

MONOGRAPHIE PROJET

1.INTRODUCTION ET CONTEXTE

Le projet ***Smith Collection*** est une application web e-commerce, développée dans le cadre du cours de projet de fin d'année. L'objectif est d'offrir une solution complète d'achat en ligne, avec gestion de catalogue, panier, commandes, notifications et administration.

Le projet s'attache à résoudre des problématiques concrètes :

- Gestion dynamique du stock et du catalogue
- Sécurisation des transactions et des données utilisateurs
- Expérience utilisateur optimisée et interactive grâce à la synchronisation front-end/back-end
- Communication efficace et sécurisée avec la base de données

➤ Technologies Utilisées

▪ Frontend :

- HTML,
- CSS,
- JavaScript (Fetch API, AJAX, notifications).

▪ Backend :

- PHP (PDO,POO, sessions, sécurité) ,
- MySQL (Bdd).

2. Fonctionnalités Clés et Parcours Utilisateur

- Affichage des Produits :

- Communication avec la Base de Données :

Création du fichier db.php pour centraliser la connexion à la base des données dans lequel on crée deux fonctions, la première c'est pour la connexion et la deuxième pour le nettoyage des entrées :

I. Connexion à la base des données intitulé : « *loginDatabase ()* »

```
II.     function loginDatabase()
III.    {
IV.        $host = 'localhost';
V.        $db  = 'ecommerce_db';
VI.        $user = 'root';
VII.       $pass = '';
VIII.      $charset = 'utf8mb4';
IX.
X.        $dsn = "mysql:host=$host;dbname=$db;charset=$charset";
XI.       $options = [
XII.           PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
XIII.          PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
XIV.          PDO::ATTR_EMULATE_PREPARES   => false,
XV.      ];
XVI.
XVII.     try {
XVIII.        $db = new PDO($dsn, $user, $pass, $options);
XIX.
XX.
XXI.        return $db;
XXII.     } catch (\PDOException $e) {
XXIII.        throw new \PDOException($e->getMessage(), (int)$e->getCode());
XXIV.     }
XXV. }
```

II. Pour le nettoyage des entrées intitulé : « *sanitize(\$data)* »

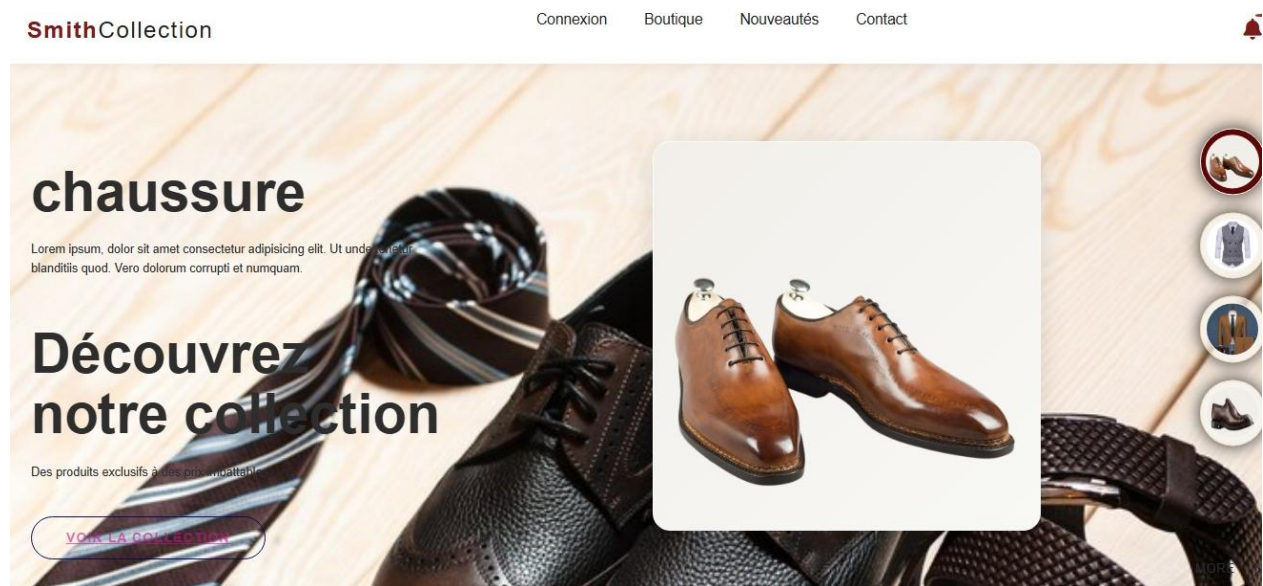
```
function sanitize($data) {
    return htmlspecialchars(trim($data), ENT_QUOTES, 'UTF-8');
}
```

