

Rapport de stage

Développement d'une Application Web de Gestion des Stocks et des Ventes pour MSDM Horizon

Etudiant : Alan TERRIER

Entreprise : MSDM HORIZON

Tuteur de stage : Thomas MAREAU

Dates : 10/06/2024 – 02/08/2024

The logo for Shoptapaire, featuring the brand name in a bold, black, sans-serif font.

L ' e x c l u s i v i t é

by MSDM HORIZON

Remerciements

Je tiens à exprimer ma profonde gratitude à **Monsieur Thomas MAREAU**, Président de MSDM Horizon, pour m'avoir offert l'opportunité de réaliser ce stage au sein de son entreprise. Monsieur MAREAU a su créer un environnement propice à l'apprentissage et à l'épanouissement, où j'ai pu développer non seulement mes compétences techniques, mais aussi à gérer des projets complexes.

Mes remerciements vont également à **Monsieur Chollet** et **Monsieur Garcia**, qui ont joué un rôle essentiel dans l'encadrement et la supervision de ce stage. Leur disponibilité a grandement facilité mon apprentissage. Grâce à leur encadrement, j'ai pu aborder les défis du projet avec confiance et rigueur, et acquérir des compétences précieuses pour mon futur scolaire et professionnel.

Je tiens à remercier l'ensemble de mes enseignants de l'IUT. Grâce à leur enseignement et à leur expertise, j'ai pu acquérir des compétences solides qui m'ont été indispensables durant mon stage. Leur enseignement m'a permis de relever les défis qui se sont présentés à moi. Je les remercie pour leur soutien continu, leur patience, et les précieux conseils qui ont grandement contribué à ma formation et à mon développement.

Je tiens aussi à exprimer ma reconnaissance à l'ensemble du personnel administratif de l'IUT, par leur efficacité et leur professionnalisme, ont contribué à créer un cadre de travail optimal.

Ce stage a été une expérience formatrice et enrichissante à bien des égards, et je suis très reconnaissant à toutes les personnes qui ont contribué à faire de cette expérience un succès. Ce rapport est le fruit d'une collaboration étroite et d'un apprentissage continu, c'est un témoignage de ma gratitude à tous ceux qui ont joué un rôle dans mon parcours durant ces deux mois d'été.

Résumé et mot clés :

Résumé

Ce stage a principalement porté sur la conception et le développement d'une application web de gestion des stocks et des ventes pour une entreprise. Le projet avait pour objectif de créer un outil performant capable de gérer efficacement les articles en stock, suivre les ventes, et générer des statistiques pour faciliter la prise de décision.

Les missions réalisées ont inclus la conception d'une base de données relationnelle, le développement d'un back-end en PHP selon le modèle MVC (Modèle-Vue-Contrôleur), et la création d'une interface utilisateur en HTML, CSS, et JavaScript. Parmi les fonctionnalités clés mises en place, on compte la gestion des utilisateurs, des articles en stock, des ventes, ainsi que l'implémentation de mesures de sécurité, telles que le hachage des mots de passe.

Le stage a également impliqué l'amélioration des performances, notamment en optimisant la base de données et en automatisant la mise à jour des statistiques via des triggers SQL. Enfin, des comparaisons avec des solutions existantes sur le marché, telles que BabaSpace, ont permis d'inspirer certaines fonctionnalités pour enrichir l'application.

Ce projet a permis de développer une application complète répondant aux besoins spécifiques de gestion de stocks et de ventes, tout en intégrant des aspects de performance, de sécurité, et d'ergonomie pour offrir une expérience utilisateur optimale.

Mots-clés

Gestion des stocks, Gestion des ventes, Application web, Modèle-Vue-Contrôleur (MVC), Base de données MySQL, HTML/CSS/JavaScript, Sécurité des données, Hachage de mots de passe, Triggers SQL, Statistiques de vente, Interface utilisateur, Développement web, Automatisation des processus, Optimisation des performances

Abstract

This internship primarily focused on the design and development of a web application for inventory and sales management for a company. The project's objective was to create an efficient tool capable of effectively managing items in stock, tracking sales, and generating statistics to facilitate decision-making.

The tasks carried out included designing a relational database, developing a back-end in PHP following the MVC (Model-View-Controller) model, and creating a user interface using HTML, CSS, and JavaScript. Key features implemented include user management, inventory management, sales tracking, as well as the implementation of security measures such as password hashing.

The internship also involved performance improvements, notably by optimizing the database and automating the updating of statistics through SQL triggers. Finally, comparisons with existing market solutions, such as BabaSpace, inspired certain features to enhance the application.

This project allowed for the development of a comprehensive application that meets the specific needs of inventory and sales management, while integrating aspects of performance, security, and ergonomics to provide an optimal user experience.

Keywords

Inventory management, Sales management, Web application, Model-View-Controller (MVC), MySQL database, HTML/CSS/JavaScript, Data security, Password hashing, SQL triggers, Sales statistics, User interface, Web development, Process automation, Performance optimization

Table des matières

Remerciements.....	2
Résumé et mot clés.....	3
Résumé	3
Mots-clés.....	3
Abstract.....	3
Keywords.....	4
Tables des figures.....	6
Glossaire.....	7
Contexte du stage.....	10
Présentation de l'entreprise	10
Enjeux du stage.....	11
Analyse des missions de stage	12
Analyse du sujet.....	12
Analyse de l'environnement technique	12
Analyse de l'existant	13
Spécifications techniques (ou backlog)	14
Rapport technique.....	16
Conception.....	16
Réalisation.....	20
Gestion des erreurs et des exceptions	28
Focus sur des parties techniques	29
Gestion des graphiques avec Chart.js	29
Configuration de l'application : le fichier `config.php`	35
Création d'un Trigger	39
Récupération des statistiques pour les afficher à jour.....	45
Un site responsif	50
Rapport d'activité	52
Proposition d'évolution de l'application.....	54
Méthodologie et organisation du projet	55
Conclusion.....	58
Visa du maître de stage.....	58
Bibliographie	59
Annexes.....	60

Tables des figures

Diagramme 1 : Architecture MVC de l'application.....	17
Diagramme 2 : Modélisation E/A de la base de données.....	18
Figure 1 : Code la fonction de construction d'un objet Produit.....	20
Figure 2 : Fonction connexion du ControleurUtilisateur.....	21
Figure 3 : Différentes fonction de ControleurVente	22
Figure 4 : Table Membres.....	23
Figure 5 : Table Inventaire	23
Figure 6 : Table Ventes	24
Figure 7 : Table Statistiques	25
Figure 8 : Table Possede.....	25
Figure 9 : Table Vends	26
Diagramme 3 : Préchargement des données	30
Figure 10 : ControleurDashboard.....	31
Figure 11 : Création d'un graphique	32
Figure 12 : Page web Dashboard	33
Figure 13 : Informations de connexion.....	35
Figure 14 : Configuration globale	36
Figure 15 : PEPPER.....	37
Diagramme 4 : Interaction du fichier `config.php` avec les composants de l'application.....	38
Figure 16 : Déclarations de Variables.....	39
Figure 17 : Récupération des Données.....	40
Figure 18 : Calcul et mise à jour des statistiques	41
Figure 19 : Mise à jour des données.....	41
Diagramme 5 : Processus du trigger `update_statistiques_after_vente`	42
Figure 20 : Récupération des statistiques par utilisateurs.....	45
Figure 21 : ControleurDashboard.....	46
Figure 22 : Vue Dashboard avec chart.js.....	47
Diagramme 6 : Affichage des statistiques.....	49

Glossaire

- **Rainbow Table Attack :**

Une technique utilisée pour craquer les mots de passe hachés en utilisant des tables pré-calculées de hachages pour des mots de passe courants. Ces tables permettent de retrouver rapidement le mot de passe en clair à partir de son hachage.

