel des Données

Annexe 2 : Modèle Logique des Données

Annexe 3 : Vue relationnelle de la base de données




1. Introduction

1.1 Résumé du dossier de projet

Ce dossier présente une partie de mon travail effectué en formation pour l'obtention du titre RNCP Développeur Web et Web Mobile au sein de l'école La Plateforme_ située dans le deuxième arrondissement de Marseille.

Il présente le projet "Boutique en ligne" réalisé en groupe de 3 avec mes camarades Valentin Mathieu et Stéphane Mathieu. Nous avons développé la boutique en ligne de vente de briquets au détail et en gros nommée iLigther. La boutique a été développée essentiellement avec les langages de programmation web HTML, CSS et PHP.

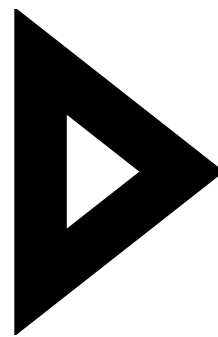


1.2 Présentation personnelle

Je m'appelle Thomas Serdjebi, et j'habite à Marseille dans le 6ème arrondissement. J'ai suivi la formation pour l'obtention du titre RNCP Développeur Web et Web Mobile au sein de l'école la Plateforme_ à Marseille.

Initialement issu d'un parcours d'étude en comptabilité et fort d'une expérience professionnelle de 3 ans, j'ai souhaité me réorienter dans le domaine du web afin de relever de nouveaux défis professionnels et satisfaire ma grande curiosité.

J'ai choisi de préparer le titre RNCP de Développeur Web et Web Mobile à la Plateforme_ pour sa pédagogie d'apprentissage orientée projet, c'est-à-dire par la réalisation de multiples projets en autonomie au cours de l'année de formation.





1. Introduction

1.3 Compétences du titre RNCP Développeur Web & Web Mobile

Le projet Boutique en ligne présenté dans ce dossier couvre l'ensemble des compétences attendues dans le référentiel du titre RNCP Développeur Web et Web Mobile :

Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

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

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
- Elaborer et mettre en oeuvre des composants dans une application de gestion de contenu ou e-commerce



2. Préparation du projet

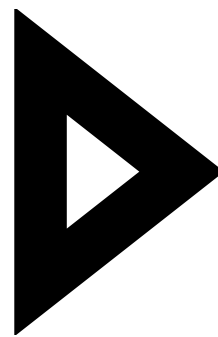
2.1 Résumé du projet

J'ai réalisé le projet Boutique en Ligne avec deux de mes camarades de formation, Valentin Mathieu et Stéphane Mathieu. Nous avons choisi de créer une boutique en ligne de vente de briquet en gros et au détail, du nom de iLighter. Ce projet a été réalisé environ en un mois et demi entre Février et Mars 2022. Les étapes clefs de la réalisation de ce projet ont été :

- l'organisation et la répartition des tâches
- le maquettage du site web
- la récupération des informations des produits sur différents sites de ventes de briquets
- la documentation et la veille en anglais sur le développement d'un site web e-commerce et de ses fonctionnalités
- la modélisation et la création de la base de données
- le développement back-end des fonctionnalités d'administration du site web
- le développement back-end des fonctionnalités des utilisateurs du site web
- le développement front-end de l'interface administrateur du site web
- le développement front-end de l'interface utilisateur du site web

Le projet a été développé avec l'éditeur de code Visual Studio Code et nous avons utilisé gitHub pour stocker notre code et gérer le versionning. Il a été développé en Programmation Orientée Objet qui est un modèle de langage de programmation qui s'articule autour d'objets et de données, plutôt que d'actions et de logique.

Ce projet est le projet clef de notre année de formation puisqu'il nous a permis de nous former de façon complète et d'acquérir l'ensemble des compétences pour l'obtention du titre de Développeur Web et Web Mobile, mais également des compétences transversales comme le travail en équipe.



2. Préparation du projet

2.2 Spécifications fonctionnelles

Voici la liste non exhaustive des fonctionnalités que notre Boutique en Ligne devait comporter :

- 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
- Validation du panier (simulation du paiement)
- Design contemporain et respectant la charte graphique

2.3 Plannification et organisation

Notre groupe a utilisé Trello, un outil de gestion de projet en ligne, afin de définir la liste des tâches à effectuer et les répartir entre les membres du groupe. En effet, Trello permet de créer des cartes de tâches que l'on peut affecter à un membre du groupe. Dans ces cartes il est possible de créer des TO DO LIST des plusieurs étapes de chaque tâche afin de pouvoir suivre l'avancement de chaque tâche. Au niveau de la répartition du travail, nous avons choisi de nous répartir le travail par tâche globales afin que chacun puisse avancer à son rythme sans se retrouver freiner par les autres dans l'avancement du site web.

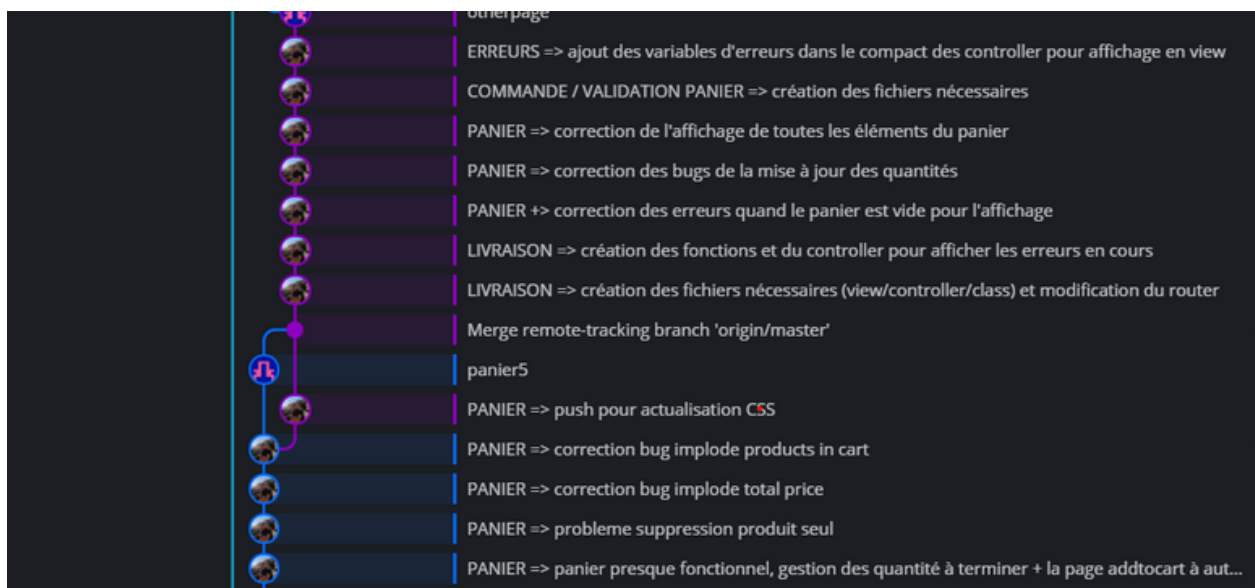
2. Préparation du projet

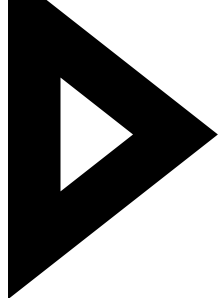
2.4 Gestion du versionning

En ce qui concerne la gestion des versions, nous avons utilisé GitHub, qui est une plateforme open source de gestion de versions et de collaboration destinée aux développeurs de logiciels.

Après avoir créé un dépôt sur le GitHub de Stéphane Mathieu, qui nous a ajouté comme collaborateur sur le dépôt du projet, chacun des membres a cloné le dépôt en local sa station de travail. Ensuite nous avons créé une branche pour chacun des membres du groupe. Après avoir effectué une tâche, chaque membre du groupe pouvait donc faire un commit avec des informations détaillées sur la fonctionnalité qu'il avait développé puis lancer une fusion(merge) sur la branche principale.

Avec le recul, nous avons constaté qu'il aurait été plus pratique de créer une branche par fonctionnalité, afin d'éviter des commits trop lourds et modifiant un trop grand nombre de fichiers en même temps. En effet, nous avons parfois rencontré des conflits de merge au cours du développement du site web que nous avons dû corriger.





3. Développement de la partie back-end

3.1 La base de données : modélisation, création, composants d'accès

Modélisation de la base de données

Pour la modélisation de la base de données, nous avons utilisé le site Lucidchart qui est une plateforme de collaboration en ligne, basée sur le cloud, permettant la création de diagrammes et la visualisation de données, et autres schémas conceptuels. Nous avons modéliser la base de données en utilisant la méthodologie MERISE.

Dans un premier temps, nous avons donc créer le Modèle Conceptuel de Données,(MCD). Dans ce modèle, les informations sont représentées logiquement en utilisant un ensemble de règles et de diagrammes codifiés :

- les entités (1 rectangle = 1 objet) ;
- les propriétés (la liste des données de l'entité) ;
- les relations qui expliquent et précisent comment les entités sont reliées entre elles par un verbe
- les cardinalités représentant le nombre minimum et maximum d'instances autorisées à participer à la relation. La cardinalité est définie pour les deux sens de la relation.