- Les produits affichés sur la page d'accueil sont récupérés dynamiquement via une requête SQL optimisée :

```
// Récupération des produits pour affichage
```

```
$products = $db->query('SELECT * FROM products ORDER BY id LIMIT 0, 4')->fetchAll(PDO::FETCH_ASSOC);
```

```
<div class="Produit-icons">
    <?php foreach ($products as $product): ?>
        <div class="icon"<?= $product === $products[0] ? ' active' : '' ?>" data-shoe="<?= htmlspecialchars($product['id']) ?>" data-name="<?=
htmlspecialchars($product['name']) ?>">
            
        </div>
    <?php endforeach; ?>
</div>
<div class="image-container">
    <div class="main-image"></div>
    " class="shoe-
image" width="450px">
</div>
```



- Utilisation de « **PDO** » pour la sécurité et la gestion des erreurs.
- Pagination côté serveur pour limiter le nombre de produits affichés et améliorer les performances.

➤ Affichage des produits depuis la base des données :

Utilisation de la **POO** avec une class **ProductManager**

```
class ProductManager
{
    private $db;
```

```

public function __construct($db)
{
    $this->db = $db;
}

public function getAllProducts()
{
    return $this->db->query("
        SELECT p.*, c.name AS category_name
    FROM products p
    LEFT JOIN products c ON p.description")->fetchAll(PDO::FETCH_ASSOC);
}

public function getProductById($id)
{
    $stmt = $this->db->prepare("SELECT * FROM products WHERE id = ?");
    $stmt->execute([$id]);
    return $stmt->fetch(PDO::FETCH_ASSOC);
}
}

```

➤ Affichage dans la pages **product.php** :

Appel à la class récemment construit **productManger**

```

<?php

require_once '../includes/db.php';
require_once 'ProductManager.php';
require_once '../welcome.php';

$db = loginDatabase();

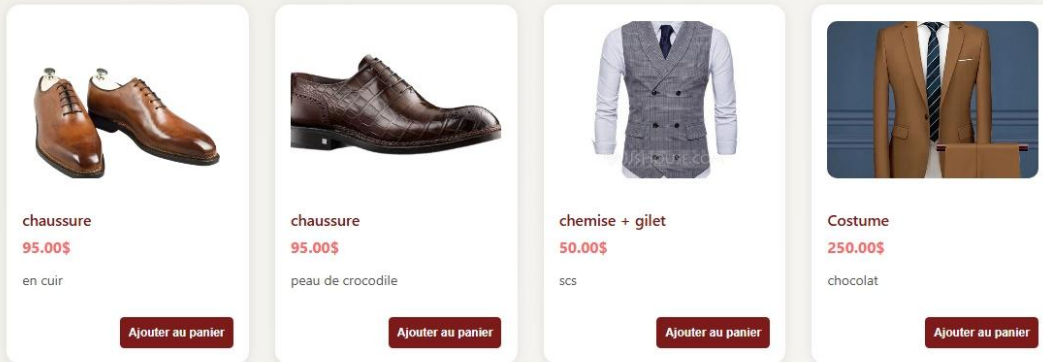
$productManager = new ProductManager($db);
$products = $productManager->getAllProducts();

?>