- **MVC (Modèle-Vue-Contrôleur) :**

Un modèle d'architecture logicielle qui divise une application en trois composantes principales : le Modèle (gestion des données), la Vue (interface utilisateur) et le Contrôleur (logique de gestion).

- **Modèle :**

Dans le contexte de l'architecture MVC, le modèle représente la structure des données et la logique métier de l'application. Il gère les données, les règles d'affaires et les interactions avec la base de données.

- **Vue :**

La composante de l'architecture MVC qui gère l'affichage des données à l'utilisateur. La vue est responsable de la présentation des informations, généralement sous forme d'interface graphique.

- **Contrôleur :**

La composante de l'architecture MVC qui gère les interactions entre l'utilisateur, le modèle, et la vue. Le contrôleur reçoit les entrées de l'utilisateur, interagit avec le modèle pour traiter les données, puis met à jour la vue.

- **Chart.js :**

Une bibliothèque JavaScript open-source utilisée pour créer des graphiques interactifs dans les applications web. Elle est couramment utilisée pour représenter visuellement des données sous forme de graphiques.

- **CRUD :**

Un acronyme pour Create, Read, Update, Delete. Ce sont les quatre opérations de base effectuées sur les données dans une application web ou une base de données.

- **Mot de passe haché :**

Un mot de passe qui a été transformé en une chaîne de caractères de longueur fixe à l'aide d'une fonction de hachage. Le hachage est un procédé irréversible, utilisé pour stocker les mots de passe de manière sécurisée.

- **Trigger :**

Un composant de la base de données qui exécute automatiquement une action prédéfinie en réponse à un événement spécifique, comme l'insertion, la mise à jour ou la suppression d'une ligne dans une table.

- **Front-end :**

La partie visible et interactive d'une application web, avec laquelle l'utilisateur interagit directement. Elle inclut le design, la mise en page, et la logique de l'interface utilisateur, souvent développée en HTML, CSS et JavaScript.

- **Back-end :**

La partie de l'application qui gère la logique métier, les bases de données et la communication avec le front-end. Le back-end traite les requêtes, stocke les données, et exécute les opérations côté serveur.

- **Poivre :**

Un élément de sécurité supplémentaire ajouté à un mot de passe avant son hachage. Contrairement au sel, le poivre est une valeur secrète et non stockée dans la base de données, rendant plus difficile la réussite d'attaques par rainbow table.

- **Index base de données :**

Une structure de données dans une base de données qui améliore la vitesse de récupération des enregistrements. Les index permettent un accès rapide aux données en créant des points d'entrée dans les tables, similaires à un index dans un livre.

- **SQL (Structured Query Language) :**

Un langage standardisé utilisé pour interagir avec des bases de données relationnelles. SQL permet de créer, modifier et interroger des bases de données, ainsi que de gérer les accès aux données.

- **Responsive :**

Un terme utilisé pour décrire un site web ou une application dont le design et la mise en page s'adaptent automatiquement à différentes tailles d'écran et types de dispositifs (ordinateurs, tablettes, smartphones), assurant une expérience utilisateur cohérente sur tous les supports.

Contexte du stage

Présentation de l'entreprise

Nom de l'entreprise : MSDM Horizon

Forme juridique : Société par actions simplifiée (SAS)

Numéro Siren : 979471539

Numéro Siret : 97947153900010 (siège de l'entreprise)

Numéro TVA intracommunautaire : FR31979471539

Date d'immatriculation : 12/09/2023

Adresse : 1080 Route de Dieulefit, 26160 La Bégude-de-Mazenc, Drôme, France

Code NAF / APE : 4642Z (commerce de gros (commerce interentreprises) d'habillement et de chaussures)

Président : Thomas MAREAU

Activité :

MSDM Horizon est une entreprise spécialisée dans le commerce de gros et de détail d'articles de chaussures, de prêt-à-porter et d'accessoires de mode. Elle exploite également un site internet pour la vente en ligne de ces articles. L'entreprise a été fondée récemment, le 12 septembre 2023, et est implantée à La Bégude-de-Mazenc, une commune dans la Drôme.

Mission de l'entreprise :

L'entreprise se concentre sur la distribution d'articles de mode via des canaux de vente en gros et en détail, incluant une plateforme en ligne pour toucher un public plus large. Cette double approche permet à MSDM Horizon de diversifier ses revenus et d'accroître sa présence sur le marché de la mode et des sneakers.

Marché cible :

Le marché de MSDM Horizon comprend les détaillants de mode, les boutiques spécialisées, ainsi que les consommateurs finaux atteints par le biais de la plateforme en ligne. L'entreprise vise à fournir des articles de mode tendance et de qualité à des prix

compétitifs, attirant ainsi une clientèle variée allant des jeunes adultes aux professionnels à la recherche d'accessoires de mode rare.

Dimension commerciale :

MSDM Horizon génère son chiffre d'affaires à travers la vente de gros et de détail de ses produits de mode. La vente en ligne constitue une part importante de son modèle économique, permettant d'atteindre une clientèle nationale et potentiellement internationale. Le profil de ses clients inclut des boutiques de mode indépendantes ainsi que des clients particuliers achetant via le site internet de l'entreprise.

Enjeux du stage

Les enjeux pour l'entreprise en accueillant un stagiaire incluent l'apport de nouvelles idées et perspectives pour optimiser les processus de vente en ligne, l'amélioration de la gestion des stocks. Le stage permettra de lier les missions confiées aux besoins spécifiques de l'entreprise en termes de croissance et d'efficacité opérationnelle.

Analyse des missions de stage

Analyse du sujet

Le projet principal consiste en la conception et le développement d'une application web de gestion des stocks et des ventes pour une entreprise. Cette application a pour but de gérer efficacement les articles en stock, de suivre les ventes, et de générer des statistiques pour aider à la prise de décision. Les tâches confiées incluent :

- Conception de la base de données.
- Développement d'un back-end en PHP utilisant le modèle MVC (Modèle-Vue-Contrôleur).
- Création de l'interface utilisateur en HTML, CSS et JavaScript.
- Mise en place de fonctionnalités de gestion des utilisateurs, d'inventaire, de ventes et de statistiques.
- Implémentation de sécurités telles que le hachage et le salage des mots de passe.
- Développement de triggers SQL pour automatiser les mises à jour des statistiques.

Analyse de l'environnement technique

Description de l'architecture logicielle

L'application est développée selon une architecture MVC, où :

- **Modèle (Model)** : Contient la logique métier et gère les données. Les données sont stockées dans une base de données MySQL. Chaque entité de la base de données (par exemple, Membres, Articles, Ventes) a un modèle correspondant qui définit ses attributs et ses méthodes.
- **Vue (View)** : Gère l'affichage des données et l'interaction utilisateur. Les vues sont développées en HTML, CSS et JavaScript pour offrir une interface utilisateur intuitive et réactive.
- **Contrôleur (Controller)** : Gère la communication entre les modèles et les vues. Les contrôleurs reçoivent les entrées de l'utilisateur via les vues, appellent les méthodes appropriées des modèles et retournent les données à la vue pour affichage.