Dans un seconds temps, nous avons créer le Modèle Logique de Données découlant du MCD. Dans ce modèle, plusieurs transformations sont effectuées depuis le MCD :

- chaque entité devient une table et les identifiants de la classe d'entité sont appelés clés primaires de la table
- les propriétés deviennent les attributs de la table
- disparition des verbes faisant les liens entre les tables
- disparition des cardinalités et apparition des clés étrangères en fonction des cardinalités qui étaient établis entre les tables.

Vous pouvez retrouver le MCD et le MLD en annexes.



3. Développement de la partie back-end

Création de la base de données

Dans un troisième temps, nous avons donc créé la base de données sur PHPMYADMIN, une application web programmée en PHP utilisée pour la gestion des bases de données MYSQL et MariaDB.

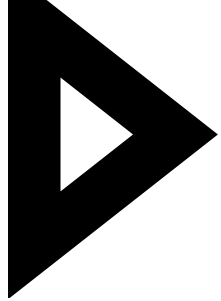
Dans la requête de création de la base de données, nous avons spécifié que le moteur de stockage de chaque table soit en InnoDB afin de maintenir l'intégrité référentielle de la base de données par la gestion des clés étrangères et des contraintes d'intégrité.

Autre remarque, l'ordre de création des tables dans la requête de création de la base de données est important : en effet, si une table contient une clé étrangère provenant d'une autre table, la table d'où provient la clé étrangère doit être créée en amont.

Composants d'accès à la base de données

L'accès à la base de données a été développé dans le fichier Database.php. Il a été développé en PHP Data Objects qui est une extensions définissant l'interface pour accéder à une base de données orientée objet et utilisant la classe PDO.

```
1  <?php
2  namespace Models;
3  use PDO;
4
5  class Database{
6
7      public static function getPdo(): PDO{
8          $pdo = new PDO('mysql:host=localhost;dbname=boutique;charset=utf8', 'root', '', [
9              PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
10             PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
11         ]);
12         return $pdo;
13     }
14 }
15 ?>
```



3.2 L'architecture MCV et le routeur

L'architecture MVC

Afin de mieux organiser notre code source, nous avons développé notre site web en suivant l'architecture Model-View-Controller. En effet, ce modèle permet de séparer la logique du code en trois parties que l'on retrouve dans des fichiers à rangés dans des dossiers distincts :

- **Model** : cette partie gère les données du site. Son rôle est d'aller récupérer les informations brutes dans la base de données, de les organiser et de les assembler pour qu'elles puissent être ensuite traitées par le contrôleur, et inversement, c'est-à-dire d'utiliser les données récupérées par le contrôleur pour ajouter, modifier ou supprimer des éléments de la base de données. On y trouve donc les composants d'accès aux données et les fichiers de classe spécifiques à chaque table de la base de données et contenant des fonctions utilisant des requêtes SQL.
- **View** : cette partie se concentre sur l'affichage. Elle ne fait presque aucun calcul et récupère les variables pour savoir ce qu'elle doit afficher. On y trouve essentiellement du code HTML mais aussi des boucles et conditions PHP, permettant d'afficher plusieurs éléments avec un code raccourci.
- **Controller** : cette partie gère la logique du code qui prend les décisions. C'est l'intermédiaire entre le Model et la View. Le contrôleur va demander au modèle des données, les analyser, prendre des décisions et renvoyer le texte à afficher à la View et inversement. En effet, le contrôleur va aussi récupérer des informations de la View comme pour les données des formulaires, traiter les informations et utiliser les classes et fonctions du Model des données pour qu'il puisse ajouter, modifier, ou supprimer des éléments de la base de données.

3. Développement de la partie back-end

Le router

Pour compléter l'architecture MVC nous avons mis en place un router, qui est la page que l'utilisateur demande pour se connecter au site. C'est donc le script PHP de cette page qui est exécuté. Cette page est chargée de récupérer l'action envoyée avec la demande de la page et d'appeler le fichier de traitement PHP du dossier Controller correspondant.

Pour ce faire, nous avons utilisé la classe de routage AltoRouter qui utilise les méthodes http et utilise un routage dynamique avec des paramètres de routes nommés. Toutes les requêtes sont redirigées vers l'index.php avec le fichier .htaccess placé à la racine du site qui va agir sur les fichiers du répertoire où il se trouve ainsi que tous ses sous-répertoires, et va aussi permettre la réécriture des URLs.

Le router a été configuré dans le fichier index.php avec les fonctions map et match de la classe altorouter. La fonction map prend 4 paramètres :

1. la méthode de requête HTTP (généralement GET ou POST)
2. l'itinéraire qui doit être suivi et sera affiché dans l'URL
3. le fichier de traitement PHP qui fait office de controller de la page lorsque la route du paramètre précédent correspond
4. le nom unique de la route

```
$router = new AltoRouter();
$router->setBasePath('/boutique-en-ligne');
//root user//
$router->map('GET', '/home',function(){ $controller = new \Controllers\Index(); $controller->index();},'home');
$router->map('GET/POST', '/connexion', function(){ $controller = new \Controllers\User\Connexion(); $controller->connexion();},'connexion');
$router->map('GET/POST', '/inscription',function(){ $controller = new \Controllers\User\Inscription(); $controller->inscription();},'inscription');
$router->map('GET/POST', '/deconnexion',function(){ $controller = new \Controllers\User\Deconnexion(); $controller->deconnexion();},'deconnexion');
```

La fonction match tente de faire correspondre un chemin d'URL avec un ensemble de routes. Si le match ne trouve pas d'informations, il redirige l'utilisateur sur la page définie dans la fonction, soit la page d'accueil "Home" pour notre boutique en ligne.

```
/* Match the current request */
$match = $router->match();
if (is_array($match)){
    if(is_callable($match['target'])){
        call_user_func_array($match['target'],$match['params']);
    }
}
else{
    Http::redirect("home");
}
```

3. Développement de la partie back-end

3.3 Développement des fichiers de classe (Model)

Instanciation de la connexion à la base de données et utilisation de l'héritage

Après avoir créé le fichier d'accès Database.php, nous avons créé une classe abstraite Model qui instancie la connexion à la base de données dans son constructeur grâce à la variable `$this->pdo`.

```

1  <?php
2
3  namespace Models;
4
5  use Models\Database;
6
7  abstract class Model {
8
9      protected $pdo;
10
11     public function __construct(){
12         $this->pdo = Database::getPdo();
13     }
14 }
15
16 ?>

```

Ainsi, tous fichiers de classe vont hériter de la classe Model afin de pouvoir accéder à la base de données en utilisant le mot clé `extends` après le nom de la classe.

Développement des fonctions et méthodes de classes en programmation orientée objet

Les fonctions et méthodes de classes ont été développées en programmation orientée objet. En effet, les requêtes SQL sont préparées et exécutées sur la ou les valeurs d'une clé fournie par le fichier de traitement (controller) qui aura par exemple récupéré les informations saisies dans un formulaire par l'utilisateur et appellera la fonction nécessaire de la classe. Les fonctions de bases CRUD (Create, Read, Update, Delete) ont été créées pour les classes qui le nécessitait comme la classe User(utilisateur) et la classe Article(produits),:

```

public function InsertUser($firstname,$lastname,$email,$password,$number,$address,$date,$role):void{

    $data = [
        'firstname' =>$firstname,
        'lastname' =>$lastname,
        'email' =>$email,
        'number' =>$number,
        'password' =>$password,
        'address' =>$address,
        'date' =>$date,
        'role' =>$role,
    ];

    $query = "INSERT INTO users (email, firstname, lastname, password, address, number, date,role) VALUES (:email, :firstname, :lastname, :password, :address, :number, :date ,:role)"
    $insert_user = $this->pdo->prepare($query);
    $insert_user->execute($data);
}

```

3. Développement de la partie back-end

3.4 Développement des fichiers de traitement PHP (Controller)

Instanciation du model utilisé dans les fichiers de traitement

D'abord nous avons créé un fichier Controllers.php qui contient une classe abstraite Controllers ayant deux propriétés : \$model et \$modelName. Dans le constructeur de la classe on instancie le model (la classe) qui sera utilisée grâce à la variable \$this->model qui prendra la valeur du \$modelName défini dans chaque fichier de traitement.

```
<?php
namespace Controllers;

abstract class Controllers{

    protected $model;
    protected $modelName;

    public function __construct(){
        $this->model = new $this->modelName();
    }
}
```

```
class Inscription extends Controllers{

    protected $modelName = '\Models\User::class;

    public function Inscription(){
```

Un fichier de traitement PHP par pagea été créé. Grâce au mot clé extends après le nom de la classe attribuée au controller, le fichier hérite de la méthode construct instanciant la classe utilisée.

On attribue à la variable \$modelName le nom de la classe dont aura besoin le fichier de traitement pour appeler des fonctions. Chaque fichier de traitement contient une fonction globale qui sera appelée grâce au router et permettra d'exécuter la gestion des erreurs et le lancement des fonctions souhaitées. Dans le cas où, plusieurs classes doivent être utilisées dans un même fichier de traitement, il suffit de créer une nouvelle variable \$model à laquelle on attribuera le nom de la nouvelle classe qu'on souhaite utiliser.