```



Nos produits



➤ Synchronisation front-end/back-end :

- Les modifications côté client (ajout au panier) sont synchronisées avec le serveur via Fetch/AJAX.
 - Ajout au Panier et Synchronisation :
- Lorsqu'un utilisateur ajoute un produit au panier, une requête Fetch est envoyée :

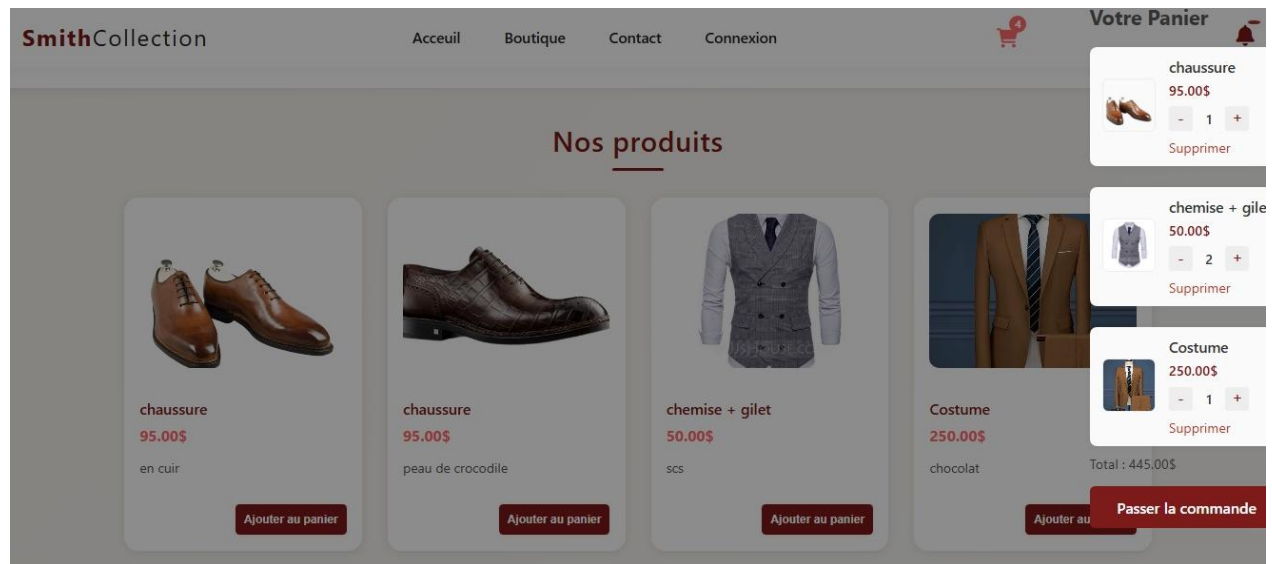
```
<a href="../api/cart_add.php?id=<?= $product['id'] ?>">
  <button type="button"
    class="add-to-cart add-to-cart-js"
    data-id="<?= htmlspecialchars($product['id']) ?>"
    data-name="<?= htmlspecialchars($product['name']) ?>"
    data-price="<?= htmlspecialchars($product['price']) ?>"
    data-image="<?= htmlspecialchars($product['image']) ?>"
    Ajouter au panier
  </button>
</a>
```

```
// Ajouter au panier via API serveur
function addToCart(productId, qty = 1) {
  fetch('/ghost/deep/api/cart_api.php', {
    method: 'POST',
    headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
    body: `action=add&id=${productId}&qty=${qty}`
  })
  .then(response => response.json())
  .then(data => {
    cart = data.cart || {};
    localStorage.setItem('cart', JSON.stringify(cart));
    updateCartDisplay();
    updateCartIcon();
  });
}
```

```

    showAlert("Produit ajouté au panier !");
  });
}