Principales fonctionnalités existantes et celles à modifier

Fonctionnalités à modifier/ajouter :

- Gestion des utilisateurs : création de comptes, authentification, hachage et salage des mots de passe.

- Gestion des articles : ajout, modification, suppression et consultation des articles en stock.
- Gestion des ventes : enregistrement des ventes, suivi des articles vendus.
- Génération de statistiques : nombre d'articles vendus, bénéfices, chiffre d'affaires.
- Mise en place de triggers SQL pour automatiser la mise à jour des statistiques après chaque vente.
- Amélioration de l'interface utilisateur pour une meilleure expérience utilisateur.
- Ajout de fonctionnalités de filtrage et de recherche dans les vues inventaire et ventes.
- Renforcement de la sécurité avec l'ajout de validations et de protections contre les injections SQL.

Analyse de l'existant

État actuel du projet/produit

Le projet en est en début de développement. Les principales entités et relations de la base de données ont été définies, et les modèles correspondants ont été créés. Les contrôleurs de base sont en place et permettent les opérations CRUD (Create, Read, Update, Delete) sur les principales entités.

Évaluation des performances et identification des points à améliorer

- **Performances de la base de données** : La structure actuelle de la base de données est correcte, mais l'indexation des colonnes fréquemment utilisées pour les recherches pourrait améliorer les temps de réponse.
- **Sécurité** : Le hachage et le salage des mots de passe sont implémentés.
- **Interface utilisateur** : Les interfaces ne sont pas encore fonctionnelles.
- **Automatisation des statistiques** : Actuellement, les statistiques doivent être mises à jour manuellement. L'implémentation de triggers SQL pour automatiser cette mise à jour est en cours.

Inspiration de ce qu'il se fait déjà : *BabaSpace* :

BabaSpace est une marketplace de gestion des stocks où les utilisateurs peuvent créer des comptes pour lister leur stock, indiquer leurs ventes et suivre toutes leurs transactions. L'application offre également la possibilité à d'autres utilisateurs d'accéder aux stocks de tous les membres pour acheter directement des articles. BabaSpace propose des fonctionnalités telles que :

- **Scan des paires** : Ajout direct des articles à l'inventaire en les scannant.

- **Suivi des côtes en temps réel** : Mise à jour des valeurs des articles en fonction du marché.
- **Calcul automatique des bénéfices** : Suivi des marges et des profits réalisés.
- **Suivi des colis** : Suivi de l'acheminement des articles vendus.
- **Mise en relation avec des entreprises certifiées** : Facilitation des ventes avec des acheteurs professionnels.

BabaSpace est disponible sur iOS et Android, offrant une solution mobile complète pour la gestion des stocks et des ventes. Elle permet une gestion centralisée et optimisée des inventaires, rendant le processus de vente plus transparent et efficace. Le modèle et les fonctionnalités de BabaSpace sont similaires à celles que nous développons, avec un accent sur la convivialité et l'automatisation des processus critiques.

Spécifications techniques (ou backlog)

Le cahier des charges initial spécifie les besoins suivants :

- Gestion des utilisateurs avec des rôles (administrateurs et utilisateurs).
- Gestion des articles en stock avec des détails complets (nom, marque, taille, prix d'achat, prix de vente, etc.).
- Suivi des ventes et génération de rapports statistiques.
- Interface utilisateur conviviale et réactive.

Spécifications fonctionnelles et non fonctionnelles

Spécifications fonctionnelles :

- **Gestion des utilisateurs** : Création, modification, suppression et authentification des utilisateurs. Gestion des rôles et permissions.
- **Gestion des articles** : Ajout, modification, suppression et consultation des articles en stock.
- **Gestion des ventes** : Enregistrement des ventes avec détails complets. Suivi des articles vendus.
- **Statistiques** : Génération automatique des statistiques de vente, bénéfices et chiffre d'affaires.

Spécifications non fonctionnelles :

- **Performance** : L'application doit répondre rapidement aux requêtes des utilisateurs et gérer efficacement les opérations de base de données.
- **Sécurité** : Les données des utilisateurs doivent être protégées avec des techniques de hachage et salage des mots de passe, et des mesures contre les injections SQL.

- **Accessibilité** : L'interface utilisateur doit être accessible et utilisable sur différents navigateurs et appareils. Description des besoins fonctionnels et non fonctionnels

Besoins fonctionnels :

- **Interface utilisateur intuitive** : Les utilisateurs doivent pouvoir naviguer facilement dans l'application, avec des options de filtrage et de recherche efficaces.
- **Automatisation des processus** : Mise à jour automatique des statistiques après chaque vente grâce aux triggers SQL.
- **Rapports détaillés** : Génération de rapports détaillés sur les ventes, les bénéfices et le stock.

Besoins non fonctionnels :

- **Sécurité renforcée** : Protection des données sensibles des utilisateurs, validation des entrées et gestion des sessions.
- **Haute disponibilité** : L'application doit être disponible en permanence pour les utilisateurs.
- **Maintenance facile** : Le code doit être bien documenté et structuré pour faciliter la maintenance et les futures évolutions.

Rapport technique

Conception

La conception de l'application de gestion des stocks et des ventes s'est articulée autour de plusieurs principes clés : modularité, sécurité, et facilité d'utilisation. L'application devait être construite selon une architecture MVC (Modèle-Vue-Contrôleur), ce qui permet de séparer les responsabilités entre la gestion des données, la logique métier, et la présentation à l'utilisateur (vues).

