

Rapport Détail du Projet

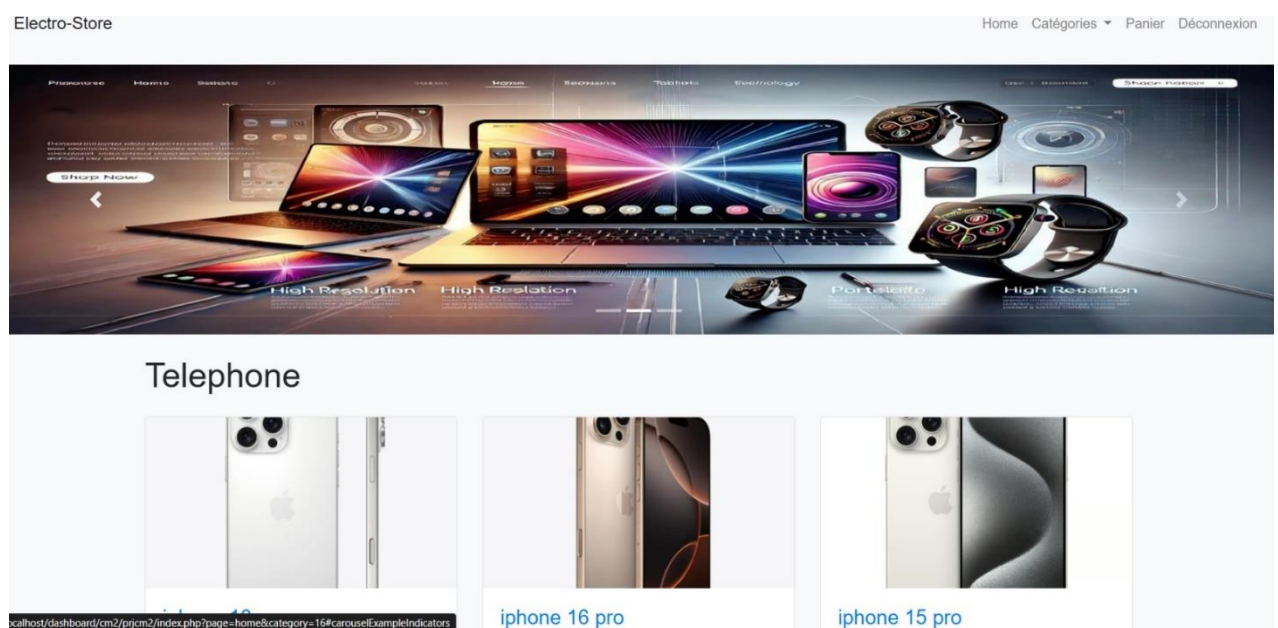
Travail de deux personnes :

- Mohamed amine Sdiri
- Lyna Mazguene

Introduction

Dans ce document, nous, en tant que binôme, avons pour objectif de fournir une explication détaillée des fonctionnalités de notre application web. En développant ce projet, nous avons appliqué l'architecture Modèle-Vue-Contrôleur (MVC) pour organiser les différentes parties de notre site de manière logique et structurée. Chaque composant de l'application – fichiers de configuration, contrôleurs, modèles, vues, fichiers d'administration, et fichiers d'inclusion – a été conçu avec soin pour répondre à des besoins spécifiques tout en assurant la flexibilité et la facilité de maintenance du projet.

Nous examinerons chaque composant de l'application, expliquant comment il interagit avec les autres, quelles sont ses responsabilités et comment il contribue à l'expérience utilisateur globale. Chaque section de ce document sera dédiée à un aspect particulier de l'application, détaillant les fonctionnalités spécifiques, les interactions avec les autres composants, et les meilleures pratiques que nous avons suivies pour garantir la qualité et la sécurité du projet. Ce document servira de guide complet pour comprendre l'architecture et les principes sur lesquels repose notre application, facilitant ainsi la maintenance future et les mises à jour nécessaires.

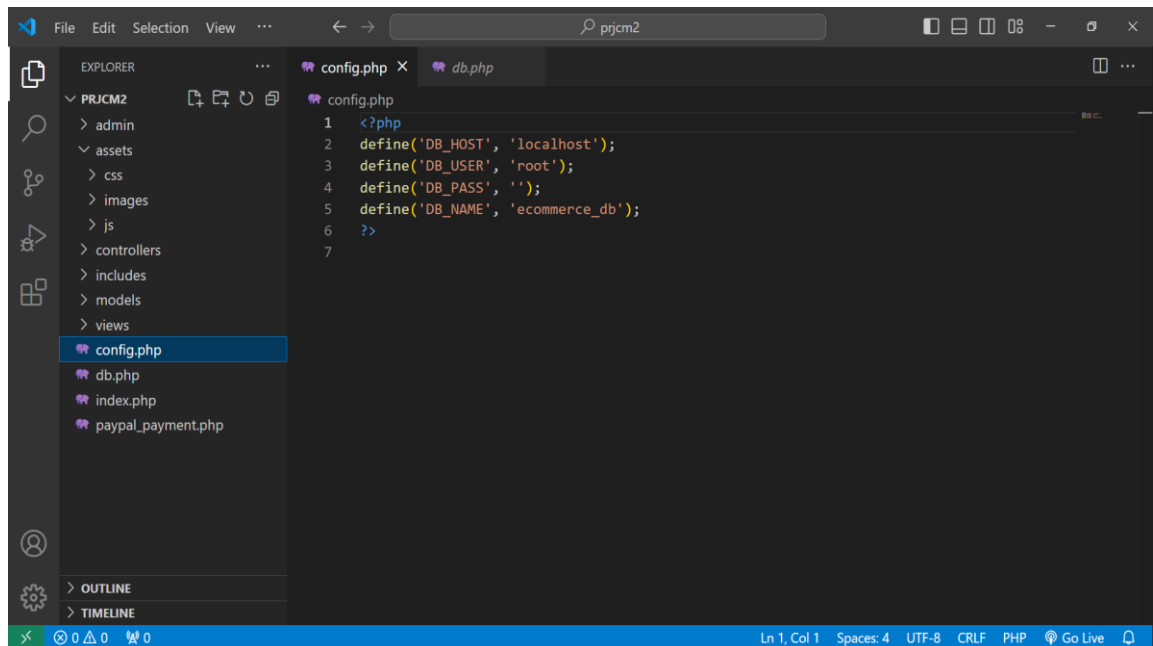


- **Fonctionnalités principales de config.php**

1. **Centralisation des paramètres globaux** : Le fichier config.php regroupe toutes les constantes globales utilisées dans l'application, y compris celles qui se rapportent aux connexions à la base de données, aux chemins d'accès aux fichiers, aux URL de l'application, aux messages d'erreur, et à d'autres configurations spécifiques. Cela permet une gestion centralisée des paramètres, rendant l'application plus facile à maintenir. En cas de changement dans l'environnement de déploiement (par exemple, passage de l'environnement de développement à la production), il suffit de modifier un seul fichier plutôt que de répéter des changements dans tout le code applicatif.
2. **Gestion des environnements de l'application** : config.php permet la gestion des différents environnements dans lesquels l'application peut être déployée (par exemple, développement, test, production). Il définit des constantes pour chaque environnement, ce qui permet de séparer clairement les paramètres spécifiques de chaque contexte. Par exemple, une constante peut déterminer si les erreurs seront affichées à l'utilisateur ou stockées dans un journal pour une vérification ultérieure, ce qui est particulièrement utile pour la phase de développement où les erreurs doivent être traitées rapidement.
3. **Constantes pour les connexions aux bases de données** : Une section cruciale de config.php est dédiée aux constantes relatives à la base de données. Cela comprend les informations d'hôte, de port, de nom d'utilisateur, de mot de passe et de nom de la base de données. Ces constantes centralisées permettent à l'application de se connecter à la base de données en utilisant un point de référence unique. Si l'hôte ou les informations d'identification changent, les modifications sont effectuées uniquement dans config.php, ce qui diminue les risques d'erreurs dues à la propagation de mauvaises informations.
4. **Chemins d'accès** : config.php définit également des constantes pour les chemins d'accès aux fichiers (par exemple, les répertoires de stockage des images, des fichiers temporaires, des logs). Cela simplifie le déplacement de l'application d'un serveur à un autre ou d'un environnement de test à un environnement de production. Ces chemins sont utilisés pour toutes les opérations de lecture/écriture, assurant ainsi que l'application sait exactement où rechercher les fichiers nécessaires ou où écrire les nouvelles données.
5. **URL de l'application** : Les URL de l'application (l'URL de base, les chemins vers les pages spécifiques) sont définies dans config.php. Cela permet de modifier facilement l'URL de l'application sans nécessiter de changements dans les chemins relatifs ou absolus du code. Cela est crucial pour le SEO, la détection des redirections, et le bon fonctionnement des fonctionnalités de l'application telles que les liens de partage ou les notifications par e-mail.
6. **Paramètres spécifiques à l'application** : En plus des paramètres de base (comme les informations de connexion à la base de données et les chemins d'accès), config.php peut également définir des constantes spécifiques à l'application, telles que le nombre d'articles par page, les délais d'expiration des sessions, les types de contenu acceptés, etc. Ces constantes permettent de garder les paramètres de l'application accessibles et modifiables sans toucher aux composants plus critiques du code applicatif.
7. **Gestion des erreurs et des messages** : config.php joue également un rôle clé dans la gestion des erreurs. Les constantes peuvent définir des messages d'erreur génériques à afficher aux

utilisateurs finaux en cas de problème, ainsi que des messages plus détaillés destinés aux développeurs ou aux administrateurs via des fichiers de journalisation. Cela permet une meilleure communication lors du diagnostic des problèmes et améliore la qualité du service fourni aux utilisateurs.

8. **Optimisation des performances** : Parfois, config.php inclut des constantes relatives à l'optimisation des performances, comme la définition de la cache, les valeurs de session, ou la configuration des logs. Cela peut inclure la configuration de la taille de la mémoire tampon ou le nombre de requêtes par page pour éviter des surchargeurs d'appels ou améliorer le temps de réponse de l'application.



En conclusion, config.php est plus qu'un simple fichier de configuration ; il est le cœur de l'application, garantissant la cohérence, la flexibilité et la sécurité dans toutes les interactions avec l'environnement d'exécution. Il joue un rôle crucial dans le processus de développement en facilitant la transition entre différents environnements, en simplifiant les processus de déploiement et en assurant une maintenance simplifiée à long terme.