Gestion des erreurs

Les erreurs sont gérées par une variable \$check qui prend la valeur true. Dès qu'une erreur est vérifiée par une condition IF ou ELSE IF, la variable \$check devient false et une variable d'erreur est créée : elle sera affichée dans la page PHP en utilisant une condition IF couplée à la fonction PHP isset(). A la fin du fichier de traitement, si aucune erreur n'est détectée, alors \$check a toujours la valeur true et la fonction souhaitée est lancée.

```
if(empty($_POST['firstname'])){
    $check = false;
    $error_firstname = "Renseignez votre prénom.";
}
```

```
if($check){
    $password = password_hash($password, PASSWORD_BCRYPT);
    // insert les donner dans la bdd
    $add_user= $this->model->InsertUser($firstname,$lastname,$email,$password,$number,$address,$date,$droits);
```

3. Développement de la partie back-end

Le Renderer

En aval de chaque fichier de traitement du controller, on peut trouver la fonction render issue de la classe renderer créée dans le dossier Model.

C'est une méthode d'affichage. En effet, elle prend en premier paramètre le nom de la view puis en second les données qui sont réunies dans un tableau grâce à la fonction PHP compact().

Ainsi toutes les variables créées dans le fichier de traitement pourront être affichées dans la view dans la page désignée dans la fonction. Nous avons créé deux fonctions render : une première pour toutes les pages accessibles aux utilisateurs clients, une deuxième pour toutes les pages accessibles aux utilisateurs administrateur.

```

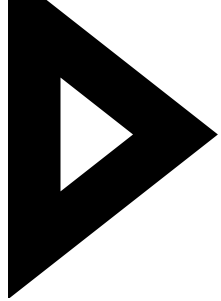
1  <?php
2  namespace Models;
3  class Renderer {
4
5      public static function render(string $path, array $variable = []){
6
7          extract($variable); // Importe les variables dans la table des symboles
8          ob_start();
9          require_once('view/' . $path . '.html.php');
10         $pageContent = ob_get_clean();
11         require_once('view/layout.html.php');
12     }
13
14
15     public static function render2(string $path, array $variable = []){
16         extract($variable); // Importe les variables dans la table des symboles
17         ob_start();
18         require_once('view/' . $path . '.html.php');
19         $pageContent = ob_get_clean();
20         require_once('view/layout-admin.html.php');
21     }
22
23 }
24
25
26 ?>

```

```

$pageTitle = "inscription";
@Renderer::render('users/inscription',compact('pageTitle', 'display_form', 'error_firstname',
'error_lastname', 'error_email', 'error_address', 'error_password', 'error_password_confirm',
'firstname', 'lastname', 'email', 'address', 'message'));

```



3. Développement de la partie back-end

3.5 Développement des mesures de sécurité

Afin de répondre aux besoins de sécurité du site internet, nous nous devons de mettre en place des mesures de sécurité nécessaires. En effet, la création d'un site internet nécessite de sécuriser tous les champs où l'utilisateur a la possibilité de saisir des données entrantes, c'est-à-dire les formulaires et l'URL.

Sécurisation des formulaires

Les formulaires ont été sécurisés de plusieurs façons : d'abord en utilisant une méthode POST afin que les renseignements envoyés par l'utilisateur n'apparaissent pas dans l'URL. Ensuite, nous avons utilisé les fonctions php `htmlspecialchars()` et `htmlspecialchars()` sur les données récupérées par les fichiers de traitement et stockées dans des variables. Ces fonctions permettent de transformer les caractères éligibles en entités HTML et permettent de contrer les attaques de types injections SQL ou XSS Cross Site Scripting.

```
$password = htmlspecialchars(htmlentities($_POST['password']));  
$email = htmlspecialchars(htmlentities(trim($_POST['email'])));  
$number = htmlspecialchars(htmlentities(trim($_POST['number'])));  
$firstname = htmlspecialchars(htmlentities(ucwords(strtolower(trim($_POST['firstname'])))));  
$lastname = htmlspecialchars(htmlentities(ucwords(strtolower(trim($_POST['lastname'])))));  
$address = htmlspecialchars(htmlentities($_POST['address']));  
$password_confirm = htmlspecialchars(htmlentities($_POST['password_confirm']));
```

Nous avons également ajouté dans la gestion des erreurs des conditions IF qui permettent de vérifier la cohérence de la saisie de l'utilisateur en fonction du champ de saisie. Par exemple, pour la saisie d'un numéro de téléphone, on vérifie que la saisie de l'utilisateur est bien numérique avec la fonction PHP `is_numeric()` et grâce à la fonction `strlen()` s'il y a bien 10 chiffres. Autre exemple, pour la saisie d'un prénom, on vérifie à l'aide de la fonction PHP `preg_match()` avec un regex si le prénom ne contient ni chiffres ni caractères spéciaux. Autre exemple, pour la saisie d'un prénom, on vérifie à l'aide de la fonction PHP `preg_match()` avec un regex si le prénom ne contient ni chiffres ni caractères spéciaux.

```
elseif(!is_numeric($number)){  
    $check = false;  
    $error_number = "Votre numéro de téléphone n'est pas au bon format.";  
    $number = '';  
}  
  
elseif(strlen($number) != 10 ) {  
    $check = false;  
    $error_number = "Votre numéro de téléphone doit contenir 10 chiffres.";  
    $number = '';  
}
```

3. Développement de la partie back-end

Pour finir sur la sécurisation des formulaires, les fonctions de classe ont été développées en programmation orientée objet et en utilisant des requêtes préparées. En effet, les requêtes préparées permettent de séparer la requête SQL des paramètres d'exécution des requêtes qui sont notamment les saisies en input de l'utilisateur ou les saisies de l'utilisateur dans l'URL.

En séparant les deux, le moteur de stockage de la base de données va donc traiter toute donnée entrante non pas comme une requête SQL mais comme une chaîne de caractère de type string qui va donc empêcher l'exécution d'une requête malveillante saisie par l'utilisateur en données entrante.

Voici ci-dessous plusieurs exemples de fonctions exécutées à partir de formulaires qui ont des requêtes préparées séparées des paramètres d'exécution :

```
// insert le user dans la bdd
public function InsertUser($firstname,$lastname,$email,$password,$number,$address,$date,$role):void{

    $data = [
        'firstname' => $firstname,
        'lastname' => $lastname,
        'email' => $email,
        'number' => $number,
        'password' => $password,
        'address' => $address,
        'date' => $date,
        'role' => $role,
    ];

    $query = "INSERT INTO users (email, firstname, lastname, password, address, number, date,role) VALUES (:email, :firstname, :lastname, :password, :address, :number, :date, :role)";
    $insert_user = $this->pdo->prepare($query);
    $insert_user->execute($data);
}
```

```
public function UpdateProfil($email,$firstname,$lastname,$address,$number,$id){

    $data = [
        'email'=>$email,
        'firstname'=>$firstname,
        'lastname'=>$lastname,
        'address'=>$address,
        'number'=>$number,
        'id'=>$id,
    ];

    $query = " UPDATE users SET email = :email, firstname = :firstname, lastname =:lastname, address = :address, number = :number WHERE id = :id";
    $update = $this->pdo->prepare($query) ;
    $update->execute($data);
}
```


3. Développement de la partie back-end

Gestion des accès aux pages

Lors de la création de la base de données, dans la table users nous avons créé un champ role qui peut prendre la valeur "utilisateur" ou "admin". Ensuite, lors de la connexion d'un utilisateur, on utilise la fonction `findAllInfosUser()` définie dans la classe User, prenant en paramètre l'email saisi par l'utilisateur lors de sa connexion et qui retourne un tableau contenant toutes les informations de l'utilisateur. Plusieurs variables de session sont alors créées à partir du tableau qui est retourné :

```
session_start();
$_SESSION['email'] = $user['0']['email'];
$_SESSION['role'] = $user['0']['role'];
$_SESSION['userId'] = $user['0']['id'];
```

Ainsi, à partir de ces variables de session, nous avons pu mettre en place la gestion des accès aux pages avec une condition IF dans le fichier de traitement de chaque page.

Par exemple pour accéder aux pages du panel administrateur, si l'utilisateur a une `$_SESSION['role']` ayant la valeur "admin" il aura alors accès aux pages administrateur et sera redirigé vers le panel administrateur au moment de sa connexion. Dans le cas inverse, si l'utilisateur ayant une `$_SESSION['role']` ayant la valeur "utilisateur" et essaye d'accéder aux pages administrateur via l'URL, il sera simplement redirigé vers la page d'accueil et n'aura pas accès au contenu des pages.

```
public function AdminArticle(){
    session_start();
    if($_SESSION['role'] == "admin"){
        $Article = $this->model->findAllArticle();
        $pageTitle = "adminArticle";
        $render::render2('admin/AdminArticle', compact('pageTitle','Article'));
    }else{
        Http::redirect("home");
    }
}
```

Sécurisation des paramètres GET dans l'URL

Afin d'afficher certaines informations, le paramètre GET dans l'URL peut être utilisé comme paramètre d'une fonction appelée dans le fichier de traitement PHP de la page qui va retourner les informations souhaitées, comme par exemple pour l'espace mon compte affichant les informations personnelles de l'utilisateur. En utilisant la fonction `htmlentities()` lors de la récupération du paramètre GET de l'URL, on le sécurise contre les attaques de type injection SQL. On peut également vérifier la nature du paramètre GET récupéré : par exemple, pour un paramètre GET comme l'id d'un utilisateur qui est numérique, nous vérifions que ce paramètre soit en effet numérique avec la fonction PHP `is_numeric()`. Cela doit être couplé à des fonctions en PDO et des requêtes préparées pour être optimal.