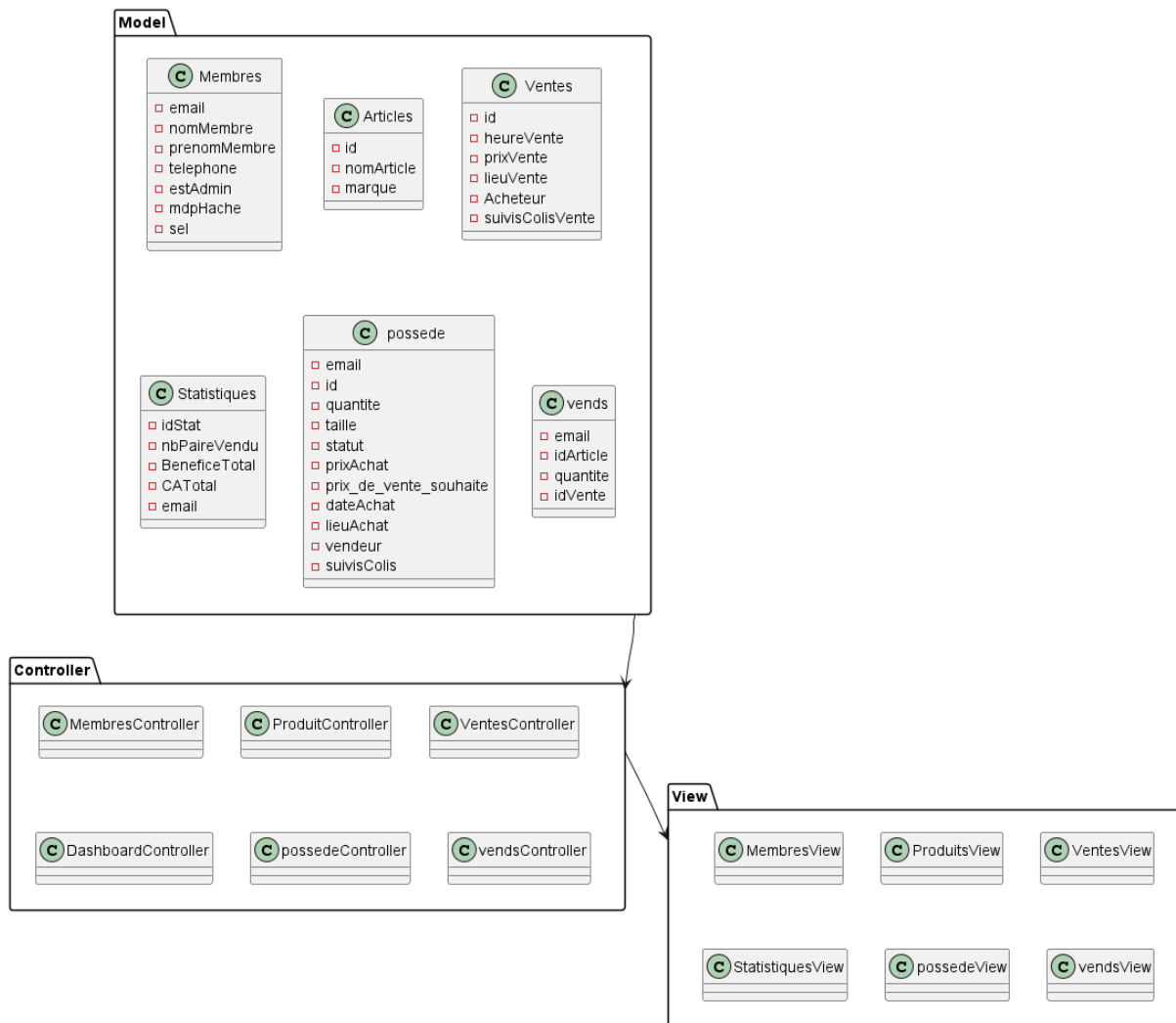
Architecture MVC :

L'architecture MVC a été choisie pour plusieurs raisons :

- **Séparation des préoccupations** : Le modèle MVC permet de séparer clairement la logique métier (Model), la présentation (View), et le contrôle des flux de données (Controller). Cette séparation facilite la maintenance et l'évolution de l'application.
- **Réutilisabilité** : En séparant les différentes couches, il devient possible de réutiliser des composants, tels que les modèles ou les vues, dans d'autres parties de l'application sans avoir à dupliquer le code.
- **Testabilité** : La structure modulaire rend le code plus facile à tester, car chaque composant peut être testé indépendamment des autres.

Le diagramme ci-dessous illustre l'architecture générale de l'application :

Diagramme 1 : Architecture MVC de l'application.



Modélisation de la base de données :

La base de données a été conçue pour une gestion des stocks et des ventes. Un schéma de base de données relationnelle a été créé, comprenant les tables suivantes :

- **Membres** : Contient les informations des utilisateurs, tels que l'email, le nom, le prénom, le rôle (administrateur ou utilisateur), et les informations de sécurité (mot de passe haché, sel).

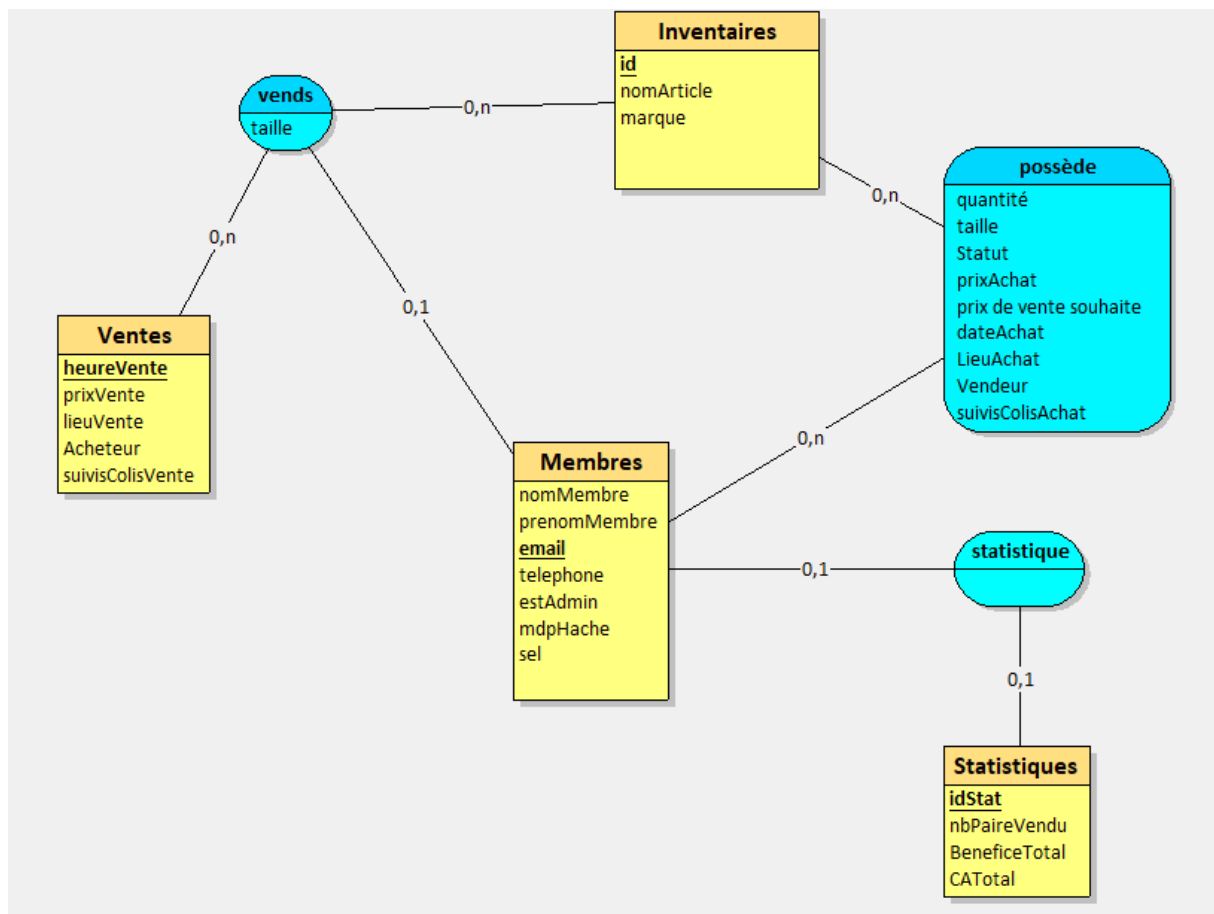
- **Articles** : Stocke les informations sur les articles en inventaire, y compris le nom, la marque, et l'identifiant unique.

- **Ventes** : Enregistre les transactions de vente, incluant l'heure de la vente, le prix de vente, le lieu, et l'acheteur.

- **Statistiques** : Contient les statistiques des ventes, telles que le nombre d'articles vendus, le bénéfice total, et le chiffre d'affaires.