- **Rôle du fichier db.php dans l'application**

1. **Établissement de la connexion à la base de données :**

Le fichier db.php sert principalement à initier une connexion entre le serveur web et la base de données. Cette étape est critique, car presque toutes les opérations de l'application (insertion, lecture, mise à jour, suppression) reposent sur l'interaction avec les données. Sans une connexion stable et correctement configurée, l'application ne peut pas fonctionner.

2. **Gestion des erreurs de connexion :**

Si une erreur se produit lors de la tentative de connexion (par exemple, des informations d'identification incorrectes ou un serveur de base de données injoignable), db.php joue un rôle essentiel en détectant ces erreurs et en arrêtant l'exécution pour éviter des comportements imprévisibles. De plus, il peut fournir des messages d'erreur clairs qui facilitent le diagnostic et la résolution des problèmes.

3. **Encapsulation de la logique de connexion :**

Plutôt que de répéter la configuration de connexion dans différents fichiers du projet, db.php centralise cette logique en un seul endroit. Cela améliore la maintenabilité : si des modifications doivent être apportées (par exemple, un changement d'hôte ou de nom d'utilisateur), elles peuvent être effectuées dans un seul fichier, ce qui réduit les risques d'erreurs.

4. **Abstraction pour les interactions avec la base de données :**

En plus d'établir la connexion, db.php peut offrir une interface simplifiée pour exécuter des requêtes SQL, gérant automatiquement des tâches complexes comme la gestion des erreurs ou la prévention des injections SQL.

• **Principales fonctionnalités de db.php**

1. **Configuration centralisée des paramètres de connexion :**

- **Hôte de la base de données :** Indique l'adresse du serveur, qui peut être localhost en développement ou une adresse distante en production.
- **Nom d'utilisateur et mot de passe :** Utilisés pour authentifier la connexion.
- **Nom de la base de données :** Spécifie la base que l'application utilisera.
- **Port :** Par défaut, les bases de données MySQL utilisent le port 3306, mais cela peut varier en fonction des configurations.

Ces informations sont souvent récupérées depuis le fichier config.php ou définies directement dans db.php. Cela garantit une cohérence dans les paramètres utilisés.

2. **Sécurité de la connexion :**

Le fichier veille à protéger les informations sensibles en utilisant des pratiques telles que l'encodage des identifiants ou leur stockage sécurisé. Par exemple :

- Utilisation de connexions sécurisées (SSL) pour les bases de données externes.
- Gestion stricte des exceptions pour éviter la divulgation d'informations sensibles dans les messages d'erreur.

3. **Support des bibliothèques modernes (mysqli ou PDO) :**

- **PDO (PHP Data Objects)** est souvent préféré pour sa flexibilité et ses fonctionnalités avancées, comme la prise en charge de plusieurs types de bases de données (MySQL, PostgreSQL, etc.) et la gestion des requêtes préparées pour protéger contre les injections SQL.
- **mysqli** est utilisé spécifiquement pour les bases de données MySQL et offre des fonctionnalités similaires, bien que moins étendues.

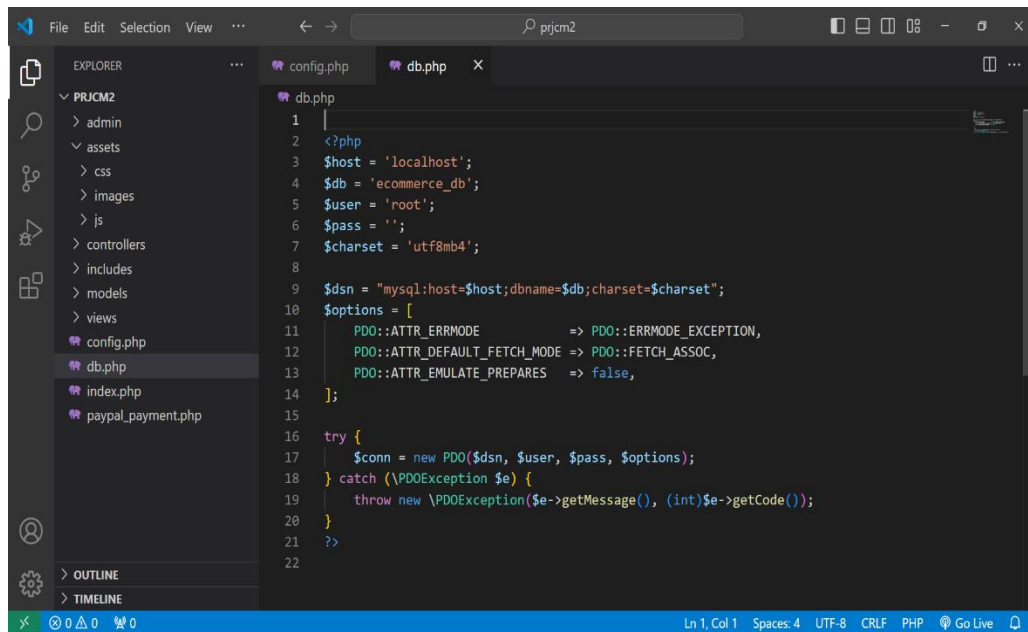
4. **Gestion des connexions persistantes :**

Une connexion persistante peut être utilisée pour améliorer les performances en réduisant le temps nécessaire pour établir une connexion à chaque requête. db.php peut inclure cette optimisation tout en s'assurant que les connexions ne sont pas surutilisées ou mal fermées.

5. Gestion des erreurs et des exceptions :

En cas d'échec de connexion, db.php doit non seulement arrêter l'exécution du programme, mais aussi fournir des messages informatifs à l'administrateur. Par exemple :

- Un message générique pour l'utilisateur final (« Une erreur technique est survenue. ») afin d'éviter toute fuite d'informations sensibles.
- Un journal d'erreurs détaillé pour le développeur, consignnant les détails comme le code d'erreur et le moment de l'incident.



```
1 |
2 | <?php
3 | $host = 'localhost';
4 | $db = 'ecommerce_db';
5 | $user = 'root';
6 | $pass = '';
7 | $charset = 'utf8mb4';
8 |
9 | $dsn = "mysql:host=$host;dbname=$db;charset=$charset";
10 | $options = [
11 |     PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
12 |     PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
13 |     PDO::ATTR_EMULATE_PREPARES => false,
14 | ];
15 |
16 | try {
17 |     $conn = new PDO($dsn, $user, $pass, $options);
18 | } catch (\PDOException $e) {
19 |     throw new \PDOException($e->getMessage(), (int)$e->getCode());
20 | }
21 |
22 |
```

En résumé, **db.php** n'est pas simplement un fichier de connexion : il représente un pont vital entre l'application et ses données. Il garantit que cette interaction est sécurisée, fiable et facile à gérer, tout en offrant une base flexible pour les futures évolutions ou optimisations de l'application.

• Contrôleurs (Controllers)

CategoryController.php : Le contrôleur des catégories, CategoryController.php, est essentiel pour la gestion des catégories de produits dans l'application. Ce fichier agit comme un intermédiaire entre l'utilisateur et la logique métier, facilitant la navigation dans les différentes catégories disponibles sur le site. Le contrôleur reçoit les requêtes des utilisateurs, telles que la demande d'affichage d'une catégorie spécifique ou la modification d'une catégorie existante, puis applique les règles métier nécessaires avant de transmettre les données aux vues appropriées pour l'affichage.

Le rôle principal de CategoryController.php est de gérer toutes les actions liées aux catégories. Cela inclut l'affichage des catégories disponibles, la création de nouvelles catégories, la modification des catégories existantes et la suppression des catégories. Lorsqu'une catégorie est créée, le contrôleur s'assure que toutes les données nécessaires, telles que le nom de la catégorie, la description, et les autres attributs pertinents, sont validées selon les règles définies dans l'application (par exemple, longueur maximale du nom, caractères autorisés, etc.). Le contrôleur utilise les modèles pour interagir avec la base de données, enregistrer les nouvelles catégories ou mettre à jour les informations existantes. En outre, il applique des contrôles d'accès pour s'assurer que seules les

personnes autorisées peuvent effectuer des modifications sur les catégories, protégeant ainsi l'intégrité des données et garantissant une gestion sécurisée des informations sensibles.

En outre, `CategoryController.php` est responsable de la gestion des relations entre les catégories et les produits. Lorsqu'un produit est ajouté à une catégorie, le contrôleur met à jour les informations correspondantes dans la base de données, en s'assurant que la relation est correctement établie et que toutes les données nécessaires sont synchronisées. De même, lorsqu'un produit est retiré d'une catégorie, le contrôleur effectue les opérations nécessaires pour que la base de données reflète les changements. Cette gestion des relations est cruciale pour l'affichage des produits par catégorie dans les pages de navigation et pour l'optimisation des performances du site, en limitant le nombre de requêtes nécessaires pour afficher les informations de produit pertinentes.

OrderController.php : Le fichier `OrderController.php` est au cœur du processus de gestion des commandes dans l'application. Il coordonne toutes les interactions liées à la commande, de la création initiale à la validation et à la gestion des commandes existantes. Le rôle principal du contrôleur des commandes est de garantir que toutes les étapes du processus de commande, telles que l'ajout de produits au panier, la validation du panier, la création d'une commande, le traitement du paiement, et la gestion des livraisons, sont exécutées correctement et dans le bon ordre. Il joue également un rôle crucial dans l'expérience utilisateur en assurant une interface intuitive et fluide pour l'utilisateur, minimisant les frictions tout au long du parcours d'achat.

Le `OrderController.php` reçoit les requêtes de l'utilisateur pour la création ou la modification des commandes. Lorsqu'une commande est créée, le contrôleur s'assure que toutes les informations nécessaires sont validées, comme les détails de la livraison, les informations de paiement, et les produits sélectionnés. Ce processus inclut la validation des stocks disponibles pour chaque produit afin de s'assurer que la commande peut être honorée. En cas d'incohérence ou de problème avec les données fournies, le contrôleur retourne des erreurs explicites à l'utilisateur, minimisant ainsi les risques de processus de commande incorrects.

Une autre responsabilité clé du `OrderController.php` est la gestion des états des commandes. Il est chargé de mettre à jour les statuts des commandes, comme "en attente", "validée", "expédiée", "livrée" ou "annulée". Le contrôleur interagit avec la base de données pour enregistrer ces modifications et s'assurer que les informations sont visibles dans les tableaux de bord des utilisateurs et des administrateurs. En cas de conflit ou de modification requise, le contrôleur s'assure que toutes les étapes intermédiaires sont correctement suivies, en mettant à jour les états de la commande, ajustant les informations de stock en conséquence, et en alertant les utilisateurs concernés par les changements.