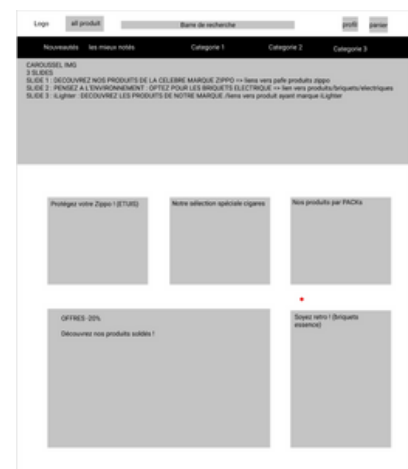
4. Développement de la partie front-end

4.1 Maquettage du site

Pour maquetter notre site Web, nous avons utilisé Figma, qui est un éditeur de graphiques vectoriels et un outil de prototypage collaboratif en ligne.

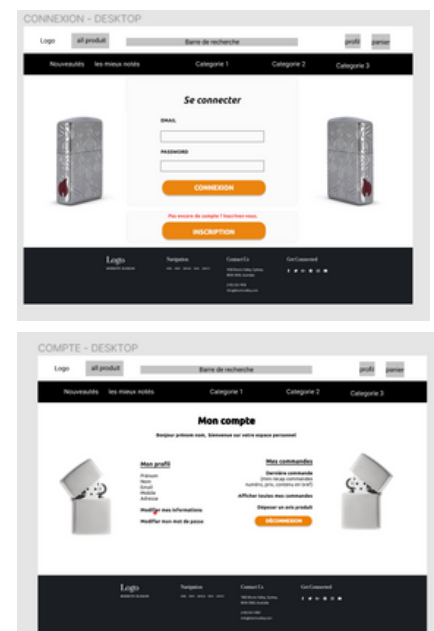
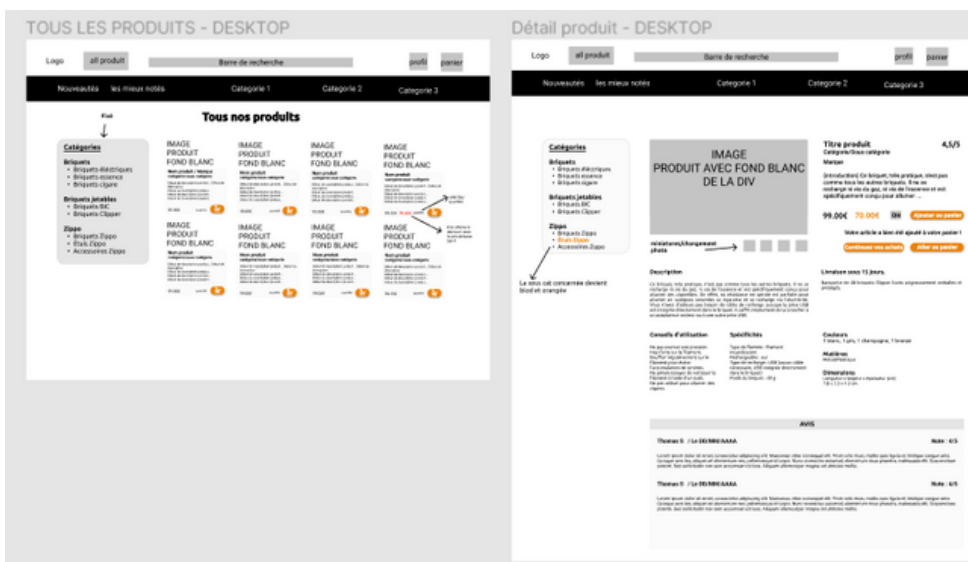
Le wireframe basse fidélité

Dans un premier temps, nous avons conçu le wireframe basse fidélité. En effet, nous avons conçu les pages sur Figma en formant les blocs principaux qui seront affichés sur les pages du site. Dans l'exemple ci-contre, on peut voir la page d'accueil avec son header, sa barre de navigation et les différents blocs que nous souhaitons y voir afficher pour rendre la page attractive pour l'utilisateur client.



Le wireframe haute fidélité

Dans un second temps, nous avons conçu le wireframe haute fidélité. Nous avons choisi un style épuré avec trois couleurs clés : blanc pour le fond, noir pour les fonts, la barre de navigation et le footer, orange pour les boutons. Nous avons sélectionné les fonts ubuntu pour les titres et roboto pour le corps de texte.



4. Développement de la partie front-end

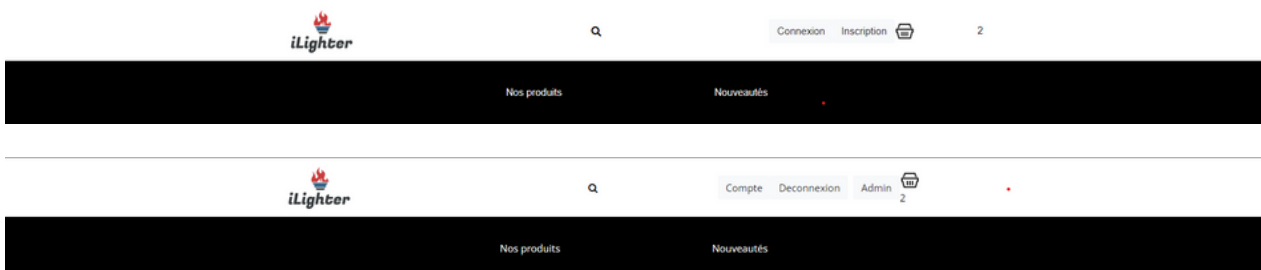
4.2 Le header et le footer

Le header et le footer ont été conçu avec Bootstrap 5.

Le header

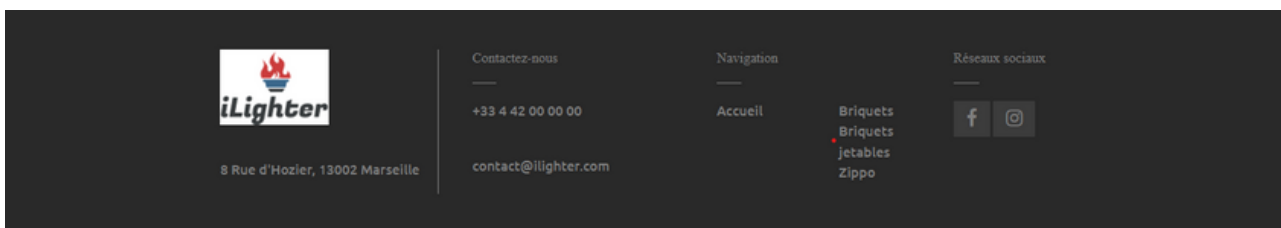
Le header et sa barre de navigation se compose de plusieurs parties :

- la partie supérieure avec le logo cliquable qui redirige l'utilisateur vers la page d'accueil, un bouton connexion redirigeant vers la page de connexion, un bouton inscription redirigeant vers la page d'inscription et un bouton panier redirigeant vers le panier. Lorsque des produits sont ajoutés au panier, le nombre d'articles s'affiche à côté de ce dernier. Lorsque l'utilisateur est connecté, le bouton connexion est transformé en bouton compte, redirigeant l'utilisateur vers son espace personnel.
- la partie inférieure où on retrouve un onglet "Nos produits" redirigeant l'utilisateur vers la page affichant tous les produits, et un bouton nouveautés, redirigeant l'utilisateur vers la page affichant les nouveaux produits



Le footer

Le footer se compose d'informations clés : le logo et l'adresse de la boutique, un premier bloc de contact, un second bloc de navigation et enfin un dernier bloc pour les réseaux sociaux.