Le diagramme entité-association (E/A) suivant montre la structure des tables et les relations entre elles :

Diagramme 2 : Modélisation E/A de la base de données.



```
ALTER TABLE Inventaires ADD CONSTRAINT positive_prixAchat CHECK (prixAchat >= 0);  
ALTER TABLE Inventaires ADD CONSTRAINT positive_prixVenteSouhaite CHECK (prixVenteSouhaite >= 0);  
ALTER TABLE Ventes ADD CONSTRAINT positive_prixVente CHECK (prixVente >= 0);  
ALTER TABLE Statistiques ADD CONSTRAINT positive_nbPaireVendu CHECK (nbPaireVendu >= 0);  
ALTER TABLE Statistiques ADD CONSTRAINT positive_BeneficeTotal CHECK (BeneficeTotal >= 0);  
ALTER TABLE Statistiques ADD CONSTRAINT positive_CATotal CHECK (CATotal >= 0);
```

Inventaires = (id VARCHAR(50), nomArticle VARCHAR(50), marque VARCHAR(50));

Ventes = (heureVente DATETIME, prixVente CURRENCY, lieuVente VARCHAR(50), Acheteur VARCHAR(50), suivisColisVente VARCHAR(50));

Statistiques = (idStat INT, nbPaireVendu INT, BeneficeTotal CURRENCY, CATotal CURRENCY);

Membres = (email VARCHAR(50), nomMembre VARCHAR(50), prenomMembre VARCHAR(50), telephone INT, estAdmin LOGICAL, mdphache VARCHAR(50), sel VARCHAR(50), #idStat*);

possède = (#email, #id, quantité INT, taille VARCHAR(50), Statut VARCHAR(50), prixAchat CURRENCY, prix_de_vente_souhaite CURRENCY, dateAchat DATE, LieuAchat VARCHAR(50), Vendeur VARCHAR(50), suivisColisAchat VARCHAR(50));

vends = (#email, taille VARCHAR(50), #id, #heureVente);

Réalisation

Développement des Modèles :

Les modèles ont été développés pour chaque entité de la base de données. Ces modèles sont responsables de la gestion des données, y compris les opérations CRUD (Create, Read, Update, Delete). Voici un exemple de code simplifié pour le modèle `Produit` :

Figure 1 : Code la fonction de construction d'un objet Produit

```
2 usages  ▲ ternera
public function __construct(int $id, string $nomArticle, string $marque, string $taille, float $prixAchat, float $prixVenteSouhaite, string $dateAchat, string $statut, string $lieuAchat, string $vendeur, string $suivisColis)
{
    $this->id = $id;
    $this->nomArticle = $nomArticle;
    $this->marque = $marque;
    $this->taille = $taille;
    $this->prixAchat = $prixAchat;
    $this->prixVenteSouhaite = $prixVenteSouhaite;
    $this->dateAchat = $dateAchat;
    $this->statut = $statut;
    $this->lieuAchat = $lieuAchat;
    $this->vendeur = $vendeur;
    $this->suivisColis = $suivisColis;
}
```

Développement des Vues :

Les vues sont développées en HTML, CSS, et JavaScript. L'une des fonctionnalités clés de l'application est le tableau de bord (dashboard), où les utilisateurs peuvent visualiser des statistiques sur leurs ventes. Pour rendre cette partie plus interactive et informative, **Chart.js** a été utilisé.

Chart.js est une bibliothèque JavaScript simple et flexible qui permet de créer différents types de graphiques, tels que des barres, des lignes, des radars, et des pie charts. L'intégration de Chart.js dans le projet a été faite pour fournir une visualisation graphique des données de vente. Le choix de Chart.js a été motivé par sa simplicité d'utilisation, sa flexibilité, et la qualité des graphiques générés.

Développement des Contrôleurs :

Les contrôleurs orchestrent l'interaction entre les vues et les modèles. Ils sont responsables de traiter les requêtes HTTP, de récupérer les données des modèles, et de les transmettre aux vues pour affichage. Un exemple de contrôleur pour gérer la connexion utilisateur ressemble à ceci :

Figure 2 : Fonction connexion du ControleurUtilisateur

```
no usages  terra
public static function connecter()
{
    session_start();
    if (isset($_REQUEST['login']) && isset($_REQUEST['mdp'])) {
        $login = $_REQUEST['login'];
        $mdp = $_REQUEST['mdp'];
        $utilisateur = (new UtilisateurRepository())->recupererParClePrimaire($login);

        if (!is_null($utilisateur) && MotDePasse::verifier($mdp, $utilisateur->getMdpHache())) {
            $_SESSION['utilisateur'] = $login;
            ConnexionUtilisateur::connecter($login);

            MessageFlash::ajouter( type: 'succes', message: "Utilisateur connecté");
            $url = "?action=afficherListe&controleur=chaussure";
            ControleurUtilisateur::redirectionVersURL($url);
        } else {
            MessageFlash::ajouter( type: 'warning', message: "l'utilisateur " . $login . " n'existe pas ou le mot de passe n'est pas correct");
            $url = "?action=afficherListe&controleur=chaussure";
            ControleurUtilisateur::redirectionVersURL($url);
        }
    } else {
        MessageFlash::ajouter( type: 'warning', message: "login ou mdp non renseigné");
        $url = "?action=afficherListe&controleur=chaussure";
        ControleurUtilisateur::redirectionVersURL($url);
    }
}
```

Le contrôleur ci-dessous prend en charge l'enregistrement d'une vente :

Figure 3 : Différentes fonction de ControleurVente

```
<?php

namespace App\Controller;

use App\Modele\Repository\VenteRepository;
use App\Modele\DataObject\Vente;

no usages new *
class ControleurVente extends ControleurGenerique
{
    no usages new *
    public static function ajouterVente($id, $heureVente, $prixVente, $lieuVente, $acheteur, $suivisColisVente): void
    {
        $nouvelleVente = new Vente($id, $heureVente, $prixVente, $lieuVente, $acheteur, $suivisColisVente);

        (new VenteRepository())->sauvegarder($nouvelleVente);

        self::redirectionVersURL( url: "?action=afficherListe&controleur=vente");
    }

    no usages new *
    public static function supprimerVente($id): void
    {
        (new VenteRepository())->supprimer($id);

        self::redirectionVersURL( url: "?action=afficherListe&controleur=vente");
    }

    no usages new *
    public static function modifierVente($id, $heureVente, $prixVente, $lieuVente, $acheteur, $suivisColisVente): void
    {
        $venteModifiee = new Vente($id, $heureVente, $prixVente, $lieuVente, $acheteur, $suivisColisVente);

        (new VenteRepository())->modifier($venteModifiee);

        self::redirectionVersURL( url: "?action=afficherListe&controleur=vente");
    }
}
```

Création de la base de données :

La création de la base de données pour l'application de gestion des stocks et des ventes est une étape fondamentale qui garantit le bon fonctionnement et l'intégrité des données manipulées par l'application. Cette section détaille la structure de la base de données, les tables créées, ainsi que les relations entre ces tables, en se basant sur le modèle conceptuel fourni.

Tables et Relations

La base de données est composée de plusieurs tables interconnectées, représentant les entités principales du système. Chaque table est conçue pour stocker des informations spécifiques liées aux utilisateurs, aux articles, aux ventes, et aux statistiques.

1. Table `Membres`

La table `Membres` stocke les informations relatives aux utilisateurs de l'application. L'email est utilisé comme clé primaire pour identifier de manière unique chaque utilisateur.