ProductController.php : Le fichier `ProductController.php` est chargé de la gestion complète des produits dans l'application. Il est responsable de l'affichage des informations relatives aux produits, de leur mise à jour, de leur ajout ou de leur suppression. Le contrôleur des produits garantit que les informations sur les produits affichées aux utilisateurs sont toujours à jour et exactes, minimisant ainsi les incohérences ou les erreurs. Il facilite également les recherches, le filtrage et le tri des produits, optimisant ainsi l'expérience utilisateur en permettant un accès rapide aux informations pertinentes.

Le `ProductController.php` interagit principalement avec les modèles de produits pour récupérer les données, effectuer des mises à jour ou enregistrer de nouveaux produits dans la base de données. Il reçoit les requêtes des utilisateurs pour afficher les produits, rechercher par nom, catégorie ou autre critère, et retourne les données formatées aux vues pour le rendu. Le contrôleur applique des règles métier pour garantir que les données saisies, comme le prix, la description, les stocks disponibles, et

les images, sont valides avant d'être ajoutées à la base de données. En outre, le `ProductController.php` est responsable de la gestion des images associées aux produits, de la validation des formats acceptés (comme JPEG, PNG), et de la gestion des chemins de stockage des images dans l'application.

La suppression d'un produit est également une tâche gérée par le `ProductController.php`. Lorsqu'un produit est supprimé, le contrôleur s'assure que toutes les données associées, telles que les images, les avis des utilisateurs, et les informations de stock, sont également supprimées ou désactivées. Cela préserve l'intégrité des données et empêche les utilisateurs de visualiser des informations incorrectes ou obsolètes. Le contrôleur s'assure également que l'action de suppression est irréversible, protégeant ainsi la base de données contre les actions accidentelles ou les manipulations involontaires des données.

UserController.php : Le fichier `UserController.php` est dédié à la gestion des utilisateurs dans l'application. Il est chargé de traiter toutes les opérations liées aux utilisateurs, telles que l'enregistrement, la connexion, la gestion des profils, et le déconnexion. Le contrôleur utilisateur est responsable de l'application des règles de validation des données pour chaque action utilisateur, garantissant ainsi la sécurité et la cohérence des informations de l'utilisateur stockées dans la base de données. En agissant comme un intermédiaire entre l'interface utilisateur et les modèles, le `UserController.php` assure une interaction fluide et sécurisée pour l'utilisateur final.

L'enregistrement des utilisateurs est l'une des premières fonctions du `UserController.php`. Lorsqu'un nouvel utilisateur s'inscrit, le contrôleur valide les informations d'entrée, telles que le nom d'utilisateur, l'adresse e-mail, le mot de passe et d'autres informations personnelles. Il applique les règles de sécurité pour garantir que les mots de passe sont correctement hachés et que les informations sensibles ne sont pas stockées en texte clair. Une fois les données validées, le contrôleur crée un nouvel utilisateur dans la base de données et initialise un profil utilisateur avec les informations nécessaires pour les interactions futures avec l'application.

Pour la connexion, le `UserController.php` vérifie que les informations d'utilisateur sont correctes avant de permettre l'accès aux fonctionnalités de l'application réservées aux utilisateurs authentifiés. Il utilise les informations de la base de données pour vérifier le mot de passe, s'assure que le compte est actif, et redirige les utilisateurs vers des pages spécifiques en fonction de leur rôle ou de leurs permissions. Lorsqu'un utilisateur est connecté, le contrôleur démarre une session et stocke les informations de session en toute sécurité, garantissant que l'utilisateur reste authentifié pendant toute la durée de la session.

La gestion des profils utilisateur est également sous la responsabilité du `UserController.php`. Ce fichier permet aux utilisateurs de mettre à jour leurs informations personnelles, y compris leur adresse e-mail, leur mot de passe, et leurs préférences de notification. Le contrôleur interagit avec les modèles pour mettre à jour les informations dans la base de données, appliquant les règles de validation pertinentes. En cas de problème, le contrôleur retourne des messages d'erreur spécifiques pour informer les utilisateurs des modifications nécessaires à apporter avant que leurs informations soient correctement mises à jour.

```
1 <?php
2 require_once 'db.php';
3 require_once 'models/Category.php';
4
5 class CategoryController {
6     private $conn;
7     private $category;
8
9     public function __construct($db) {
10         $this->conn = $db;
11         $this->category = new Category($db);
12     }
13
14     public function getCategories() {
15         $stmt = $this->category->read();
16         $categories = $stmt->fetchAll(PDO::FETCH_ASSOC);
17         return $categories;
18     }
19
20     public function getCategory($id) {
21         $this->category->id = $id;
22         $stmt = $this->category->read_single();
23         $category = $stmt->fetch(PDO::FETCH_ASSOC);
24         return $category;
25     }
26 }
```

En conclusion, les contrôleurs dans l'application jouent un rôle crucial en fournissant l'interface entre les utilisateurs et la logique métier de l'application. Ils centralisent les actions utilisateur, appliquent les règles de validation, et interagissent avec les modèles pour garantir que les données sont correctement traitées, sécurisées et présentées à l'utilisateur final de manière efficace et cohérente.

- **Modèles (Models)**

Category.php : Le modèle Category.php dans l'application joue un rôle crucial en servant de couche abstraite entre la logique métier de l'application et les données physiques stockées dans la base de données. Ce modèle permet de structurer et de manipuler les informations relatives aux catégories de produits, ce qui facilite l'organisation des produits au sein de l'application. Le modèle Category.php contient des méthodes qui interagissent avec la base de données pour récupérer, créer, modifier ou supprimer des catégories. Cela inclut des méthodes comme `getAllCategories()`, qui retourne une liste de toutes les catégories disponibles dans la base de données, ou `updateCategory($id, $data)`, qui permet de mettre à jour les informations d'une catégorie spécifique.

Le rôle principal de Category.php est de centraliser la gestion des données des catégories, ce qui simplifie les modifications dans l'application. Lorsqu'il est nécessaire de changer le nom ou la description d'une catégorie, le modèle gère ces opérations, garantissant que les données sont correctement validées et que toutes les relations avec d'autres données (comme les produits associés) sont mises à jour en conséquence. En encapsulant ces fonctions dans Category.php, l'application bénéficie d'une architecture plus propre et modulaire, ce qui facilite les tests et les maintenances futures. De plus, le modèle s'assure que les erreurs de requêtes SQL sont capturées, ce qui évite les interruptions imprévues du service lors des manipulations de données.

En interaction avec les contrôleurs, Category.php agit comme un facilitateur entre les utilisateurs et la base de données. Les contrôleurs, en demandant des opérations de catégorie, font appel à Category.php pour exécuter les requêtes nécessaires. Par exemple, un contrôleur qui cherche à afficher les produits par catégorie utilisera Category.php pour récupérer la liste des catégories, puis sélectionnera les produits associés à chaque catégorie. Cette séparation des préoccupations entre la logique métier (dans les contrôleurs) et la manipulation des données (dans les modèles) permet de

maintenir un code propre et de limiter les interactions directes avec la base de données, réduisant ainsi le risque d'erreurs de programmation et de failles de sécurité.

Order.php : Le modèle Order.php dans l'application est crucial pour la gestion des commandes des utilisateurs. Il encapsule les données liées aux commandes et contient des méthodes pour interagir avec la base de données de manière efficace et sécurisée. Les principales fonctions de Order.php incluent la création de nouvelles commandes, la mise à jour des statuts des commandes existantes, et la récupération de l'historique des commandes pour un utilisateur spécifique. Ce modèle joue également un rôle dans le suivi des transactions, ce qui permet aux utilisateurs de vérifier le statut de leurs commandes passées, comme "en attente", "validée", "expédiée" ou "livrée".

Le Order.php centralise la logique de gestion des commandes en un seul endroit, ce qui simplifie le processus de mise à jour et de maintenance. Lorsqu'une nouvelle commande est passée, le modèle s'assure que toutes les informations nécessaires (comme les détails de la livraison, les produits achetés, le coût total, etc.) sont enregistrées dans la base de données. Le modèle utilise des relations de base de données pour connecter les commandes aux produits et aux utilisateurs, facilitant ainsi la gestion des transactions complexes impliquant plusieurs produits ou plusieurs utilisateurs. En interagissant avec les autres modèles, tels que Product.php, le Order.php peut mettre à jour les informations de stock en fonction des produits achetés, garantissant ainsi qu'il y a toujours suffisamment de stock disponible pour satisfaire les commandes.

En ce qui concerne la mise à jour des statuts des commandes, Order.php fournit des méthodes qui changent l'état d'une commande en fonction des étapes du processus de commande. Cela peut inclure le passage de "en attente" à "validée" après que le paiement a été effectué, ou de "validée" à "expédiée" après que les produits ont été envoyés au client. Le modèle s'assure que ces changements sont effectués correctement et en toute sécurité, minimisant les risques d'erreurs humaines ou de manipulations frauduleuses. De plus, Order.php utilise des vérifications de sécurité pour s'assurer que les utilisateurs ne modifient pas directement les statuts des commandes sans les autorisations nécessaires.

Product.php : Le modèle Product.php dans l'application est responsable de la gestion des données liées aux produits, encapsulant toutes les opérations fréquentes sur les produits comme la recherche, l'ajout, la mise à jour et la suppression. Il contient des méthodes pour récupérer des informations spécifiques sur les produits, comme les détails du produit (nom, description, prix, catégorie, etc.), et pour gérer les informations de stock (quantité disponible). Ce modèle permet une interaction simplifiée avec la base de données, en isolant la logique métier du code applicatif, ce qui facilite la maintenance et les mises à jour futures.

Le modèle Product.php fournit des méthodes pour rechercher des produits en fonction de divers critères tels que la catégorie, le nom ou le prix. Cela est crucial pour l'optimisation de l'expérience utilisateur, en permettant par exemple aux utilisateurs de rechercher facilement un produit spécifique ou de naviguer à travers les catégories pour découvrir des produits similaires. En outre, Product.php contient des méthodes pour ajouter de nouveaux produits à la base de données, mettre à jour les détails existants des produits ou supprimer des produits obsolètes. Ces opérations sont effectuées en respectant les règles de validation des données, garantissant que toutes les informations saisies sont correctes avant d'être enregistrées dans la base de données.