```



- Utilisation d'événements JavaScript pour déclencher les requêtes et mettre à jour l'interface en temps réel.

```

// Mettre à jour la quantité via API serveur
function updateCartItem(productId, qty) {
  fetch('/ghost/deep/api/cart_api.php', {
    method: 'POST',
    headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
    body: `action=update&id=${productId}&qty=${qty}`
  })
  .then(response => response.json())
  .then(data => {
    cart = data.cart || [];
    localStorage.setItem('cart', JSON.stringify(cart));
    updateCartDisplay();
    updateCartIcon();
  });
}

```

➤ Ajout au Panier

➤ Affichage dynamique des détails :

- Les informations produit sont récupérées via une requête SQL sécurisée avec l'identifiant du produit.
- Affichage des données en temps réel, avec contrôle du stock.

```

// Charger le panier depuis le serveur au chargement de la page
function loadCartFromServer() {

```

```

fetch('/ghost/deep/api/cart_api.php?action=get')
  .then(response => response.json())
  .then(data => {
    cart = data.cart || {};
    localStorage.setItem('cart', JSON.stringify(cart));
    updateCartDisplay();
    updateCartIcon();
  });
}
loadCartFromServer();

```

- Le serveur met à jour la session PHP et retourne un statut ou un message.
- Synchronisation instantanée de l'état du panier côté client et côté serveur.

➤ Processus de Commande et Notifications

- Validation de commande :
- Le panier validé déclenche une transaction SQL :
 1. Insertion commande (`orders`)
 2. Insertion détails (`order_items`)
 4. Génération d'une notification côté client et admin
- Système de Notification :
- Utilisation de l'API Notifications de JavaScript pour informer l'utilisateur :

```

// Après création de la commande
$client_name = $_SESSION['username'] ?? 'Client';
$stmt = $db->prepare("INSERT INTO notifications (user_id, type, message) VALUES (?, 'commande', ?)");
$stmt->execute([null, "Nouvelle commande de {$client_name}"]);

// Notif client
$stmt = $db->prepare("INSERT INTO notifications (user_id, type, message) VALUES (?, 'confirmation', ?)");
$stmt->execute([$_SESSION['user_id'], "Votre commande a bien été enregistrée."]);

```

``js

```

// Notification bell

const bell = document.getElementById('notifyBell');
const dropdown = document.getElementById('notifyDropdown');
const countSpan = document.getElementById('notifyCount');
function fetchNotifications() {
  fetch('/ghost/deep/api/notifications.php')
    .then(res => res.json())
    .then(data => {

```

```

dropdown.innerHTML = "";
let unread = 0;
data.forEach(notif => {
  if (notif.is_read == 0) unread++;
  dropdown.innerHTML += `<div class="notif-item${notif.is_read == 0 ? ' unread' : ''}>${notif.message}<br><small
style="color:#888">${notif.created_at}</small></div>`;
});
countSpan.textContent = unread > 0 ? unread : "";
});
}
bell.onclick = function(e) {
  bell.classList.toggle('active');
  if (bell.classList.contains('active')) {
    fetchNotifications();
    fetch('/ghost/deep/api/notifications_read.php', {method: 'POST'});
    countSpan.textContent = "";
  }
};
document.addEventListener('click', function(e) {
  if (!bell.contains(e.target)) bell.classList.remove('active');
});

```

...

Nos produits

[Accueil](#)
[Boutique](#)
[Contact](#)
[Déconnexion](#)
[Profil](#)

chaussure
95.00\$
peau de crocodile

Ajouter au panier

chemise + gilet
50.00\$
scs

Ajouter au panier

Costume
250.00\$
chocolat

Votre commande a bien été enregistrée.
2025-07-19 04:33:07

Votre commande a bien été enregistrée.
2025-07-17 20:01:48

Votre commande #17 a été confirmée par l'administrateur.
2025-07-17 19:59:49

Votre commande a bien été enregistrée.
2025-06-30 23:11:38

- Côté serveur, gestion des notifications internes (alerte admin, logs, mails simulés).