Figure 4 : Table Membres

```
CREATE TABLE Membres (  
    email VARCHAR(50) PRIMARY KEY,  
    nomMembre VARCHAR(50),  
    prenomMembre VARCHAR(50),  
    telephone INT,  
    estAdmin BOOLEAN,  
    mdpHache VARCHAR(50),  
    sel VARCHAR(50)  
);
```

- email : Identifiant unique de chaque utilisateur.
- nomMembre et prenomMembre : Nom et prénom de l'utilisateur.
- telephone : Numéro de téléphone de l'utilisateur.
- estAdmin : Booléen indiquant si l'utilisateur est un administrateur.
- mdpHache : Mot de passe haché de l'utilisateur pour sécuriser l'authentification.
- sel : Chaîne aléatoire utilisée pour le salage du mot de passe.

2. Table `Produits`

La table `Produits` contient les informations sur les articles stockés par les utilisateurs. Chaque article est identifié de manière unique par son `id`.

Figure 5 : Table Produits

```
CREATE TABLE Articles (  
    id VARCHAR(50) PRIMARY KEY,  
    nomArticle VARCHAR(50),  
    marque VARCHAR(50)  
);
```

- id : Identifiant unique de l'article.
- nomArticle : Nom de l'article.
- marque : Marque de l'article.

3. Table `Ventes`

La table `Ventes` enregistre les transactions de vente. Chaque vente est identifiée par l'heure de vente (`heureVente`).

Figure 6 : Table Ventes

```
CREATE TABLE Ventes (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    heureVente DATETIME,  
    prixVente DECIMAL(10, 2),  
    lieuVente VARCHAR(50),  
    Acheteur VARCHAR(50),  
    suivisColisVente VARCHAR(50)  
);
```

- heureVente : Date et heure de la vente, servant d'identifiant unique.
- prixVente : Prix de vente de l'article.
- lieuVente : Lieu où la vente a eu lieu.
- Acheteur : Nom de l'acheteur.
- suivisColisVente : Numéro de suivi du colis lors de l'envoi à l'acheteur.

4. Table `Statistiques`

La table `Statistiques` stocke les données agrégées concernant les ventes, telles que le nombre d'articles vendus, le bénéfice total et le chiffre d'affaires.

Figure 7 : Table Statistiques

```
CREATE TABLE Statistiques (  
    email PRIMARY KEY,  
    nbPaireVendu INT,  
    BeneficeTotal DECIMAL(10, 2),  
    CATotal DECIMAL(10, 2),  
    email VARCHAR(50),  
    FOREIGN KEY (email) REFERENCES Membres(email)  
);
```

- nbPaireVendu : Nombre total de paires ou articles vendues.
- BeneficeTotal : Bénéfice total réalisé.
- CATotal : Chiffre d'affaires total.
- email : Clé étrangère liant les statistiques à un utilisateur spécifique et aussi clé primaire car un utilisateur possède qu'une statistique propre à lui-même.

5. Table `possede`

La table `possede` représente la relation entre les utilisateurs et les articles qu'ils possèdent.

Figure 8 : Table Possede

```
CREATE TABLE possede (  
    email VARCHAR(50),  
    id VARCHAR(50),  
    quantite INT,  
    taille ENUM('XS', 'S', 'M', 'L', 'XL', 'XXL', 'XXXL', '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36', '36.5', '37', '37.5',  
        '38', '38.5', '39', '39.5', '40', '40.5', '41', '41.5', '42', '42.5', '43', '44', '44.5', '45', '45.5', '46', '46.5', '47', '48', '48.5', '49', '50', 'autre', 'TAILLE UNIQUE'),  
    statut ENUM('pas reçu', 'reçu', 'en dépôt vente', 'envoyé au client', 'vendu'),  
    prixAchat DECIMAL(10, 2),  
    prix_de_vente_souhaite DECIMAL(10, 2),  
    dateAchat DATE,  
    lieuAchat VARCHAR(50),  
    vendeur VARCHAR(50),  
    suivisColis VARCHAR(50),  
    PRIMARY KEY (email, id, taille, statut),  
    FOREIGN KEY (email) REFERENCES Membres(email),  
    FOREIGN KEY (id) REFERENCES articles(id)  
);
```

- email : Clé étrangère vers la table `Membres`, identifiant le propriétaire de l'article.
- id : Clé étrangère vers la table `Inventaires`, identifiant l'article possédé.
- quantite : Quantité de cet article dans une certaine taille possédé par l'utilisateur.
- taille : Taille de l'article (chaussures ou vêtements).

- prixAchat : Prix d'achat de l'article.
- prix_de_vente_souhaite : Prix de vente souhaité par l'utilisateur.
- dateAchat : Date d'achat de l'article.
- Statut : Statut de l'article (e.g., en stock, vendu, etc.).
- LieuAchat : Lieu où l'article a été acheté.
- Vendeur : Nom du vendeur qui a vendu l'article.
- suivisColisAchat : Numéro de suivi du colis lors de l'achat.

6. Table `vends`

La table `vends` relie les ventes effectuées aux articles vendus, en tenant compte de la quantité vendue.

Figure 9 : Table Vends

```
CREATE TABLE vends (  
  email VARCHAR(50),  
  idArticle VARCHAR(50),  
  taille ENUM('XS', 'S', 'M', 'L', 'XL', 'XXL', 'XXXL', '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36', '36.5', '37', '37.5',  
    '38', '38.5', '39', '39.5', '40', '40.5', '41', '41.5', '42', '42.5', '43', '44', '44.5', '45', '45.5', '46', '46.5', '47', '48', '48.5', '49', '50', 'autre', 'TAILLE UNIQUE'),  
  idVente INT,  
  PRIMARY KEY (email, idArticle, taille, idVente),  
  FOREIGN KEY (email) REFERENCES Membres(email),  
  FOREIGN KEY (idArticle) REFERENCES Articles(id),  
  FOREIGN KEY (idVente) REFERENCES Ventes(id)  
);
```

- email : Clé étrangère vers la table `Membres`, identifiant le vendeur.
- idArticle : Clé étrangère vers la table `Inventaires`, identifiant l'article vendu.
- idVente : Clé étrangère vers la table `Ventes`, identifiant la vente.

Validation de la Base de Données :

Après la création des tables, plusieurs tests ont été effectués pour valider la structure de la base de données :

- Opérations CRUD : Chaque table a été testée pour les opérations CRUD (Create, Read, Update, Delete) afin de s'assurer que les données peuvent être manipulées sans erreur.
- Performances : Les requêtes ont été optimisées à l'aide d'index sur les colonnes fréquemment interrogées, améliorant ainsi les performances de l'application sous des charges de travail

réalistes.

```
CREATE INDEX idx_membre_email ON Membres(email);  
CREATE INDEX idx_inventaires_nomArticle ON Inventaires(nomArticle);  
CREATE INDEX idx_ventes_heureVente ON Ventes(heureVente);
```

Conclusion :

La création de la base de données a été une étape clé dans le développement de l'application, assurant que les données sont stockées de manière efficace et sécurisée. Le modèle relationnel, basé sur le diagramme E/A, garantit l'intégrité des données et permet une manipulation fluide des informations à travers l'application. Les tests de validation effectués confirment que la structure de la base de données répond aux exigences fonctionnelles du projet.

Gestion des erreurs et des exceptions

Un effort particulier a été fait pour gérer les erreurs et les exceptions de manière appropriée. Cela inclut la validation des données en entrée, la gestion des erreurs SQL, et la fourniture de messages d'erreur clairs pour l'utilisateur.