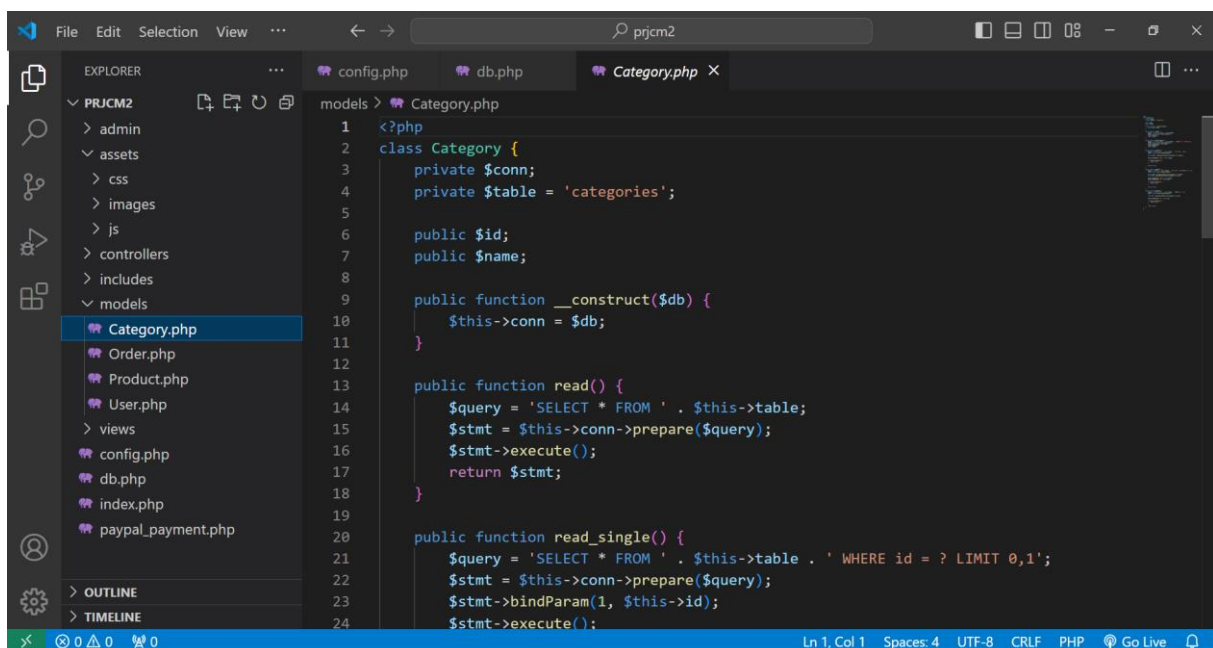
En ce qui concerne la gestion des stocks, Product.php offre des méthodes pour augmenter ou diminuer les quantités disponibles en fonction des achats et des retours. Lorsque les stocks d'un produit sont bas, le modèle peut déclencher des alertes pour les administrateurs ou pour la réapprovisionnement automatique via des services externes. De plus, le modèle Product.php

interagit avec le modèle Order.php pour mettre à jour le stock lors de la validation d'une commande, assurant ainsi que la gestion des produits est intégrée au processus de commande global de l'application.

User.php : Le modèle User.php est essentiel pour la gestion des données utilisateur dans l'application. Il encapsule les informations personnelles des utilisateurs, leurs rôles (client ou administrateur), et les états d'authentification. Ce modèle est conçu pour protéger les informations des utilisateurs, notamment les mots de passe, en les stockant de manière sécurisée, typiquement en les hachant avant de les enregistrer dans la base de données. User.php offre des méthodes pour enregistrer de nouveaux utilisateurs, vérifier les informations de connexion, et gérer les sessions utilisateur pour l'accès sécurisé aux fonctionnalités de l'application.

Lors de l'enregistrement d'un utilisateur, User.php valide les informations saisies, comme le nom, l'adresse e-mail, et le mot de passe. Il applique des règles de validation strictes pour s'assurer que les informations sont conformes aux exigences de sécurité, minimisant ainsi les risques de failles de sécurité comme les injections SQL ou les attaques par force brute. Une fois que les informations sont validées, le modèle hache le mot de passe avant de l'enregistrer dans la base de données, protégeant ainsi l'utilisateur contre les accès non autorisés. De plus, User.php gère les différentes permissions basées sur le rôle de l'utilisateur (client, administrateur), facilitant ainsi la gestion des accès et des fonctionnalités spécifiques selon le profil utilisateur.

Pour la gestion des sessions utilisateur, User.php fournit des méthodes pour démarrer, vérifier et terminer les sessions. Lorsqu'un utilisateur se connecte, le modèle commence une session sécurisée, stocke les informations pertinentes (comme l'ID de l'utilisateur, le rôle, et les préférences de session), et retourne un identifiant de session. Le modèle vérifie également les sessions en cours pour s'assurer que seul l'utilisateur actif peut accéder à ses informations personnelles et aux fonctionnalités de l'application. En cas d'expiration de session ou de déconnexion, User.php met fin à la session en toute sécurité, supprimant les données stockées temporairement pour éviter les risques de sécurité.



```
1 <?php
2 class Category {
3     private $conn;
4     private $table = 'categories';
5
6     public $id;
7     public $name;
8
9     public function __construct($db) {
10         $this->conn = $db;
11     }
12
13     public function read() {
14         $query = 'SELECT * FROM ' . $this->table;
15         $stmt = $this->conn->prepare($query);
16         $stmt->execute();
17         return $stmt;
18     }
19
20     public function read_single() {
21         $query = 'SELECT * FROM ' . $this->table . ' WHERE id = ? LIMIT 0,1';
22         $stmt = $this->conn->prepare($query);
23         $stmt->bindParam(1, $this->id);
24         $stmt->execute();
```

En résumé, les modèles dans l'application jouent un rôle crucial en encapsulant la logique métier et les interactions avec la base de données, en fournissant un accès simplifié et sécurisé aux données. Ils

garantissent que les données sont validées correctement avant d'être stockées ou manipulées, et ils facilitent l'interaction avec les contrôleurs pour l'affichage des informations pertinentes aux utilisateurs finaux. Grâce à cette séparation, l'application reste modulaire, maintenable et sécurisée, ce qui permet de gérer efficacement la complexité des données tout en offrant une interface utilisateur intuitive.

- **Vues (Views)**

`add_to_cart.php` : La vue `add_to_cart.php` est une interface utilisateur essentielle pour l'ajout de produits au panier. Cette vue agit comme un intermédiaire entre l'utilisateur et les fonctionnalités de gestion du panier dans l'application. Elle permet aux utilisateurs de sélectionner les produits qu'ils souhaitent ajouter à leur panier en fournissant des options telles que la quantité ou la variante du produit (par exemple, taille ou couleur). Cette vue vérifie également si les informations transmises par l'utilisateur sont valides avant de les envoyer à la logique métier via le contrôleur associé.

L'affichage de `add_to_cart.php` est conçu pour offrir une interface utilisateur intuitive, facilitant l'expérience d'achat. Les utilisateurs peuvent choisir la quantité de chaque produit qu'ils souhaitent ajouter, et, si disponible, sélectionner une variante (comme une taille ou une couleur spécifique). La vue s'assure que ces choix sont correctement visualisés et ajustés sur la page, offrant une rétroaction immédiate en cas d'erreur (par exemple, si une quantité n'est pas disponible ou si une variante non validée est choisie). En utilisant des formulaires et des boutons d'action, la vue `add_to_cart.php` permet aux utilisateurs d'ajouter des produits au panier de manière fluide, tout en garantissant que les informations fournies sont transmises au contrôleur de manière sécurisée pour validation.

En termes de validation, `add_to_cart.php` joue un rôle clé en s'assurant que toutes les informations saisies par l'utilisateur sont correctement traitées avant d'être transmises aux modèles. Par exemple, elle vérifie si la quantité demandée est disponible en stock et si la variante choisie est compatible avec le produit sélectionné. Cette validation est importante pour éviter les erreurs qui pourraient survenir lors de l'ajout au panier et pour améliorer la satisfaction de l'utilisateur en fournissant un feedback immédiat sur les choix faits. En encapsulant cette logique dans la vue, l'application peut offrir une expérience utilisateur plus agréable et plus sécurisée.

`cart.php` : La vue `cart.php` offre une interface utilisateur interactive pour visualiser et gérer le contenu du panier. Elle est conçue pour permettre aux utilisateurs de modifier les quantités, de supprimer des articles ou de recalculer le total des achats. Cette vue est essentielle pour l'expérience utilisateur car elle fournit un contrôle direct sur les produits dans le panier, influençant ainsi l'expérience de magasinage en ligne.

cart.php affiche une liste des produits actuellement dans le panier, avec des options pour modifier la quantité de chaque produit ou retirer un produit spécifique. En utilisant des formulaires et des boutons d'action, les utilisateurs peuvent facilement ajuster leurs achats en modifiant les quantités de chaque produit, ce qui déclenche une mise à jour immédiate du panier. Cette vue calcule également le total des achats en temps réel, en tenant compte des quantités modifiées et des prix de chaque produit. L'affichage de ces informations en temps réel est crucial pour aider les utilisateurs à comprendre les coûts totaux avant de passer à la caisse.

En plus des fonctionnalités de gestion de base, cart.php inclut des liens ou des boutons pour passer au processus de paiement via la vue checkout.php. Cela permet aux utilisateurs de continuer leur expérience d'achat sans avoir à quitter la page de gestion du panier. cart.php interagit étroitement avec le modèle Order.php pour gérer les mises à jour de panier et le suivi des commandes. Par exemple, lorsqu'un utilisateur met à jour la quantité d'un produit, cart.php envoie ces modifications au modèle Order.php pour recalculer le montant total de la commande, assurant ainsi une cohérence entre les informations affichées à l'utilisateur et les données stockées en base.

checkout.php : La vue checkout.php présente le processus de paiement et recueille toutes les informations nécessaires pour finaliser une commande. Elle joue un rôle crucial dans l'expérience utilisateur, en facilitant le passage du panier d'achats à une transaction en ligne. Cette vue est conçue pour offrir une interface fluide et sécurisée, intégrant les étapes de vérification d'identité, d'adresse de livraison, de méthode de paiement et de confirmation finale.

checkout.php présente un formulaire avec des champs pour les informations nécessaires à la livraison (adresse, ville, code postal, pays, etc.), ainsi que des options pour choisir la méthode de paiement (carte de crédit, PayPal, virement bancaire, etc.). Les utilisateurs remplissent ces informations en suivant un processus étape par étape, avec des validations en temps réel pour éviter les erreurs de saisie. Chaque étape est accompagnée de boutons de navigation, permettant aux utilisateurs de passer d'une section à l'autre sans confusion.

En interaction avec les modèles, checkout.php communique avec Order.php pour enregistrer les détails de la commande. Une fois que les informations du formulaire sont validées, les données sont envoyées à Order.php pour créer un nouvel enregistrement de commande. La vue gère également les calculs pour déterminer le coût total de la commande, en ajoutant les frais de livraison et les taxes selon les informations fournies par l'utilisateur. Cette transparence est essentielle pour éviter toute surprise désagréable lors de la finalisation de la commande.