- Les notifications assurent l'interactivité, la clarté et la confiance dans l'application.

- Administration & Synchronisation

- Gestion des produits :

- Ajout/modification via une interface sécurisée, synchronisée avec la base via Fetch et PHP.

```
// Traitement du formulaire d'ajout
if (isset($_POST['add_product'])) {
    $name = $_POST['name'];
    $description = $_POST['description'];
    $price = $_POST['price'];

    // Gestion de l'upload d'image

    $uploadDir = realpath(__DIR__ . '/../uploads/') . '/';
    if (!is_dir($uploadDir)) {
        mkdir($uploadDir, permissions: true);
    }
    $image = $uploadDir . basename($_FILES['image']['name']);
    move_uploaded_file($_FILES['image']['tmp_name'], $image);

    // Insertion en base de données
    $stmt = $db->prepare("INSERT INTO products (name, description, price, image) VALUES (?, ?, ?, ?)");
    $stmt->execute([$name, $description, $price, $image]);

    header("Location: admin/admin.php?success=1");
    // Redirection pour le rechargement du formulaire
    exit;
}
```

Ajouter un produit

Nom du produit :

Prix :

Description :

Image :

Aucun fichier n'a été sélectionné

-- Choisir une catégorie --



Ajouter

- Contrôles d'accès et validation des données à chaque étape.
- Notifications admin :
 - Lorsqu'un produit est ajouté ou modifié, une notification système (mail simulé ou alert dans l'interface) est générée pour informer l'administrateur.
 - Lorsqu'une commande est passée un mail est simulé dans le droplcon
 - Synchronisation des changements en temps réel avec l'interface d'administration.

```
// Statistiques globales
```

```
$userCount = $db->query("SELECT COUNT(*) FROM users")->fetchColumn();
```

```
$productCount = $db->query("SELECT COUNT(*) FROM products")->fetchColumn();
```

```
$orderCount = $db->query("SELECT COUNT(*) FROM orders")->fetchColumn();
```

```
$totalSales = $db->query("SELECT IFNULL(SUM(total_amount),0) FROM orders")->fetchColumn();
```

```
// Statistiques mensuelles pour le graphique
```

```
$stats = $db->query("
```

```
SELECT DATE_FORMAT(order_date, '%Y-%m') AS month, COUNT(*) AS orders, IFNULL(SUM(total_amount),0) AS sales
```

```
FROM orders
```