Validation :

1. Tests unitaires :

Chaque modèle et contrôleur a été soumis à des tests unitaires pour vérifier que les opérations fonctionnaient comme prévu. Par exemple, le modèle `Article` a été testé pour s'assurer que les articles pouvaient être créés, modifiés, et supprimés correctement.

2. Tests utilisateurs :

L'application a été testée par des personnes pour recueillir des retours sur l'interface utilisateur et les fonctionnalités. Les tests utilisateurs ont permis d'identifier et de corriger des bugs.

Focus sur des parties techniques

Gestion des graphiques avec Chart.js

Chart.js est une bibliothèque JavaScript open-source qui permet de créer facilement des graphiques interactifs et réactifs. Elle supporte plusieurs types de graphiques, tels que les graphiques en barre, les graphiques linéaires, les graphiques circulaires... Dans le cadre du projet, Chart.js a été utilisé pour afficher des statistiques sur les ventes dans le tableau de bord de l'application, offrant ainsi une visualisation claire et intuitive des données.

Problématique de la gestion des données dynamiques

Au début du projet, il était prévu de charger les données des ventes directement depuis le serveur via des requêtes AJAX. Cette approche permettrait de mettre à jour les graphiques en temps réel, sans nécessiter de rechargement complet de la page. Cependant, plusieurs défis sont rapidement apparus :

L'intégration d'AJAX pour le chargement des données a imposé une gestion asynchrone des requêtes, ce qui a considérablement complexifié la logique du code. En effet, la nécessité de gérer des opérations asynchrones a rendu l'interaction avec d'autres éléments dynamiques de la page plus difficile, nécessitant une orchestration minutieuse pour assurer la synchronisation entre les différents composants de l'interface. Cette complexité a été particulièrement marquée dans les cas où plusieurs éléments de la page devaient réagir simultanément aux changements de données, obligeant à concevoir des mécanismes de coordination sophistiqués pour maintenir l'intégrité et la cohérence de l'affichage.

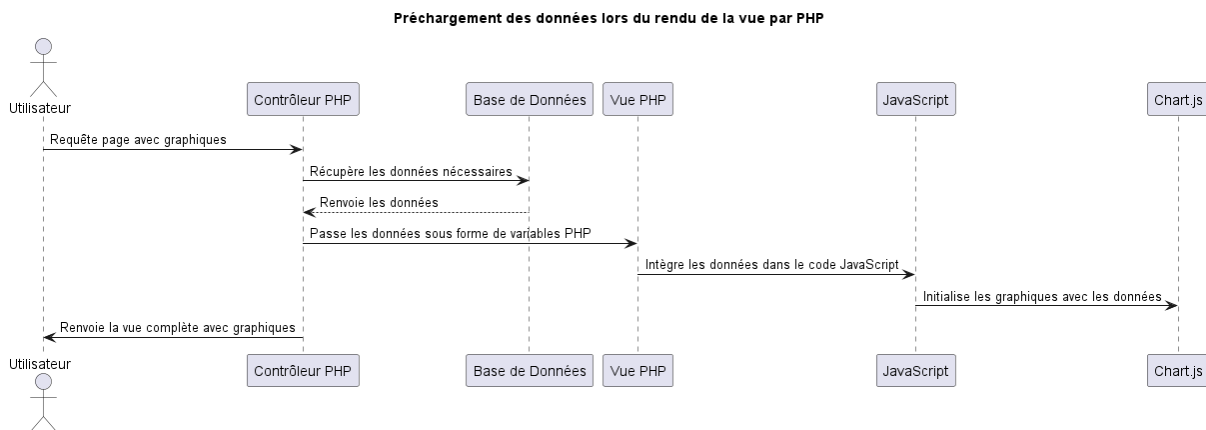
Bien que l'utilisation d'AJAX ait permis de charger les données sans interrompre l'interaction utilisateur, elle a introduit des latences perceptibles, notamment lors de la mise à jour des graphiques. Ces retards, bien que tolérables pour des volumes de données modestes, peuvent devenir de plus en plus prononcés à mesure que la quantité de données augmente. Cette dégradation progressive des performances a un impact direct sur l'expérience utilisateur, qui pouvait percevoir un ralentissement ou une certaine lourdeur dans les interactions avec les éléments dynamiques de la page.

La gestion des erreurs associées aux requêtes AJAX a ajouté une couche supplémentaire de complexité. Les erreurs de serveur, ou même des réponses inattendues nécessitaient une gestion d'erreurs robuste pour prévenir les dysfonctionnements et informer l'utilisateur de manière appropriée. Cela a impliqué non seulement la mise en place de mécanismes de récupération pour maintenir une expérience utilisateur fluide malgré les interruptions, mais aussi la gestion des cas d'échec de manière à éviter tout blocage ou comportement incohérent dans l'application. Ces considérations ont donc exigé une attention particulière pour garantir une robustesse du système face aux divers aléas liés aux communications réseau.

Solution : Préchargement des données via PHP

Pour surmonter ces défis, nous avons opté pour une approche différente : le préchargement des données lors du rendu de la vue par PHP. Cette méthode consiste à récupérer les données nécessaires depuis la base de données directement dans le contrôleur, puis à les passer à la vue sous forme de variables PHP. Les données sont ensuite intégrées dans le code JavaScript qui initialise les graphiques avec Chart.js. Cette approche présente plusieurs avantages :

Diagramme 3 : Préchargement des données



Ce code a permis une simplification significative en supprimant la nécessité d'utiliser AJAX pour le chargement des données, ce qui a conduit à un allègement considérable du code JavaScript. Au lieu de gérer le traitement des données directement dans le front-end, toute la logique complexe a été transférée côté serveur. Cette centralisation a non seulement rendu le code plus propre et plus lisible, mais elle a également facilité sa maintenance. Les développeurs peuvent désormais se concentrer sur l'interface utilisateur sans avoir à se soucier des subtilités du traitement des données, ce qui réduit les risques d'erreurs et améliore l'efficacité globale du développement.

Cette nouvelle architecture améliore considérablement les performances de l'application. Désormais, les données sont chargées une seule fois, au moment du rendu initial de la page, éliminant ainsi le besoin de requêtes asynchrones répétées. Cela signifie que les graphiques et autres éléments interactifs s'affichent immédiatement, sans la moindre latence, offrant une expérience utilisateur beaucoup plus fluide et réactive. Cette rapidité d'exécution est particulièrement appréciable dans les applications où la réactivité est essentielle pour l'utilisateur final.

En centralisant le traitement des données sur le serveur, la robustesse de l'application a été renforcée. Il est désormais plus simple de gérer les erreurs de traitement et de s'assurer que les données envoyées au front-end sont toujours valides, bien formatées et prêtes à être affichées sans problème. Cette approche minimise les risques de dysfonctionnement au niveau de l'interface utilisateur et améliore la fiabilité globale de l'application.

Mise en œuvre technique :

Voici comment cette solution a été mise en œuvre :

1. Récupération des données côté serveur :

- Le contrôleur responsable de la page du tableau de bord interagit avec les modèles pour récupérer les données de vente depuis la base de données. Ces données sont ensuite formatées en un tableau PHP que la vue peut utiliser.