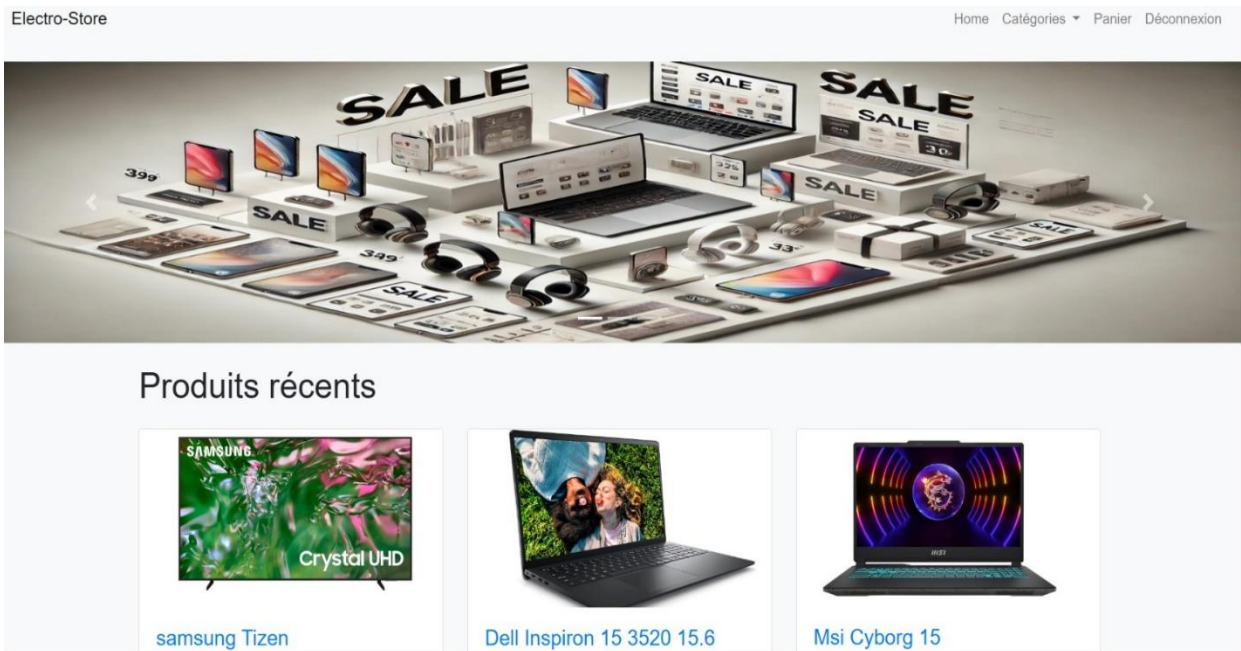
De plus, checkout.php comprend une section pour récapituler les produits dans le panier, avec des options pour modifier les quantités ou supprimer des articles avant de procéder au paiement. Cette

fonction permet aux utilisateurs de réviser leurs achats avant la confirmation finale, offrant ainsi un contrôle supplémentaire sur le processus d'achat. Une fois la commande validée, checkout.php envoie une confirmation par e-mail à l'utilisateur et aux administrateurs, garantissant ainsi une communication efficace sur l'état de la commande.

home.php : La page d'accueil home.php est la première interface que les utilisateurs voient lorsqu'ils visitent l'application. Elle joue un rôle clé en guidant les utilisateurs vers les produits ou les catégories les plus populaires, permettant ainsi une découverte facile des articles. home.php est conçue pour attirer l'attention des visiteurs sur les offres spéciales, les nouveaux produits ou les meilleures ventes, en utilisant des images, des descriptions courtes, et des boutons d'appel à l'action.

Cette vue est responsable de la présentation de contenu dynamique, souvent alimenté par des données provenant de la base de données. Elle utilise des requêtes SQL pour récupérer les produits les plus populaires, les dernières mises à jour de stock ou les catégories vedettes, puis affiche ces informations sous forme de carrousels, de listes ou de grilles, selon la mise en page choisie. home.php est également équipée de filtres de recherche et de filtres par catégorie, permettant aux utilisateurs de trouver facilement des produits spécifiques ou de naviguer à travers les produits disponibles par thème ou par type.

En interaction avec les modèles, home.php utilise les méthodes de Product.php pour récupérer les produits et les catégories. Cette vue joue un rôle clé dans l'optimisation de la navigation utilisateur, en combinant les données de plusieurs modèles pour offrir une expérience de shopping unifiée. Par exemple, elle peut utiliser Product.php pour récupérer des informations détaillées sur les produits, puis utiliser Category.php pour structurer ces produits en catégories, permettant aux utilisateurs de filtrer les résultats par type ou par prix. Cela rend la navigation plus intuitive et permet aux utilisateurs de découvrir facilement des produits similaires ou complémentaires.



login.php et register.php : Les fichiers login.php et register.php offrent les interfaces utilisateur nécessaires pour l'authentification et l'inscription des utilisateurs. login.php permet aux utilisateurs existants de se connecter en entrant leurs informations d'identification (nom d'utilisateur ou e-mail, mot de passe). La vue vérifie les informations saisies contre les données stockées dans User.php, utilisant des fonctions de hachage pour sécuriser les mots de passe. register.php, quant à lui, permet aux nouveaux utilisateurs de s'inscrire en fournissant des informations personnelles (nom, adresse e-mail, mot de passe, etc.).

Dans login.php, la vue affiche un formulaire avec des champs pour le nom d'utilisateur et le mot de passe, accompagnés de boutons pour soumettre les informations. Elle inclut également un lien pour récupérer les informations de connexion en cas d'oubli. Les formulaires sont protégés contre les attaques courantes comme les injections SQL et les failles de script cross-site (XSS) par l'intermédiaire de fonctions de validation des entrées présentes dans validate_input.php. En utilisant User.php, login.php vérifie que les informations d'identification sont correctes, et redirige l'utilisateur vers la page appropriée en fonction de son rôle (client ou administrateur).

register.php demande aux utilisateurs de remplir un formulaire avec des informations telles que leur nom, adresse e-mail, et mot de passe. Ces informations sont également validées contre les règles de sécurité définies dans validate_input.php. Une fois les informations validées, register.php envoie les données à User.php pour créer un nouvel utilisateur. Le modèle hache le mot de passe avant de le stocker, garantissant que les informations de l'utilisateur sont sécurisées. Après l'inscription, les utilisateurs sont redirigés vers la page de connexion, où ils peuvent se connecter immédiatement avec leurs nouvelles informations d'identification.

Electro
Actualiser (Ctrl+R)
Home
Catégories
Connexion
Inscription

Connexion

Nom d'utilisateur

Mot de passe

Se Connecter

ELECTRO-STORE

Bienvenue sur Electro-Store, votre destination ultime pour les gadgets innovants et les dernières avancées technologiques. Que vous soyez un passionné de high-tech à la recherche d'appareils de pointe ou un amateur de domotique, notre boutique en ligne propose une large gamme de produits soigneusement sélectionnés pour répondre à tous les besoins. De smartphones dernier cri aux objets connectés intelligents, en passant par les ordinateurs performants et les accessoires audio haut de gamme.

LIENS DE NOTRE SITE WEB

A propos

Contactez-nous

License d'utilisation

Retour et Echanges

MÉTHODES DE PAIEMENT

VISA

MasterCard

PayPal

Apple Pay

Ces vues, `login.php` et `register.php`, sont conçues pour fournir une expérience utilisateur sans friction, en garantissant que les utilisateurs peuvent facilement accéder à leurs comptes ou s'inscrire pour de nouvelles interactions sur le site. Elles simplifient le processus de connexion et d'inscription, tout en assurant que les informations utilisateur sont correctement validées et stockées en toute sécurité, minimisant ainsi les risques de sécurité pour l'application.

```

1  <?php
2  class Category {
3      private $conn;
4      private $table = 'categories';
5
6      public $id;
7      public $name;
8
9      public function __construct($db) {
10         $this->conn = $db;
11     }
12
13     public function read() {
14         $query = 'SELECT * FROM ' . $this->table;
15         $stmt = $this->conn->prepare($query);
16         $stmt->execute();
17         return $stmt;
18     }
19
20     public function read_single() {
21         $query = 'SELECT * FROM ' . $this->table . ' WHERE id = ? LIMIT 0,1';
22         $stmt = $this->conn->prepare($query);
23         $stmt->bindParam(1, $this->id);
24         $stmt->execute();

```

En conclusion, les vues dans l'application jouent un rôle crucial en fournissant une interface utilisateur conviviale pour l'interaction avec les contrôleurs et les modèles. Elles garantissent que les utilisateurs peuvent interagir efficacement avec les fonctionnalités de l'application, tout en offrant

des validations en temps réel et des messages d'erreur appropriés pour une meilleure expérience utilisateur.

- **Section Admin et Includes**

Section Admin :

La section administrateur de l'application est cruciale pour la gestion efficace des produits, des commandes et des catégories. Les fichiers inclus dans cette section, tels que `admin_add_product.php`, `admin_dashboard.php`, `admin_delete_order.php`, `admin_delete_product.php`, `admin_edit_product.php`, `admin_manage_categories.php`, `admin_manage_orders.php`, `admin_manage_products.php`, `admin_navbar.php`, `admin_view_order.php`, `delete_category.php`, `edit_category.php`, et `index.php`, permettent aux administrateurs d'avoir un contrôle précis et centralisé sur différents aspects de l'application. Ces fichiers sont conçus pour offrir des interfaces intuitives et faciles à utiliser, garantissant que les administrateurs puissent effectuer leurs tâches de manière rapide et efficace.

1. **admin_add_product.php** : Ce fichier permet aux administrateurs d'ajouter de nouveaux produits à l'application. Il offre une interface pour remplir des informations comme le nom du produit, la description, le prix, les catégories associées, et les images. L'interface est conçue pour réduire les erreurs grâce à des validations en temps réel, s'assurant que les données saisies par l'administrateur sont conformes aux règles de l'application avant d'être stockées dans la base de données. Ce fichier joue un rôle clé dans la gestion dynamique du catalogue de produits, en permettant l'ajout rapide de nouveaux articles sans nécessiter d'accès direct à la base de données.
2. **admin_dashboard.php** : Le tableau de bord administrateur `admin_dashboard.php` est l'interface principale pour les administrateurs. Il offre une vue d'ensemble de toutes les activités récentes sur le site, telles que les commandes récentes, les produits ajoutés récemment, et les statistiques de ventes. Ce fichier présente également des graphiques et des tableaux pour une analyse rapide des performances des produits, des catégories et des commandes. Il fournit des liens directs vers les autres fichiers administratifs, permettant une navigation facile entre les différentes sections de gestion.