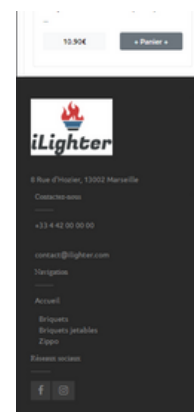
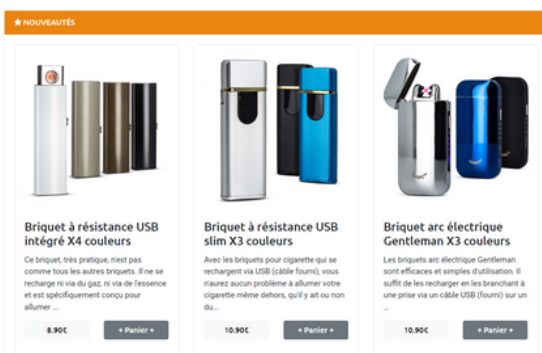
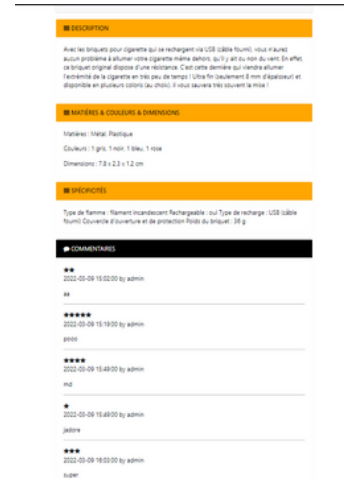
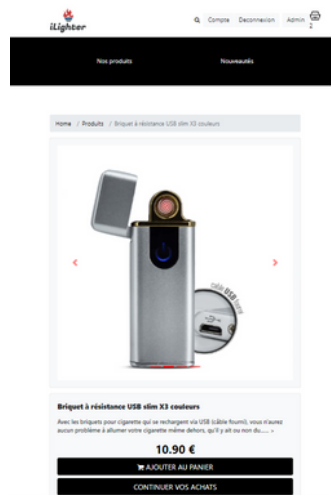


4. Développement de la partie front-end

4.3 Développement de la mise en page et du responsive

Bootstrap 5

Pour de nombreuses pages avons utilisé Bootstrap 5, un framework contenant toutes sortes de modèles de conception basés sur HTML et CSS pour diverses fonctions et composants tels que la navigation, le système de grille, les carrousels d'images et les boutons. Le principal avantage dans l'utilisation de Bootstrap réside dans le fait qu'il permet de créer des pages réactives et responsives, c'est-à-dire qu'il permet à l'interface utilisateur d'un site web de fonctionner de manière optimale sur toutes les tailles d'écran, que ce soit sur des téléphones à petit écran ou des ordinateurs de bureau à grand écran. L'import de Bootstrap se fait par l'ajout d'un lien dans les balises de métadonnées des pages de la view. Ensuite, il suffit de naviguer sur le site de Bootstrap et de consulter la documentation afin de récupérer les modèles HTML souhaités ayant des classes spécifiques à Bootstrap attribuées à leurs éléments qui vont systématiquement prendre le style développé par Bootstrap. Voici ci-dessous quelques exemples de pages :



4. Développement de la partie front-end

La méthode flexbox

Nous avons également utilisé la méthode flexbox sur certaines pages comme pour la page produits. C'est une méthode de mise en page selon un axe principal, permettant de disposer des éléments en ligne ou en colonne. Les éléments se dilatent ou se rétractent pour occuper l'espace disponible. Elle permet notamment de :

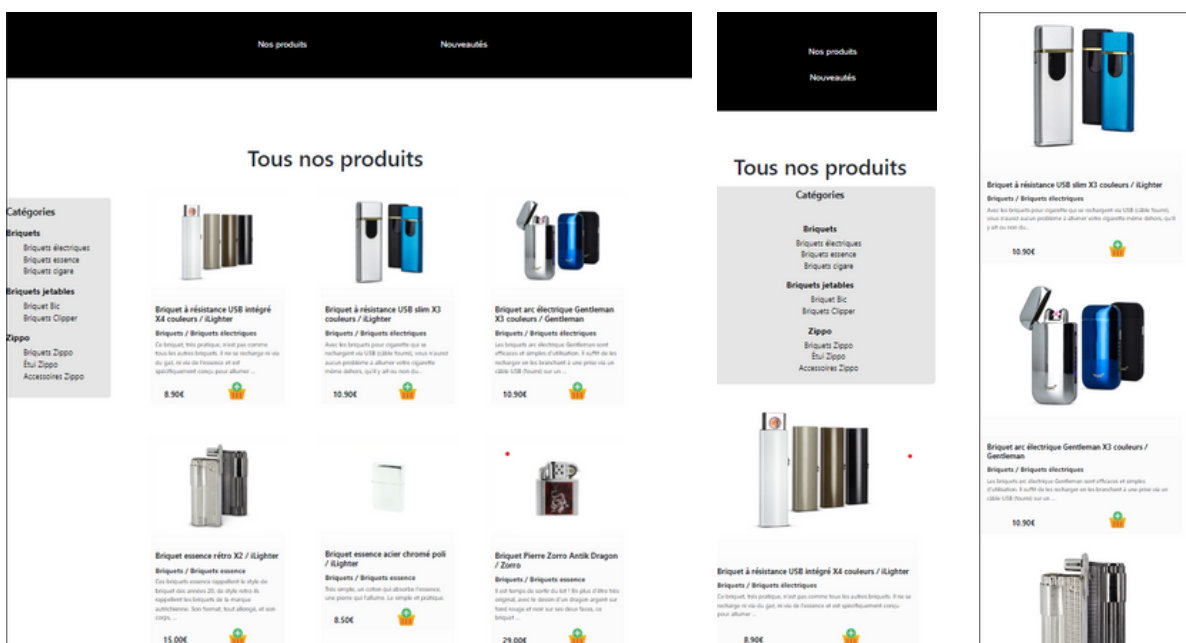
- centrer verticalement un bloc de contenu dans son parent
- faire que tous les enfants d'un conteneur occupent tous une même quantité de hauteur/largeur disponible selon l'espace offert
- faire que toutes les colonnes dans une disposition multi-colonnes aient la même hauteur même si leur quantité de contenu diffère

Les media queries

Pour les pages n'ayant pas été conçues avec Bootstrap, nous avons utilisé les media queries dans les feuilles de style CSS qui permettent d'attribuer un style au page en fonction de la taille de l'écran et notamment de sa largeur ou de sa résolution.

```
@media only screen and (max-width: 600px) {
  main {
    margin: 0;
    padding: 0;
  }
  .products_list {
    display: flex;
    flex-direction: column;
    justify-content: center;
    left: 0;
  }
}
```

Voici par exemple la page produits, conçue en méthode flexbox couplée aux media queries :



4. Développement de la partie front-end

4.1 Développement des affichages dynamiques

Les fonctions d'affichage

Dans les fichiers de classes du dossier Model relatives aux tables de la base de données, nous avons conçus des fonctions d'affichages avec des requêtes SQL qui retournent des tableaux d'informations et le contenu que nous souhaitons voir afficher. Ces fonctions peuvent ne prendre aucun paramètre, comme pour l'affichage de tous les produits, ou un paramètre comme la catégorie des produits pour trier l'affichage des produits par catégorie ou l'id de l'utilisateur pour lequel nous souhaitons afficher les informations personnelles et ses commandes.

```
//permet de selectionner tous les product et les renvoyer dans un tableau
public function FindAllArticle(): array{
    $query = $this->pdo->prepare("SELECT * FROM `products`");
    $query->setFetchMode(\PDO::FETCH_ASSOC);
    $query->execute();
    $articles=$query->fetchall();
    return $articles;
}
```

```
//permet d'afficher tous les produits trie par categories
public function DisplayAllProductsByCat($category) {
    $query = "SELECT products.id, image1, products.title, brand, products.id_category, products.id_sub_category,
introduction, price, discount, discount_available, stock, category, id_sub_category FROM products
INNER JOIN categories ON categories.id = products.id_category
WHERE categories.category = :category";
    $array_products = $this->pdo->prepare($query);
    $array_products->setFetchMode(\PDO::FETCH_ASSOC);
    $array_products->execute(['category'=>$category]);

    $all_products = $array_products->fetchAll();

    return $all_products ;
}
```

```
//retourne les infos de l'utilisateur choisis avec id
public function findInfoUser($id):array{
    $sql = "SELECT * FROM users WHERE id= :id"
    $query = $this->pdo->prepare($sql);
    $query->setFetchMode(\PDO::FETCH_ASSOC);
    $query->execute(['id' => $id]);
    $userInfos=$query->fetchall();
    return $userInfos;
}
```

```
//retourne un tableau des commandes d'un utilisateur par ordre décroissant
//utilisée pour afficher les commandes de l'utilisateur
public function UserOrders($id_user) {
    $query = "SELECT orders.id, orders.incl_tax_price, orders.date FROM orders
INNER JOIN users ON orders.id_user = users.id
WHERE orders.id_user = :id_user
ORDER BY orders.id DESC";
    $orders = $this->pdo->prepare($query);
    $orders->setFetchMode(\PDO::FETCH_ASSOC);
    $orders->execute(['id_user'=>$id_user]);

    $user_orders = $orders->fetchAll();

    return $user_orders;
}
```

4. Développement de la partie front-end

Rôle du fichier de traitement PHP du dossier Controllers dans les affichages dynamiques

Dans les fichiers de traitement PHP du dossier Controllers, nous appelons donc les fonctions d'affichage définies dans les classes. Pour ce faire, lorsque la fonction ne prend pas de paramètre comme pour afficher tous les produits, nous appelons simplement la fonction, sans oublier d'ajouter la variable de l'objet dans la fonction render en aval du fichier de traitement :

```
$all_products = $this->model->DisplayAllProducts();
```