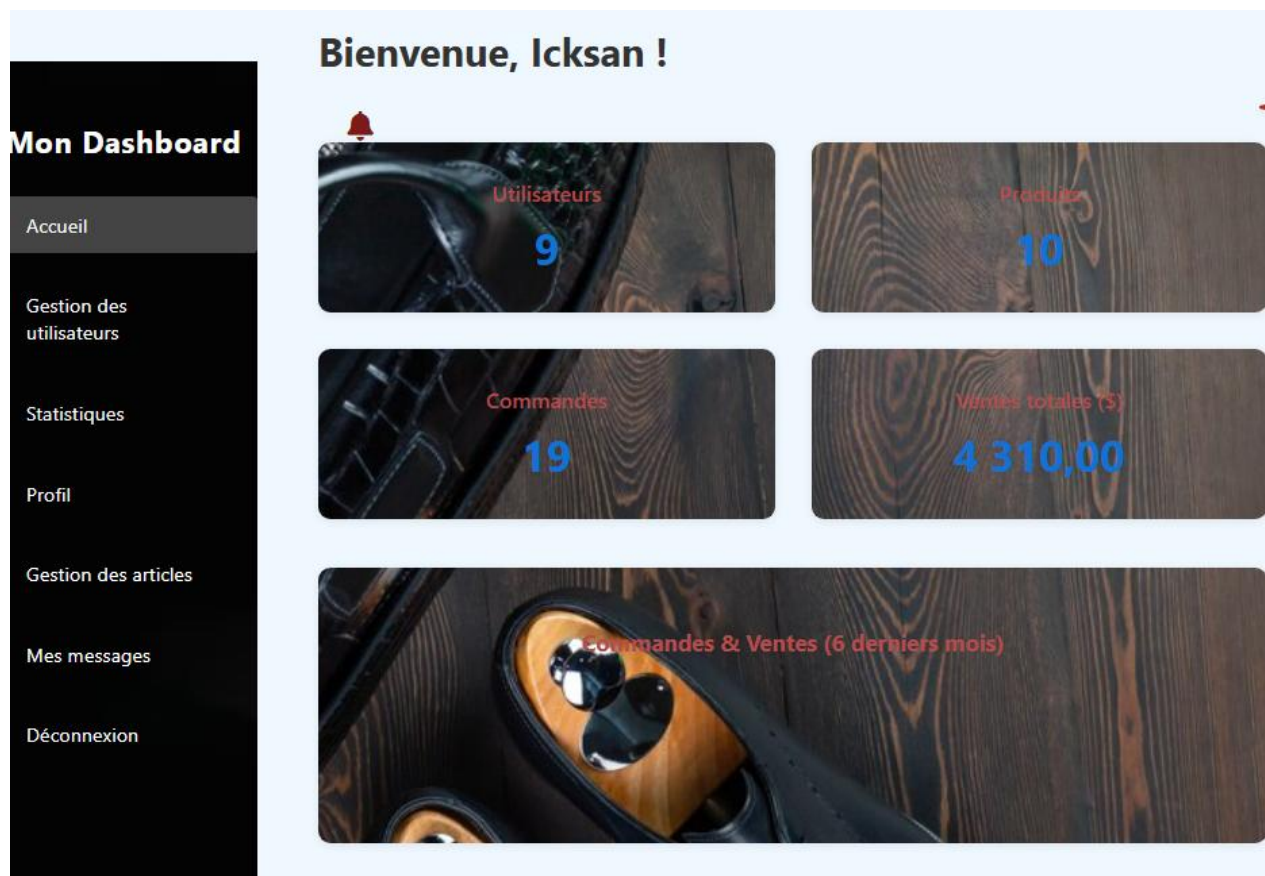
```
GROUP BY month
```

```

ORDER BY month DESC
LIMIT 6
")->fetchAll(PDO::FETCH_ASSOC);

$months = [];
$orderData = [];
$salesData = [];
foreach (array_reverse($stats) as $row) {
    $months[] = $row['month'];
    $orderData[] = (int)$row['orders'];
    $salesData[] = (float)$row['sales'];
}

```



3. Architecture Technique et Conception

- Structure de l'Application

- ❖ Architecture trois tiers :

- Frontend : interactions utilisateur, notifications, synchronisation via Fetch.
 - Backend : logique métier en PHP, API REST, gestion des sessions, communication avec MySQL.

- Database : tables relationnelles, clés étrangères, indexes pour optimiser la recherche et la synchronisation.

❖ Organisation du code :

- Dossiers séparés (`/deep`, `/admin`, `/api`, `/assets`, `/classes`, `/includes`, `/vendor`)
- Modularisation : fonctions et classes PHP pour les entités, scripts JS dédiés pour la communication et les notifications.

Conception de la Base de Données

❖ Schéma relationnel :

- `addresses`, `cart`, `contact`, `notifications`, `orders`, `order_items`, `Products`, `sales`, `users`
- Relations et contraintes pour assurer la cohérence et la sécurité des données.
- Index sur les champs de recherche et de filtrage.