Admin Dashboard

Manage Products

Manage Categories

Manage Orders

Logout

3. **admin_delete_order.php** : Cette vue permet aux administrateurs de supprimer des commandes spécifiques. Elle affiche une liste des commandes passées, avec des options pour rechercher, filtrer et supprimer des commandes sélectionnées. Le processus est sécurisé pour éviter les erreurs de suppression accidentelle et inclut une vérification supplémentaire avant de supprimer définitivement une commande. Ce fichier est crucial pour maintenir l'intégrité des données dans la base de données et pour gérer les erreurs ou les commandes annulées.
4. **admin_delete_product.php** : La gestion des produits n'est pas complète sans la possibilité de supprimer des articles de manière sécurisée. `admin_delete_product.php` offre aux administrateurs l'interface pour rechercher, filtrer et supprimer des produits de la base de données. La suppression est permanente, et les administrateurs doivent confirmer leurs actions pour chaque produit. Cette fonction est conçue pour minimiser les erreurs humaines en offrant une confirmation double avant d'effacer les produits.
5. **admin_edit_product.php** : Permet aux administrateurs de modifier les détails des produits existants. Il présente une interface similaire à `admin_add_product.php`, mais avec des champs déjà remplis basés sur les informations existantes dans la base de données. Les administrateurs peuvent ajuster les prix, les descriptions, les catégories, et les images. Les modifications sont enregistrées en temps réel et directement dans la base de données. Cette vue facilite les mises à jour de masse en regroupant toutes les informations pertinentes sur un produit en un seul endroit.
6. **admin_manage_categories.php** : Cette interface est dédiée à la gestion des catégories de produits. Elle permet aux administrateurs de créer, modifier et supprimer des catégories en toute sécurité. Les catégories jouent un rôle clé dans la structuration du catalogue de produits, et cette vue offre une interface intuitive pour organiser les catégories, assigner des produits à celles-ci, et maintenir l'ordre dans l'application. Les administrateurs peuvent également réorganiser les catégories pour une navigation plus efficace.
7. **admin_manage_orders.php** : Cette section fournit une interface détaillée pour le suivi et la gestion des commandes. Les administrateurs peuvent filtrer les commandes par statut (en attente, en cours de traitement, expédiées, annulées), rechercher des commandes

spécifiques, et modifier les détails des commandes si nécessaire. Cette vue est cruciale pour garantir que toutes les commandes sont traitées correctement et rapidement, en mettant l'accent sur la satisfaction client.

8. **admin_manage_products.php** : admin_manage_products.php permet aux administrateurs de voir tous les produits disponibles, de trier par catégorie, prix ou date d'ajout, et de rechercher par nom. Cette vue offre également des options pour modifier rapidement les informations des produits, comme les prix, les descriptions et les quantités en stock. Elle permet aux administrateurs d'avoir un contrôle total sur l'ensemble du catalogue de produits, facilitant la gestion des stocks et l'organisation des offres promotionnelles.

Gérer les Catégories

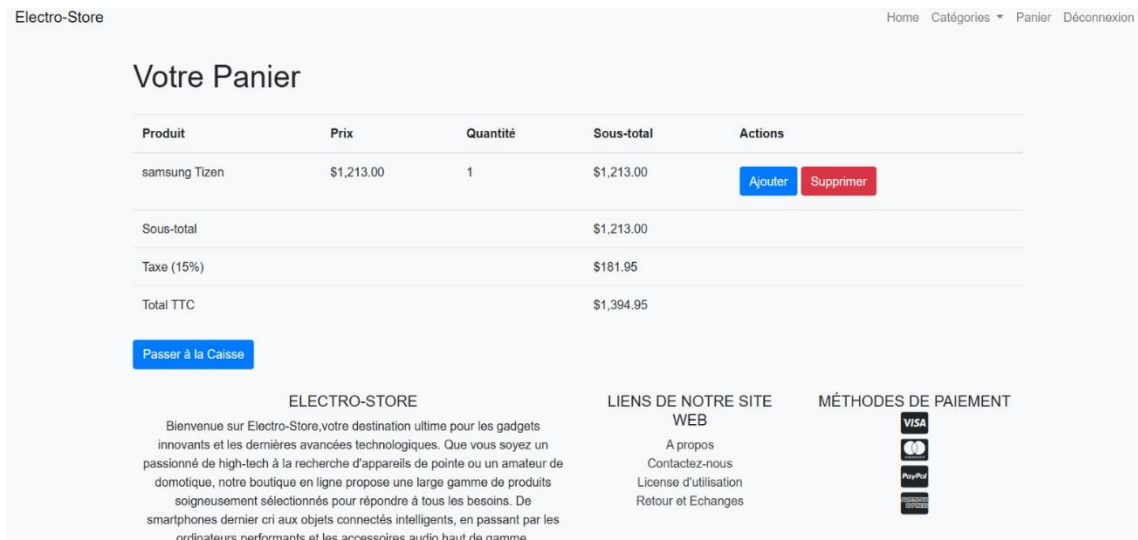
Nom de la Catégorie

Ajouter

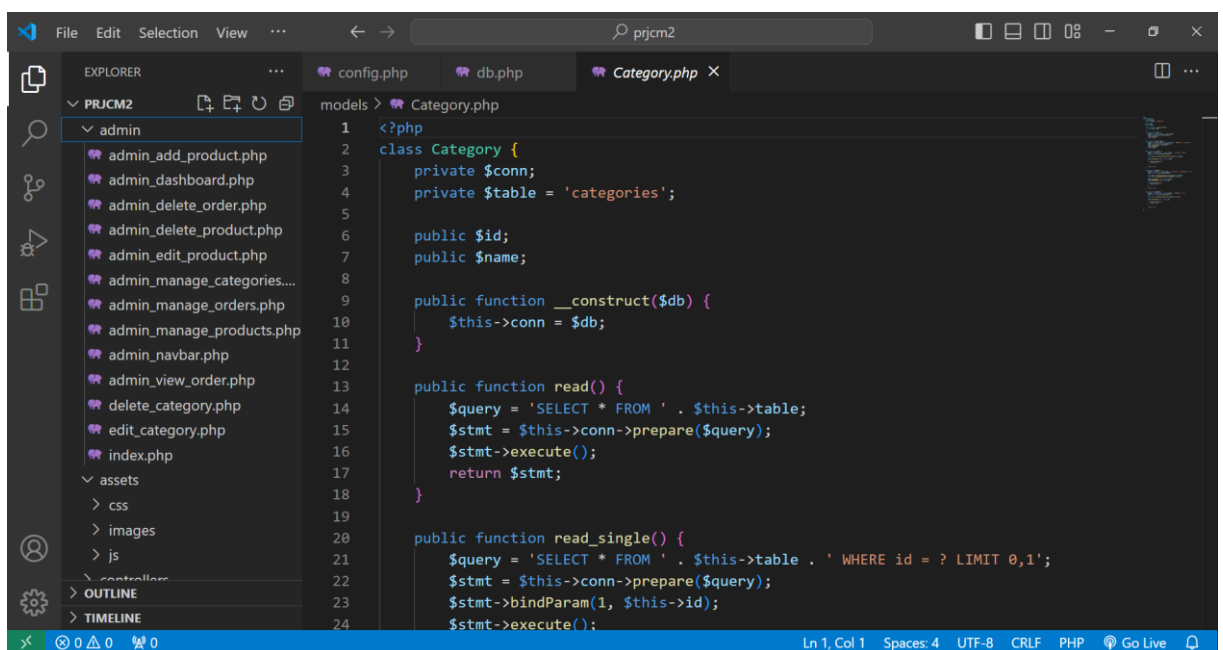
Liste des Catégories

ID	Nom	Actions
17	Ordinateurs	Supprimer
18	Televiseurs	Supprimer
16	Telephone	Supprimer

9. **admin_navbar.php** : La barre de navigation administrateur fournit un accès direct à toutes les sections administratives. Elle inclut des liens vers admin_dashboard.php, admin_add_product.php, admin_manage_orders.php, admin_manage_categories.php, admin_manage_products.php, et d'autres fichiers administratifs. Cette barre de navigation est cruciale pour une navigation rapide et efficace entre les différentes sections de l'administration, minimisant le temps passé à chercher les informations nécessaires.
10. **admin_view_order.php** : Permet aux administrateurs de voir les détails d'une commande spécifique, y compris les produits commandés, les quantités, les adresses de livraison, et le statut de la commande. Cette vue est indispensable pour comprendre le contexte d'une commande, en particulier pour les commandes complexes ou les réclamations clients. Les administrateurs peuvent également annuler ou modifier une commande directement à partir de cette interface.



11. **delete_category.php** : Cette vue permet aux administrateurs de supprimer définitivement des catégories de produits. Elle affiche une liste des catégories existantes, avec des options pour rechercher, filtrer et supprimer en masse. La suppression de catégories est un processus sensible, car elle peut affecter de nombreux produits associés à cette catégorie. Une confirmation double est nécessaire pour éviter toute erreur.
12. **edit_category.php** : Permet aux administrateurs de modifier les informations de catégories existantes. Les champs disponibles incluent le nom de la catégorie, une description, et les produits associés. Les modifications sont enregistrées en temps réel dans la base de données, garantissant que toutes les informations de la catégorie sont à jour.
13. **index.php** : La page d'accueil pour l'administration, elle offre un résumé des activités récentes et un accès rapide aux différentes sections administratives. Les administrateurs peuvent accéder directement aux commandes, produits et catégories via des liens ou des boutons, simplifiant ainsi la gestion de l'application.



Includes :

Les fichiers d'inclusion jouent un rôle crucial dans la standardisation et la gestion des composants réutilisables dans l'application. Ils fournissent des éléments de navigation, des barres d'état, des menus déroulants, et des formulaires pour garantir une expérience utilisateur cohérente à travers les différentes pages. Les fichiers d'inclusion tels que header.php, footer.php, navbar.php, fonctions.php, et validate_input.php sont intégrés dans chaque page de l'application pour simplifier l'intégration des fonctionnalités communes.