Si la fonction prend un paramètre GET, et à l'aide d'une condition IF et de la fonction PHP `isset()` qui va vérifier que le paramètre GET existe bien dans l'URL, on récupère alors ce paramètre dans une variable qui va servir de paramètre à l'exécution de la fonction comme par exemple pour l'affichage des produits selon leur catégorie, en ajoutant toujours la variable de l'objet dans la fonction render en aval du fichier de traitement :

```
if(isset($_GET['category'])) {
    $category = htmlspecialchars($_GET['category']);
    $all_products = $this->model->DisplayAllProductsByCat($category);
}
```

De même pour l'affichage des informations d'un utilisateur dans son espace mon compte, on récupère l'id en variable de session `$_SESSION['userId']` généré lors de sa connexion afin d'exécuter la fonction de récupération des informations avec ce paramètre :

```
$email = $_SESSION['email'];
$id_user = $_SESSION['userId'];
$user_infos = $this->model->findAllInfoUser($email);

$model_order = new \Models\Order();
$user_orders = $model_order->UserOrders($id_user);
$pageTitle = "compte";
Render::render('users/compte', compact('pageTitle', 'email', 'user_infos', 'user_orders'));
```


4. Développement de la partie front-end

Affichage des données dans la view

Ainsi, le tableau retourné par le fichier de traitement PHP controller permet de générer l'affichage des données dans la page comme dans la page ci-dessous pour l'espace mon compte :

```
<main>
<h2>Mon Compte</h2>

<p>Bonjour <?php echo $user_infos[0]['firstname']; ?>, bienvenue sur votre espace personnel</p>
<div class="bigcontainer">


<div class="formcolumn1">
<h3>Mon profil</h3>
<ul>
<li><?php echo $user_infos[0]['firstname']; ?></li>
<li><?php echo $user_infos[0]['lastname']; ?></li>
<li><?php echo $user_infos[0]['email']; ?></li>
<li><?php echo $user_infos[0]['number']; ?></li>
<li><?php echo $user_infos[0]['address']; ?></li>
</ul>
<a href="updateinfos">Modifier mes informations</a>
<br>
<a href="updatepassword">Modifier mon mot de passe</a>
</div>

<div class="formcolumn2">
<h3>Mes commandes</h3>
<p>Dernière commande <br>
<?php if(empty($user_orders)) { echo "<a href='macommande?id=' . $user_orders[0]['id'] . '>Commande numéro " . $user_orders[0]['id'] . "<br> du " . $user_orders[0]['date'] . "<br> " . $user_orders[0]['incl_tax_price'] . " euros</a>"; } ?>
</p>
<br>
<?php echo "<a class='lien_pad' href='mescommande?id=' . $user_infos[0]['id'] . '>Afficher toutes mes commandes</a>"; ?>
</div>


</div>
</main>
```

Utilisation des boucles PHP

Nous avons utilisé pour certaines pages comme la page produits une boucle foreach qui va parcourir chaque élément du tableau et permet de générer l'affichage avec un code raccourci :

```
<div class="products_list">
<?php foreach ($all_products as $product) {
echo "
<div class='product'>
<div class='box_img'>
<a href='produit?id=' . $product['id'] . '>
<img class='p_image' src='\" . $product['image1'] . "></a>
</div>
<div class='box_texte'>
<div><h3>\" . $product['title'] . " / \" . $product['brand'] . "</h3></div>
<div class='p_category'>\" . $product['category'] . " / \" . $product['sub_category'] . "</div>
<div class='p_intro'>\" . $product['introduction'] . "</div>
<div class='price_cart'>
<div class='price'>\" . $product['price'] . "</div> ";
if($product['stock'] > 0) { echo "
<div><a href='addtocart?id=' . $product['id'] . "><img class='icone' src='\" . $product['image1'] . "> alt='add_to_cart_icon'></a></div> " ;
} else { echo "
<div>Produit indisponible</div> " ;
}
echo "
</div>
</div>
";
}
?>
</div>
```


5. *Parcours utilisateur : le panier*

La classe Cart (panier)

J'ai d'abord créé une classe Cart, qui hérite de la classe Model permettant la connexion à la base de données. J'y ai développé plusieurs fonctions :

- une fonction permettant de récupérer les infos du produit sélectionné à ajouter au panier.
- une fonction qui ajoute le produit au panier. Le panier est en fait une variable de session `$_SESSION['cart']` avec une entrée supplémentaire sous la forme de `$_SESSION['cart']['product_id]` avec l'id du produit choisi et prenant la valeur de la quantité associée. Si le produit existe dans le panier alors ce produit est ajouté avec une quantité de 1 au panier, sinon, le nombre du produit présent dans le panier est incrémenté
- une fonction qui retourne un tableau avec toutes les informations des produits contenus dans le panier
- une fonction permettant de supprimer un produit du panier
- une fonction permettant de supprimer le panier
- une fonction permettant de calculer le prix total du panier
- une fonction permettant de recalculer le prix total du panier après modification des quantités des produits
- une fonction permettant de compter le nombre de produits présents dans le panier

Ajout d'un produit au panier

Lorsqu'un utilisateur ajoute un produit au panier, il est systématiquement redirigé dans l'immédiat sur la page produits. Cependant, le bouton ajouter au panier renvoie en fait vers la page `addtocart`(ajouter au panier) avec en paramètre GET l'id du produit choisi dans l'URL. J'ai donc créé un fichier de traitement de la page `addtocart` qui vient récupérer l'id en GET présent dans l'URL en la sécurisant avec la fonction `htmlspecialchars()` et qui l'utilise en paramètre de la fonction permettant de l'ajouter au panier. Ce fichier de traitement vérifie également le stock disponible en appelant une fonction de vérification du stock définie dans la classe propre aux produits, avant de lancer l'ajout au panier.

5. Parcours utilisateur : le panier

```
//Récupère l'id le nom et le prix d'un produit pour le panier
public function GetProductId() {

    if (isset($_GET['id'])) {

        $query = "SELECT id, title, price FROM products WHERE id = :id";
        $product_info = $this->pdo->prepare($query);
        $product_info->setFetchMode(PDO::FETCH_OBJ);
        $product_info->execute(['id'=>$_GET['id']]);

        $product = $product_info->fetch();

        return $product;

    }

    else {

        die("Vous n'avez pas sélectionner de produit à ajouter au panier");

    }

}
```

```
//Ajoute un produit au panier dans la session panier
public function AddProduct($product_id) {

    if (isset($_SESSION['cart'][$product_id])) {
        $_SESSION['cart'][$product_id]++;
    }

    else {
        $_SESSION['cart'][$product_id] = 1;
    }

}
```

```
class AddToCart extends Controllers{

    protected $modelName = \Models\Cart::class;

    public function AddToCart() {

        session_start();

        $model = new \Models\Cart();

        $product = $this->model->GetProductId();

        if (empty($product)) {
            die("Ce produit n'existe pas.");
        }

        $ModelArticle = new \Models\Article();
        $id_product = $product[0]->id;
        $verif_stock = $ModelArticle->ProductStock($id_product);
        if ($verif_stock > 0){
            $cart = $this->model->AddProduct($product[0]->id);
            Http::redirect("produits");
        }else {
        }

        // Http::redirect("panier");

        $pageTitle = "Ajouter_au_panier";

        Renderer::render('articles/addtocart', compact('pageTitle','model','product'));

    }

}
```

5. Parcours utilisateur : le panier

Affichage du panier

L'utilisateur peut accéder à son panier en cliquant sur l'icône du panier dans le header. La page panier a également un fichier de traitement PHP qui appelle la fonction qui permet d'afficher l'ensemble des informations des produits contenus dans le panier ainsi que la fonction qui calcule le prix total.

```
//Retourne un tableau pour ficher les produits dans le panier
public function ProductsInCart() {

    if(isset($_SESSION['cart']) && !isset($_GET['del'])) { // vérifie l'absence de suppression pour éviter l'affichage d'erreur PHP

        $ids = array_keys($_SESSION['cart']);
        $separator = ",";

        if (empty($ids)) {
            $products = array();
        }

        else {
            $query = "SELECT id, title, price, image1, stock FROM products WHERE id IN (".implode($separator, $ids).")";
            $products_cart = $this->pdo->prepare($query);
            $products_cart->setFetchMode(PDO::FETCH_OBJ);
            $products_cart->execute();

            $products = $products_cart->fetchAll();
        }

        return $products;
    }
}
```

```
//Retourne le prix total du panier
public function TotalPrice () {

    $total = 0;
    if(isset($_SESSION['cart'])){
        $ids = array_keys($_SESSION['cart']);
        $separator = ",";
    }

    if (empty($ids)) {
        $products = array();
    }

    else {
        $query = "SELECT id, price FROM products WHERE id IN (".implode($separator, $ids).")";
        $products_cart = $this->pdo->prepare($query);
        $products_cart->setFetchMode(PDO::FETCH_OBJ);
        $products_cart->execute();

        $products = $products_cart->fetchAll();

        foreach($products as $product) {
            $total += $product->price * $_SESSION['cart'][$product->id];
        }

        return $total;
    }
}
```

```
class Panier extends Controllers{

    protected $modelName = \Models\Cart::class;

    public function Panier(){

        $model = new \Models\Cart();







        session_start();

        $products = $this->model->ProductsInCart();

        $total_price = $this->model->TotalPrice();
    }
}
```