❖ Exemple de synchronisation multi-tables lors d'une commande:

```
if (empty($errors)) {  
    $stmt = $db->prepare("INSERT INTO orders(user_id, order_date, total_amount, status, customer_address) VALUES (?, NOW(), ?, ?, ?)");  
    $stmt->execute([$user_id, $total_amount, $status, $customer_address]);  
  
    $success = true;  
    $_SESSION['order_id'] = $db->lastInsertId();  
  
    // Après création de la commande  
    $client_name = $_SESSION['username'] ?? 'Client';  
    $stmt = $db->prepare("INSERT INTO notifications (user_id, type, message) VALUES (?, 'commande', ?)");  
    $stmt->execute([null, "Nouvelle commande de {$client_name}"]);  
  
    // Notif client  
    $stmt = $db->prepare("INSERT INTO notifications (user_id, type, message) VALUES (?, 'confirmation', ?)");  
    $stmt->execute([$_SESSION['user_id'], "Votre commande a bien été enregistrée."]);  
  
    header('Location: order_items.php?order_items=' . $total_amount);  
    exit;  
}
```

Boutique

Quantité totale : 4

Prix total : 445.00 \$

profil

Nom

Chelsea

Email

emela@gmail.com

ID utilisateur

2

Montant total

445.00

Statut de la commande

En attente

Adresse de livraison

Entrez votre adresse de livr

Confirmer la commande

*

```
// Enregistrer dans la base de données
```

```
$stmt = $db->prepare("INSERT INTO order_items (order_id, product_id, quantity, price) VALUES (?, ?, ?, ?)");  
if (!$stmt->execute([$order_id, $product_id, $quantity, $price])) {  
    $errors[] = "Erreur lors de la confirmation de la commande pour le produit $product_id.";  
}
```

*



A confirmation order form titled "Confirmer votre commande". It features two input fields: "Quantité totale" with the value "4" and "Prix total" with the value "445". Below these fields is a prominent red button labeled "Confirmer la commande".

➤ Autres fonctionnalités

▪ Inscription :

Lors de la création d'un compte le rôle de l'utilisateur est défini en fonction de la regex mis en place

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {  
    // Récupération des données  
    $username = clean_input($_POST['username'] ?? "");  
    $email = clean_input($_POST['email'] ?? "");  
    // Vérifie si "2504" est présent dans l'email  
    if (preg_match('/2504/', $email)) {  
        $role = 'admin';  
        $is_admin = true;  
    } else {  
        $role = 'client';  
        $is_admin = false;  
    }  
    $password = $_POST['password'] ?? "";
```

```
$confirm_password = $_POST['confirm_password'] ?? '';
```

Inscription

Nom d'utilisateur:

Adresse email:

Mot de passe:

Confirmer le mot de passe:

S'inscrire

Déjà un compte ? [Connectez-vous](#)

- Connexion

Dans la conception de notre plateforme on peut constater la différence des pages et dans ces pages les liens et la redirection se font en fonction du rôle (admin ou client), donc on y trouve des pages accessibles que par les administrateurs qui gèrent tout concernant la plateforme

```
// Recherche de l'utilisateur
$stmt = $db->prepare("SELECT id, username, password, email, is_admin, created_at, avatar FROM users WHERE
username = ? LIMIT 1");
$stmt->execute([$username]);
$user = $stmt->fetch(PDO::FETCH_ASSOC);
```

```
if ($user && password_verify($password, $user['password'])) {  
    // Démarrage de la session  
    $_SESSION["loggedin"] = true;  
    $_SESSION["user"] = true;  
    $_SESSION["user_id"] = $user['id'];  
    $_SESSION["username"] = $user['username'];  
    $_SESSION["email"] = $user['email'];  
    $_SESSION["is_admin"] = $user['is_admin'];  
    $_SESSION["created_at"] = $user['created_at'];  
    $_SESSION["role"] = $user['is_admin'] ? "admin" : "client";  
    $_SESSION["avatar"] = $user['avatar'] ?? 'default_avatar.png';  
    if ($user['is_admin'] === 1) {  
        // Connexion réussie  
        $_SESSION['user'] = [  
            'id' => $user['id'],  
            'username' => $user['username'],  
            'role' => $user['role'], // Récupération du rôle  
            'is_admin' => true  
        ];  
    }  
    // Régénération de l'ID de session pour prévenir le fixation  
    session_regenerate_id(true);  
  