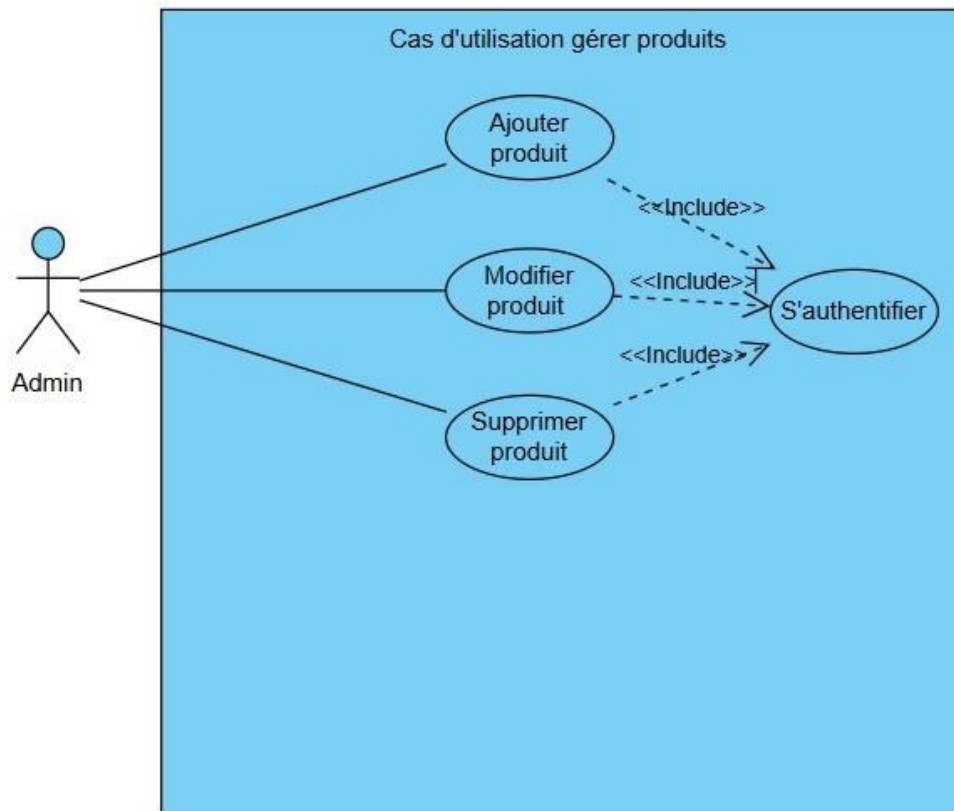
1. **header.php** : Le fichier header.php est utilisé pour afficher les informations d'en-tête sur chaque page de l'application. Il inclut des éléments tels que le logo, le menu de navigation, les boutons de connexion et d'inscription, ainsi que des liens vers les pages les plus utilisées (comme home.php, cart.php, checkout.php, etc.). Ce fichier est essentiel pour maintenir une structure uniforme dans l'application, assurant que l'utilisateur est toujours informé de son emplacement dans le site web. Il peut également inclure des scripts JavaScript nécessaires à la fonctionnalité de la page (comme des validations en temps réel ou des carrousels d'images).
2. **footer.php** : Le fichier footer.php contient les informations de pied de page, telles que les informations de copyright, les liens vers les pages importantes (comme les conditions d'utilisation, la politique de confidentialité, le support client, etc.). Il sert de point de contact pour les utilisateurs et aide à améliorer la navigation en fournissant des informations supplémentaires sans surcharger la page actuelle. Le pied de page est également utilisé pour afficher des notifications ou des alertes importantes, comme les mises à jour de la maintenance ou les nouvelles fonctionnalités ajoutées à l'application.
3. **navbar.php** : navbar.php est utilisé pour créer la barre de navigation principale sur chaque page de l'application. Il inclut des liens vers les sections principales comme home.php, shop.php, cart.php, contact.php, et admin_dashboard.php. Cette vue standardise la navigation à travers les différentes sections de l'application, assurant que les utilisateurs peuvent facilement accéder aux informations ou aux actions nécessaires, quel que soit l'endroit où ils se trouvent dans le site. La barre de navigation peut également afficher l'état de connexion de l'utilisateur, en changeant les options disponibles (comme afficher un lien login.php si l'utilisateur est non connecté ou logout.php si l'utilisateur est connecté).
4. **fonctions.php** : Ce fichier contient des fonctions réutilisables utilisées tout au long de l'application. Il peut inclure des fonctions de validation des entrées, des fonctions pour gérer les dates, des fonctions pour le formatage des prix ou encore des fonctions de manipulation de chaîne de caractères. Ces fonctions simplifient l'écriture de code dans les autres fichiers, car elles centralisent la logique réutilisable et garantissent que toutes les pages de l'application utilisent les mêmes routines pour des tâches similaires.
5. **validate_input.php** : Ce fichier est responsable de la validation des entrées utilisateur. Il inclut des fonctions pour vérifier la validité des adresses e-mail, des mots de passe, des numéros de téléphone, etc. Il s'assure que toutes les données entrantes respectent les règles définies par l'application avant d'être traitées ou enregistrées. Ce processus est crucial pour la sécurité, car il empêche les injections de code ou autres manipulations malveillantes des données par les utilisateurs. Les fonctions de validation sont utilisées partout dans l'application, garantissant que seules les données sûres et légitimes sont utilisées.

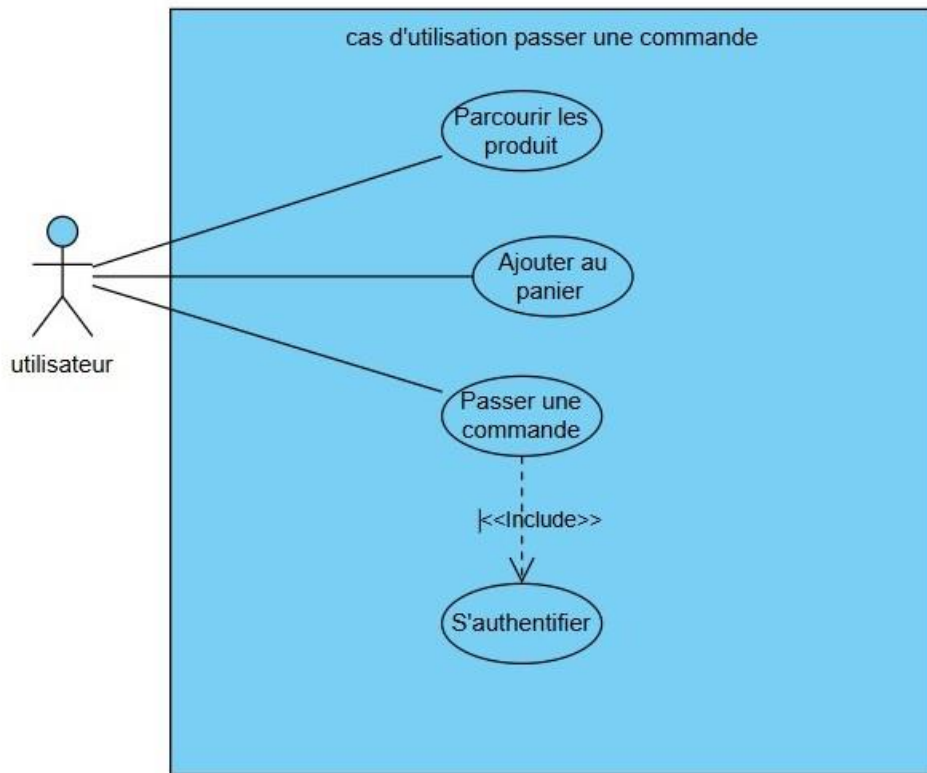
The screenshot shows the Visual Studio Code interface with a project named 'prjcm2'. The Explorer sidebar on the left displays the project structure, including folders like 'admin', 'assets', 'css', 'images', 'js', 'controllers', and 'includes', along with various PHP files. The 'includes' folder is currently selected. The main editor area shows the 'Category.php' file, which contains a PHP class named 'Category'. The class has private properties for database connection and table name, and public properties for ID and name. It includes methods for constructing the object, reading data from the database, and reading a single record.

```
1 <?php
2 class Category {
3     private $conn;
4     private $table = 'categories';
5
6     public $id;
7     public $name;
8
9     public function __construct($db) {
10         $this->conn = $db;
11     }
12
13     public function read() {
14         $query = 'SELECT * FROM ' . $this->table;
15         $stmt = $this->conn->prepare($query);
16         $stmt->execute();
17         return $stmt;
18     }
19
20     public function read_single() {
21         $query = 'SELECT * FROM ' . $this->table . ' WHERE id = ? LIMIT 0,1';
22         $stmt = $this->conn->prepare($query);
23         $stmt->bindParam(1, $this->id);
24         $stmt->execute();
```

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF PHP Go Live

Diagrammes de cas d'utilisation





- **Description et Explication des Diagrammes**

Diagramme 1 : Cas d'utilisation - Gérer les produits (Administrateur)

Description : Ce diagramme illustre les différentes actions qu'un administrateur peut effectuer dans le système pour gérer les produits. Les cas d'utilisation comprennent **Ajouter un produit**, **Modifier un produit**, et **Supprimer un produit**. Tous ces cas nécessitent une authentification préalable pour garantir que seules les personnes autorisées (administrateurs) puissent accéder à ces fonctionnalités sensibles.

Explication :

- **Acteur principal :** L'administrateur, représenté par le symbole d'humain à gauche, interagit avec le système pour gérer les produits.
- **Cas d'utilisation principaux :**
 - **Ajouter un produit :** Permet à l'administrateur de créer de nouveaux produits et de les enregistrer dans la base de données.
 - **Modifier un produit :** Permet de modifier les informations d'un produit existant, telles que son nom, son prix ou sa catégorie.
 - **Supprimer un produit :** Offre la possibilité de retirer un produit de la liste des produits disponibles.

- **Relation "Include" avec "S'authentifier"** : Tous les cas d'utilisation incluent le processus d'authentification pour garantir que l'administrateur est bien connecté avant d'effectuer une opération.

Ce diagramme met en évidence l'importance de la sécurité et du contrôle d'accès dans la gestion des produits.

Diagramme 2 : Cas d'utilisation - Passer une commande (Utilisateur)

Description : Ce diagramme illustre les actions principales qu'un utilisateur peut effectuer pour passer une commande. Les cas d'utilisation incluent **Parcourir les produits**, **Ajouter au panier**, et **Passer une commande**, avec un lien vers le cas d'utilisation **S'authentifier**.

Explication :

- **Acteur principal** : L'utilisateur, représenté par l'icône humaine, interagit avec le système pour parcourir les produits et finaliser une commande.
- **Cas d'utilisation principaux** :
 - **Parcourir les produits** : Permet à l'utilisateur de consulter les produits disponibles sur le site.
 - **Ajouter au panier** : L'utilisateur sélectionne les produits qu'il souhaite acheter et les ajoute au panier.
 - **Passer une commande** : Une fois les produits ajoutés, l'utilisateur peut finaliser son achat.
- **Relation "Include" avec "S'authentifier"** : Avant de passer une commande, l'utilisateur doit être authentifié. Cela garantit que les commandes sont associées à un compte utilisateur valide et sécurise le processus.