5. Parcours utilisateur : le panier

VOTRE PANIER

Article	Disponibilité	Quantité	Prix Hors Taxes	
 Briquet à résistance USB slim X3 couleurs	En stock	<input type="text" value="1"/>	10.90€	
 Briquet arc électrique Gentleman X3 couleurs	En stock	<input type="text" value="1"/>	10.90€	
 Briquet Pierre Zorro Antik Dragon	En stock	<input type="text" value="1"/>	29.00€	

Actualisez votre panier pour enregistrer les modifications de quantités ! [Actualiser le panier](#)

[Vider le panier](#)

Sous total HT	50.80€
Livraison HT	7.90€
Total HT	58.70€
TVA 20%	11.74€
Total TTC	70.44€

[CONTINUER VOS ACHATS](#)
[VALIDER LE PANIER](#)

Modification du panier

L'utilisateur peut modifier le contenu de son panier : soit en supprimant un ou plusieurs articles en cliquant sur l'icône de la poubelle qui va ajouter en paramètre dans l'URL le paramètre GET['del'] égal à la valeur de l'id du produit à supprimer, soit en modifiant les quantités des produits et en cliquant sur actualiser le panier. Le fichier de traitement PHP fera alors appel soit à la fonction qui supprime le produit du panier en cliquant sur l'icône de la poubelle, soit à la fonction qui va recalculer le prix total du panier puis actualiser la page du panier.

```
//Supprime un article du panier
public function DeleteProduct ($product_id) {

    unset($_SESSION['cart'][$product_id]);
    header('Refresh: 0; url=panier');

}
```





```
if (isset($_GET['del'])) {
    $delete = $this->model->DeleteProduct($_GET['del']);
}
```

5. Parcours utilisateur : le panier

```
//Met à jour les quantités/prix des articles dans le panier
public function Recalculate() {
    foreach($_SESSION['cart'] as $product_id => $quantity ) {
        $_SESSION['cart'][$product_id] = $_POST['cart'][$product_id]['quantity'];
    }
}
```

```
if (isset($_POST['cart'][$product_id]['quantity'])) {
    $recalculate = $this->model->Recalculate();
    header('Location: panier');
}
```

VOTRE PANIER

Article	Disponibilité	Quantité	Prix Hors Taxes	
 Briquet à résistance USB slim X3 couleurs	En stock	<input type="text" value="1"/>	10.90€	
 Briquet Pierre Zorro Antik Dragon	En stock	<input type="text" value="4"/>	116.00€	
Actualisez votre panier pour enregistrer les modifications de quantités !				<input type="button" value="Actualiser le panier"/>
				<input type="button" value="Vider le panier"/>
			Sous total HT	126.90€
			Livraison HT	7.90€
			Total HT	134.80€
			TVA 20%	26.96€
			Total TTC	161.76€

CONTINUER VOS ACHATS

VALIDER LE PANIER

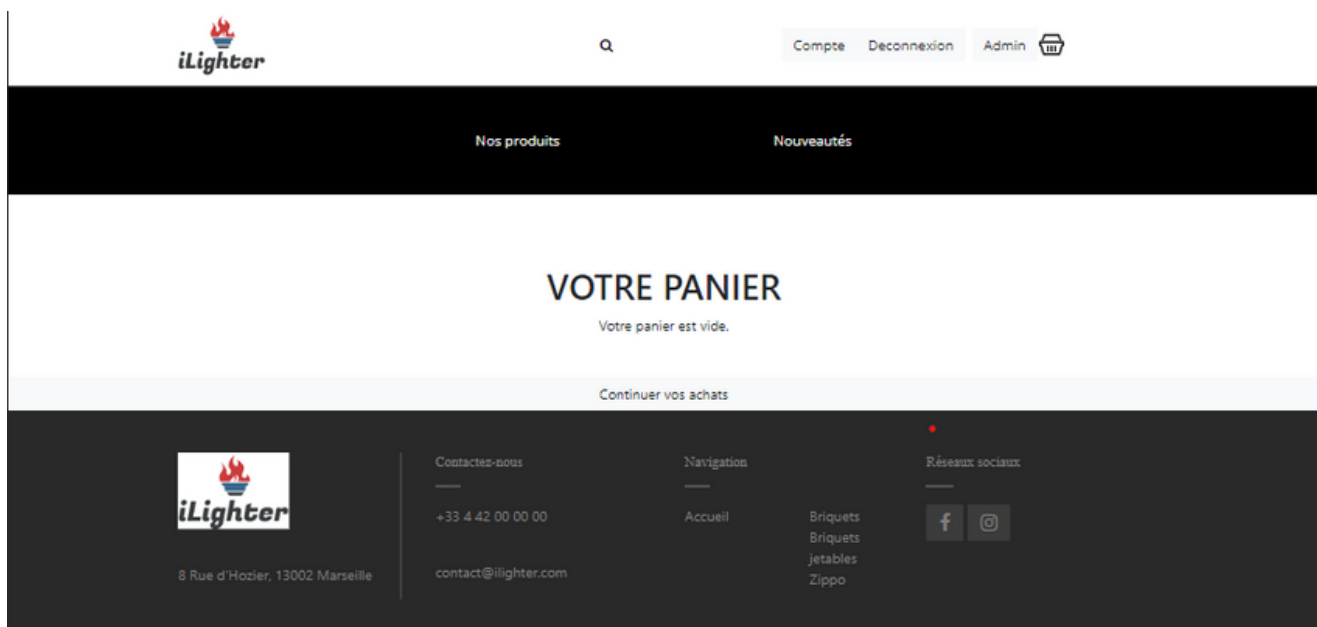
5. Parcours utilisateur : le panier

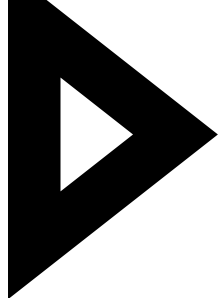
Suppression du panier

L'utilisateur peut supprimer son panier en cliquant sur le bouton vider le panier qui va le faire transiter sur une page qui va faire appel à la fonction supprimant la variable de session du panier dans le fichier de traitement PHP.

```
//Supprimer le panier
public function DeleteCart () {
    unset($_SESSION['cart']);
}
```

```
if (isset($_GET['DelCart'])) {
    $delete_cart = $this->model->DeleteCart();
    Http::redirect("panier");
}
```





6. Veille et recherche documentaire

6.1 Veille sur les failles de sécurité en anglais

Pour me documenter sur la prévention des failles de sécurité les plus connues, j'ai effectué des recherches en anglais sur les injections SQL sur le site StackOverflow.com et son forum, qui préconise de développer son site web en programmation orientée objet et donc d'utiliser des requêtes SQL préparées dans nos fonctions PHP.

6.2 Extrait et traduction de la documentation en anglais

Explanation

The SQL statement you pass to `prepare` is parsed and compiled by the database server. By specifying parameters (either a `?` or a named parameter like `:name` in the example above) you tell the database engine where you want to filter on. Then when you call `execute`, the prepared statement is combined with the parameter values you specify.

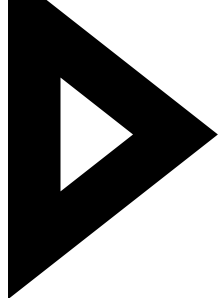
The important thing here is that the parameter values are combined with the compiled statement, not an SQL string. SQL injection works by tricking the script into including malicious strings when it creates SQL to send to the database. So by sending the actual SQL separately from the parameters, you limit the risk of ending up with something you didn't intend.

Any parameters you send when using a prepared statement will just be treated as strings (although the database engine may do some optimization so parameters may end up as numbers too, of course). In the example above, if the `$name` variable contains `'Sarah'; DELETE FROM employees` the result would simply be a search for the string `"'Sarah'; DELETE FROM employees"`, and you will not end up with [an empty table](#).

Another benefit of using prepared statements is that if you execute the same statement many times in the same session it will only be parsed and compiled once, giving you some speed gains.

Oh, and since you asked about how to do it for an insert, here's an example (using PDO):

```
$preparedStatement = $db->prepare('INSERT INTO table (column) VALUES (:column)');  
$preparedStatement->execute([ 'column' => $unsafeValue ]);
```



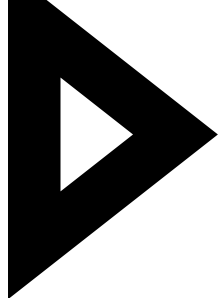
Explication

La requête SQL transmise dans le `prepare()` est analysée et compilée par le serveur de la base de données. En spécifiant des paramètres (soit par `?` ou alors en nommant le paramètre de la façon suivante `:name` dans l'exemple ci-dessus) vous indiquez au moteur de stockage de la base de données où le moteur de stockage de la base de données doit filtrer les données entrantes. Ensuite, quand vous appelez l'exécution de la fonction, la requête SQL est combinée aux paramètres que vous avez spécifiés.

L'important ici est que les valeurs des paramètres soient combinées avec la requête compilée et non avec une chaîne de caractères SQL. Une injection SQL fonctionne en trompant le script de la page en incluant des chaînes de caractères malveillantes à envoyer à la base de données. Alors, en envoyant la requête SQL et les paramètres d'exécution séparément, vous limitez le risque qu'une action que vous n'aviez pas prévu soit effectuée sur la base de données.