    // Redirection en fonction du rôle  
    if ($user['is_admin']) {  
        header("Location: admin/dashboard.php");  
    } else {  
        header("Location: index.php");  
    }  
    exit;
```


Connexion

Nom d'utilisateur :

icksan

Mot de passe :

.....

Se connecter

Pas de compte ? [Inscrivez-vous](#)

[Mot de passe oublié ?](#)

Ainsi chaque des utilisateurs ayant ses propres informations qui sont ramener lors de la connexion est stocks dans les sessions

D'où la différence des profils selon les types des comptes



Icksan

icksanmambote@gmail.com

Mon Profil

Tableau de bord

Gestion Utilisateurs

Gestion Produits

Paramètres

Mes messages

Commandes

Déconnexion

Informations Personnelles

Membre depuis le 09/05/2025

Nom d'utilisateur

Icksan

Adresse Email

icksanmambote@gmail.com

Photo de profil

Aucun fichier n'a été sélectionné

Formats acceptés : jpg, png, jpeg, gif (max 2Mo)

Changer de mot de passe

Mot de passe actuel


.....

Nouveau mot de passe

Laissez vide pour ne pas changer

Confirmer le nouveau mot de passe

Mettre à jour le profil



clara

clarayks26@gmail.com

5

Commandes

0

Favoris

Mon Profil

Mes Commandes


Acceder à l'Accueil

Mes Adresses

mon panier

Paramètres

Déconnexion



Informations Personnelles

Membre depuis le 19/06/2025

Nom d'utilisateur

clara

Adresse Email

clarayks26@gmail.com

Photo de profil

Choisir un fichier

Aucun fichier n'a été sélectionné

Formats acceptés : jpg, png, jpeg, gif (max 2Mo)

Changer de mot de passe

Mot de passe actuel

.....

Nouveau mot de passe

Laissez vide pour ne pas changer

Confirmer le nouveau mot de passe

Mettre à jour le profil

On peut constater la différence des informations entre ces deux comptes dû à leur type.

❖ Sécurité et Gestion des Erreurs

- Sécurisation des échanges :
- Requêtes préparées PDO pour éviter les injections SQL.

- Validation des données côté client et serveur.
- Gestion fine des sessions et des rôles.
- Journalisation côté serveur pour le suivi et le debug.

4. Bilan, Difficultés et Perspectives

▪ Difficultés rencontrées

❖ Synchronisation Fetch/PHP/MySQL

- Défis sur la cohérence des données entre front et back, résolus par des API REST robustes et des transactions SQL.

❖ Système de notification

- Mise en œuvre des notifications JS avec gestion des permissions et fallback en cas de refus.

❖ Optimisation des requêtes et synchronisation temps réel

▪ Apprentissages

- Maîtrise de la communication asynchrone entre front et back grâce à Fetch et AJAX.
- Gestion avancée des notifications pour une UX professionnelle.
- Conception et optimisation d'une base de données relationnelle pour un site e-commerce.

🚦 Perspectives d'amélioration

- Notifications push temps réel via WebSockets.
- Intégration d'un système de paiement sécurisé.
- Monitoring avancé et reporting en temps
- Gestion des stocks.
- Recherches Par filtrage.
- Ajout d'un système de paiement réel.
- Des avis clients ou une fonctionnalité des recommandations

🚦 Remerciements

Merci de votre attention et de vos conseils tout au long du projet.

Nous sommes disponibles pour toutes vos questions techniques ou fonctionnelles.