Ce diagramme montre comment le système guide l'utilisateur tout au long du processus d'achat tout en s'assurant que les interactions clés nécessitent une authentification.

Comparaison et Importance :

Ces deux diagrammes montrent les principales interactions entre les utilisateurs (administrateurs et utilisateurs standards) et le système. Ils mettent l'accent sur la sécurité via l'authentification et démontrent comment les rôles diffèrent dans leurs fonctionnalités accessibles. Les administrateurs gèrent les produits, tandis que les utilisateurs standard se concentrent sur la navigation et l'achat.

Conclusion

Enfin L'application que nous avons développée utilise une architecture Modèle-Vue-Contrôleur (MVC) pour organiser et gérer efficacement les différentes fonctionnalités du site web. En tant que binôme, nous avons collaboré étroitement pour comprendre chaque composant et s'assurer que chaque partie du projet était bien intégrée.

Tout d'abord, les fichiers de configuration, tels que `config.php` et `db.php`, sont essentiels pour la cohérence de l'application. `config.php` centralise les constantes et les paramètres nécessaires à la connexion à la base de données, aux URLs de l'application, et aux paramètres spécifiques à l'environnement d'exécution (local ou production). Grâce à cette centralisation, nous pouvons facilement ajuster les paramètres comme le changement de l'hôte de la base de données ou des URLs principales sans toucher directement au code applicatif. Cela simplifie les mises à jour et garantit la portabilité de l'application, en assurant que toutes les constantes nécessaires sont centralisées dans un seul endroit.

Le fichier `db.php` est l'intermédiaire clé entre l'application et la base de données. Il encapsule la logique de connexion, utilisant des bibliothèques comme `mysqli` ou `PDO` pour établir des connexions sécurisées et stables. Cette encapsulation nous permet de gérer les erreurs de connexion de manière cohérente et informatisée, en affichant un message d'erreur approprié si une connexion échoue. Cela protège le système contre des comportements inattendus et garantit que les utilisateurs reçoivent des retours clairs en cas de problèmes techniques. En encapsulant cette logique dans `db.php`, nous avons simplifié l'interaction avec la base de données à travers tout le projet, en réduisant le code répétitif et en centralisant la gestion des connexions.

Les contrôleurs sont les gestionnaires de l'application, permettant aux utilisateurs de naviguer et d'interagir efficacement avec les données. Le `CategoryController.php` gère toutes les actions liées aux catégories de produits. Il agit comme un pont entre les utilisateurs et la logique métier, en interagissant avec les modèles pour afficher, créer, modifier ou supprimer des catégories. Nous avons conçu ce contrôleur pour s'assurer que les règles métier sont appliquées correctement avant de transmettre les données aux vues. Cela nous a permis de maintenir la cohérence des données et de fournir une navigation utilisateur fluide, réduisant ainsi les erreurs utilisateur lors de la manipulation des données.

Le `OrderController.php` a été développé pour coordonner les processus liés aux commandes. Nous avons mis l'accent sur la création, la modification, la validation, et l'annulation des commandes. Ce contrôleur gère également les interactions avec la base de données pour enregistrer ou récupérer les informations liées aux commandes, garantissant une gestion précise des transactions. L'objectif était de créer une interface utilisateur intuitive qui guide les utilisateurs à travers le processus de commande, tout en minimisant les erreurs de saisie et en assurant que toutes les données sont correctement enregistrées dans la base de données.

Le `ProductController.php` a été conçu pour gérer toutes les opérations liées aux produits. Nous avons inclus des fonctionnalités pour afficher les produits, mettre à jour leurs détails, les supprimer, et les ajouter à la base de données. Ce contrôleur est crucial car il permet non seulement de gérer les informations produit de manière efficace, mais il offre également des fonctionnalités avancées telles que la recherche par catégorie ou le tri des produits, améliorant ainsi l'expérience utilisateur. Nous avons travaillé pour assurer que toutes les données produit soient mises à jour en temps réel, ce qui est essentiel pour offrir aux utilisateurs une information précise et pertinente sur le site.

Le `UserController.php` s'occupe des opérations liées aux utilisateurs. Nous avons mis en œuvre des fonctions pour gérer l'enregistrement, la connexion, et la gestion des profils utilisateurs. Cela inclut la vérification des données entrantes, l'application des règles de validation, et l'interaction avec le modèle `User` pour effectuer des actions sur la base de données. Nous avons veillé à ce que les utilisateurs non authentifiés soient redirigés vers les pages appropriées, garantissant ainsi une sécurité maximale et une gestion correcte des sessions utilisateur.

Les modèles, tels que `Category.php`, `Order.php`, `Product.php`, et `User.php`, représentent les différentes entités de l'application et fournissent des abstractions pour interagir avec les données. Le modèle `Category.php` permet d'accéder aux données des catégories, offrant des méthodes pour récupérer des listes, mettre à jour des informations, et garantir la cohérence des données. Nous avons conçu ce modèle pour agir comme un pont entre la logique métier de l'application et les données en base, ce qui facilite la gestion des catégories de manière centralisée et cohérente.

Le modèle `Order.php` encapsule les données des commandes et fournit des méthodes pour enregistrer, mettre à jour le statut des commandes, et récupérer l'historique des commandes pour un utilisateur spécifique. Cela nous a permis de suivre les transactions efficacement, en offrant une interface intuitive pour les administrateurs qui souhaitent gérer les commandes. Ce modèle est crucial pour assurer la transparence et la sécurité dans la gestion des commandes dans l'application.

Le modèle `Product.php` gère les données liées aux produits, offrant des fonctions pour rechercher par catégorie, obtenir les détails d'un produit spécifique, et mettre à jour le stock. Nous avons intégré ces fonctions pour permettre une gestion optimale des informations produit, en garantissant que les utilisateurs ont toujours accès à des données précises et actuelles. Cela est particulièrement important pour maintenir la qualité de l'expérience utilisateur, surtout lorsque les utilisateurs naviguent à travers les produits.

Le modèle `User.php` est responsable des données utilisateur, stockant des informations telles que les informations personnelles, le rôle de l'utilisateur (client ou administrateur), et l'état d'authentification. Nous avons intégré des mécanismes de sécurité, comme le cryptage des mots de passe, pour protéger les données des utilisateurs et assurer la sécurité de l'application. Cela nous permet de gérer les sessions de manière sécurisée et de garantir que les informations utilisateur sont toujours accessibles, mais protégées contre toute manipulation non autorisée.

Les vues, telles que `add_to_cart.php`, `cart.php`, `checkout.php`, `home.php`, `login.php`, `logout.php`, `product_detail.php`, `product_list.php`, `register.php`, `remove_from_cart.php`, et `success.php`, sont responsables de l'interface utilisateur. Elles fournissent des interfaces visuelles pour les différentes interactions utilisateur avec l'application. La vue `add_to_cart.php` permet aux utilisateurs d'ajouter des produits au panier, en s'assurant que toutes les informations utilisateur, comme la quantité ou la variante du produit, sont correctement affichées et mises à jour. `cart.php` offre une interface utilisateur pour gérer le contenu du panier, permettant aux utilisateurs de modifier les quantités, de supprimer des articles ou de calculer le total des achats, rendant ainsi l'expérience utilisateur plus interactive.

La vue `checkout.php` affiche le processus de paiement, en recueillant les informations nécessaires pour finaliser une commande. Elle guide les utilisateurs à travers les étapes de la commande, en s'assurant que toutes les informations requises, comme les coordonnées de livraison, sont fournies de manière claire et organisée. `home.php` présente une liste des produits ou des catégories populaires, offrant un point d'entrée central pour l'utilisateur et permettant une navigation facile à travers les différentes sections du site.

`login.php` et `register.php` sont cruciales pour la gestion des utilisateurs. Elles fournissent les interfaces pour se connecter ou s'inscrire, en vérifiant les informations saisies et en garantissant que les utilisateurs sont redirigés vers les pages appropriées selon leur état d'authentification. Nous avons également intégré des mécanismes de sécurité dans ces vues pour protéger les informations utilisateur et éviter toute manipulation malveillante des données d'authentification.

La section d'administration de l'application, avec des fichiers tels que `admin_add_product.php`, `admin_dashboard.php`, `admin_delete_order.php`, `admin_delete_product.php`, `admin_edit_product.php`, `admin_manage_categories.php`, `admin_manage_orders.php`, `admin_manage_products.php`, `admin_navbar.php`, `admin_view_order.php`, `delete_category.php`, `edit_category.php`, et `index.php`, permet aux administrateurs d'avoir un contrôle complet sur les produits, les commandes et les catégories. Chaque fichier d'administration est conçu pour offrir une interface utilisateur spécifique qui permet aux administrateurs d'ajouter, de modifier ou de supprimer des données. `admin_add_product.php` permet aux administrateurs d'ajouter de nouveaux produits à la base de données, en s'assurant que toutes les informations entrantes sont validées et conformes aux règles définies par l'application. `admin_manage_products.php` et `admin_manage_categories.php` offrent une interface pour gérer les produits et les catégories, permettant des actions comme la mise à jour des informations de produit ou la modification des catégories existantes.

Les fichiers d'inclusion, tels que `header.php`, `footer.php`, `functions.php`, `navbar.php`, et `validate_input.php`, fournissent des composants réutilisables à travers le site. `header.php` et `footer.php` standardisent les éléments communs de chaque page, comme le logo, les liens de navigation, et les informations de contact, tandis que `functions.php` contient des fonctions pour la validation des données utilisateur, assurant que toutes les entrées respectent les règles définies par l'application pour prévenir les manipulations malveillantes des données. `validate_input.php` contient des fonctions pour vérifier la validité des données saisies par les utilisateurs, comme la vérification des formats de texte, des adresses e-mail, ou des mots de passe. Cela nous a permis de maintenir une sécurité rigoureuse et de garantir que toutes les données entrantes respectaient les normes de l'application, assurant ainsi la protection des informations utilisateur.

En résumé, l'application que nous avons développée est un exemple d'organisation et de structuration efficace des différentes fonctionnalités d'un site web en utilisant l'architecture MVC. Chaque composant – les fichiers de configuration, les contrôleurs, les modèles, les vues, les fichiers d'administration, et les fichiers d'inclusion – a été soigneusement conçu pour répondre aux besoins des utilisateurs et garantir une expérience utilisateur fluide et sécurisée. Grâce à cette architecture, nous avons pu maintenir une séparation claire des préoccupations, ce qui facilite les mises à jour, la maintenance et l'évolution de l'application. Cette approche a permis de créer un site web robuste, sécurisé et évolutif, capable de répondre aux besoins changeants des utilisateurs et des exigences de gestion des données.