N'importe quel paramètre envoyé en utilisant une requête préparée sera seulement traité comme une chaîne de caractère(bien que le moteur de stockage de la base de données puisse aussi effectuer des optimisations pour que les paramètres finissent également sous forme de nombre). Dans l'exemple ci-dessus, si la variable `$name` contient `'Sarah'` ; `DELETE FROM employees` et le résultat sera simplement une recherche de la chaîne de caractère `""Sarah"; DELETE FROM employees"` et l'employée Sarah ne sera pas supprimée de la base de données.

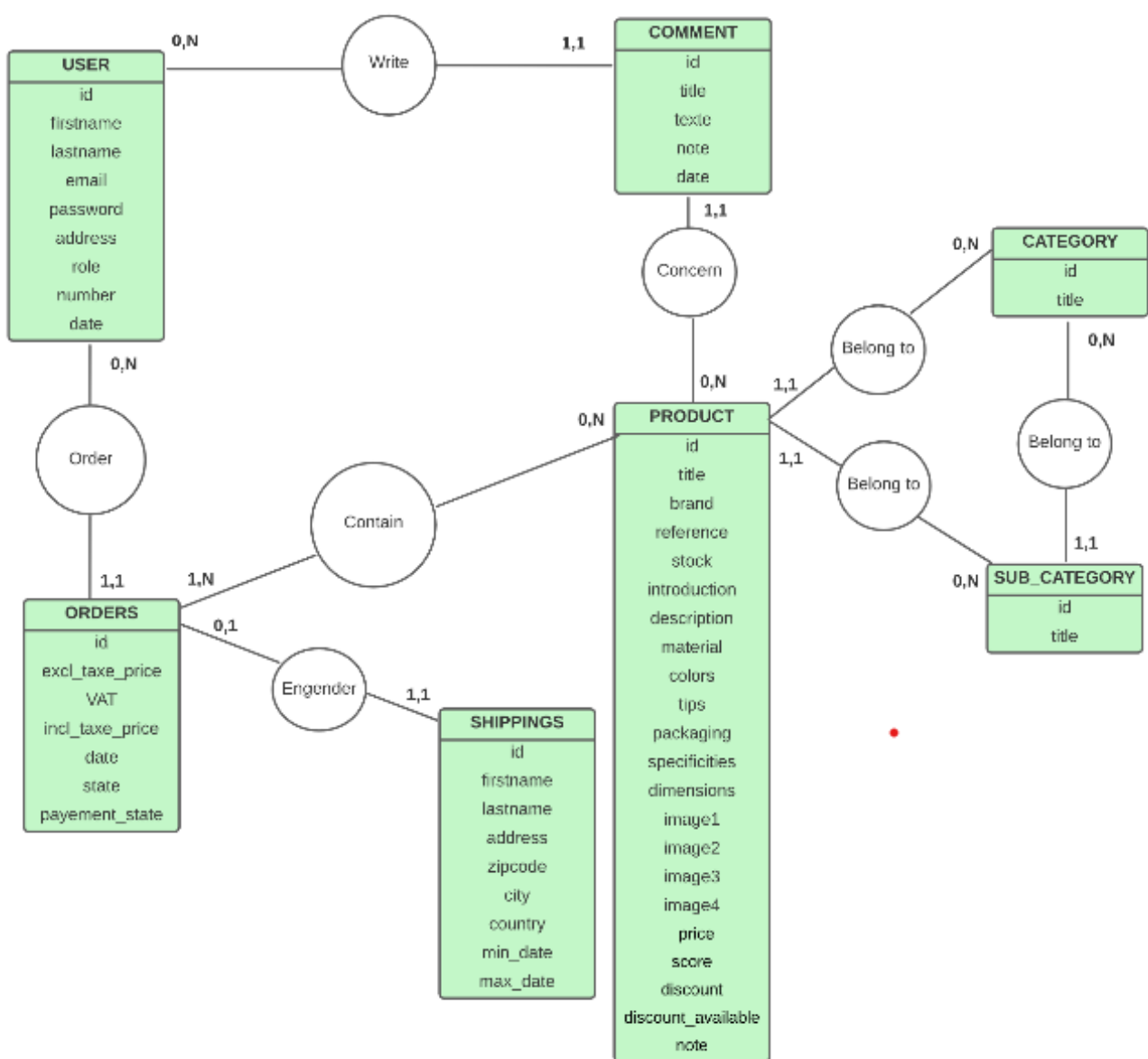
Un autre avantage d'utiliser les requêtes préparées est que si vous exécutez la même requête plusieurs fois dans la même session, elle ne sera analysée et compilée qu'une fois, ce qui vous donnera un gain de vitesse de traitement.



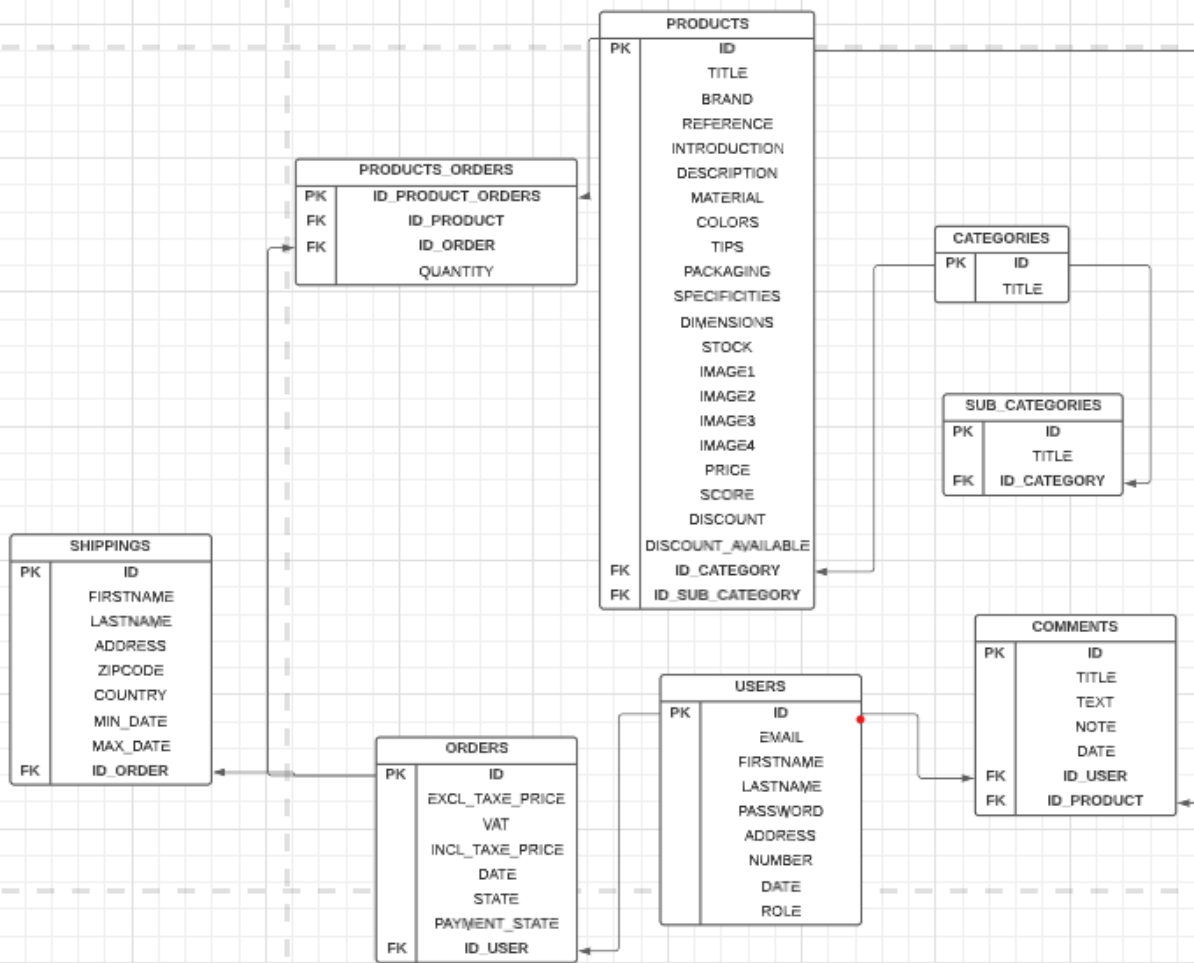
En conclusion, le développement de notre boutique en ligne iLighter nous aura permis de mettre en pratique toutes les compétences techniques assimilées au cours de l'année, qu'il s'agisse des tâches à réaliser en amont comme la modélisation de la base de données et le maquettage du site web ou des tâches de développement pures front-end et back-end, tout en respectant les mesures de sécurités essentielles..

Egalement, nous avons pu développé nos compétences transversales comme le travail d'équipe, l'organisation, la communication, l'entraide, et nous avons pu écarter de nombreuses difficultés en se documentant sur des sites web divers en anglais ou en français, et surtout en échangeant avec nos autres camarades.

Annexe 1 : Modèle Conceptuel des Données



Annexe 2 : Modèle Logique des Données





Annexe 3 : Vue relationnelle du concepteur PHPMYADMIN