Figure 10 : ControleurDashboard

```
$utilisateur = ConnexionUtilisateur::getLoginUtilisateurConnecte();

$statRepo = new StatistiquesRepository();
$statistiques = $statRepo->getStatistiquesByUserId($utilisateur);
$vendRepo = new VendRepository();
$ventesParMois = $vendRepo->getVentesParMois($utilisateur);

if ($statistiques) {
    $nbVendu = $statistiques->getNbPaireVendu();
    $beneficeTotal = $statistiques->getBeneficeTotal();
    $caTotal = $statistiques->getCATotal();
} else {
    $nbPaireVendu = 0;
    $beneficeTotal = 0.0;
    $caTotal = 0.0;
}

$this->afficherVue( cheminVue: 'dashboard/dashboardBDD.php', [
    'pagetitle' => 'Dashboard',
    'nbVendu' => $nbVendu,
    'beneficeTotal' => $beneficeTotal,
    'caTotal' => $caTotal,
    'ventesParMois' => $ventesParMois
]);
}
```

2. Passage des données à la vue :

- Dans la vue PHP, les données sont injectées directement dans le code JavaScript de Chart.js. Cela permet à Chart.js de disposer immédiatement des données nécessaires à l'affichage des graphiques, sans nécessiter de requêtes supplémentaires.

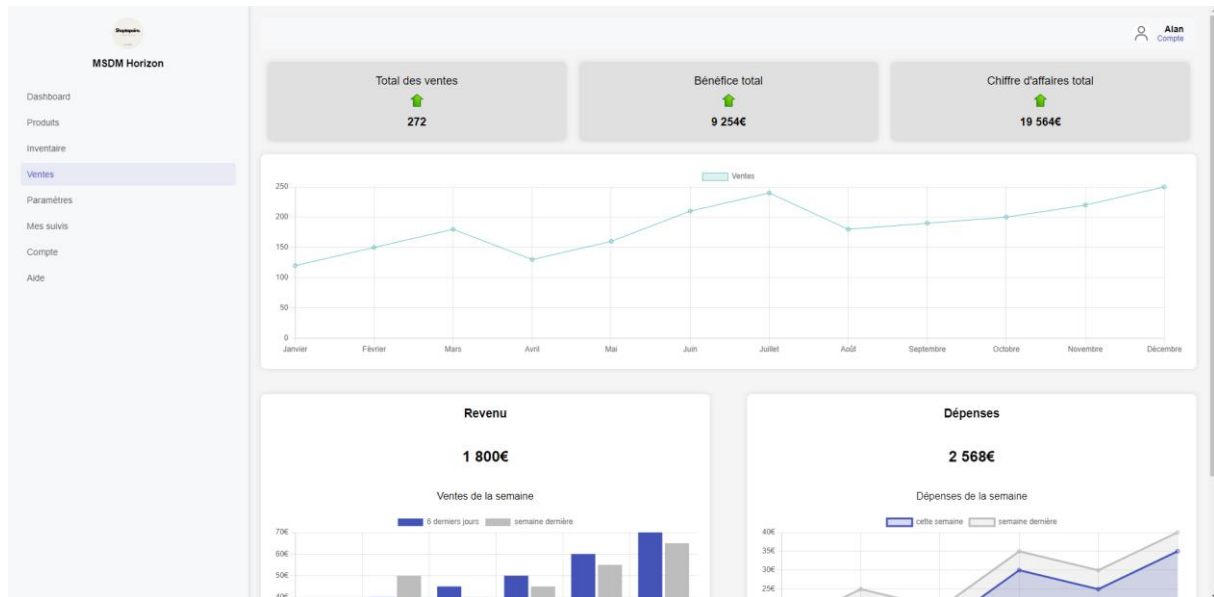
Figure 11 : Création d'un graphique

```
var myChart = new Chart(ctx, {  
  type: 'line',  
  data: {  
    labels: labels,  
    datasets: [{  
      label: 'Ventes',  
      data: data,  
      backgroundColor: 'rgba(75, 192, 192, 0.2)',  
      borderColor: 'rgba(75, 192, 192, 1)',  
      borderWidth: 1  
    }]  
  },  
  options: {  
    responsive: true,  
    maintainAspectRatio: false,  
    scales: {  
      y: {  
        beginAtZero: true  
      }  
    }  
  }  
});
```

3. Résultat :

Lorsqu'un utilisateur accède au tableau de bord, les données de vente sont déjà disponibles dans le code JavaScript. Chart.js les utilise pour générer immédiatement le graphique, offrant ainsi une expérience utilisateur fluide et réactive.

Figure 12 : Page web Dashboard



Avantages et inconvénients de la solution

Avantages :

En éliminant le besoin d'utiliser AJAX pour charger les données, la complexité du code JavaScript sur le front-end est considérablement réduite. Cela se traduit par un code plus lisible et plus facile à maintenir, ce qui facilite le travail des développeurs et réduit les risques d'erreurs liées à une logique complexe. Cette approche améliore aussi les performances globales de l'application, car les graphiques sont désormais générés directement au moment du chargement de la page, ce qui évite toute latence associée à des requêtes asynchrones supplémentaires. Par ailleurs, la gestion des erreurs devient plus robuste et centralisée, puisque les potentielles erreurs liées au chargement des données sont prises en charge côté serveur. Cela simplifie non seulement la logique front-end, mais permet aussi d'assurer une gestion cohérente des exceptions, garantissant ainsi une meilleure stabilité et une maintenance facilitée de l'application.

Inconvénients :

L'utilisation de données statiques au chargement signifie que les graphiques affichés ne se mettent pas à jour en temps réel sans un rechargement complet de la page. Cette limitation peut poser des problèmes pour les applications qui nécessitent des mises à jour fréquentes et en temps réel des données, limitant ainsi leur réactivité et leur capacité à refléter les changements instantanés mais ici ce n'est pas le cas car notre application n'avait pas ce besoin. De plus, cette approche crée une forte dépendance vis-à-vis de PHP pour le rendu des données, car le serveur doit fournir les informations préformatées lors du chargement initial. Cette dépendance peut réduire la flexibilité de l'application lorsqu'il s'agit de charger dynamiquement de nouvelles

données après que la page a été initialement rendue, rendant plus difficile l'adaptation à des besoins évolutifs ou à des données qui changent fréquemment.

L'intégration de Chart.js avec un préchargement des données via PHP a été une solution efficace pour ce projet, permettant de surmonter les défis liés à la gestion des données dynamiques. Cette méthode a simplifié le code, amélioré les performances et fourni une expérience utilisateur fluide. Bien que cette approche présente quelques limitations, elle s'est avérée bien adaptée aux besoins spécifiques de l'application développée pour MSDM Horizon. Les enseignements tirés de cette expérience pourront être appliqués à des projets futurs, en tenant compte des besoins en temps réel.

Configuration de l'application : le fichier `config.php`

Le fichier `config.php` joue un rôle central dans la configuration et le bon fonctionnement de l'application. Il centralise plusieurs paramètres critiques, tels que les informations de connexion à la base de données, les configurations globales de l'application, et les secrets de sécurité. Cette approche permet de séparer les configurations du code métier, rendant l'application plus modulaire, sécurisée, et facile à maintenir. J'ai fait cela pour faciliter au non informaticiens le déploiement de l'application. De plus le fichier est seulement accessible par le propriétaire de l'application ce qui permet une meilleure sécurité de l'application.

Contenu et structure du fichier `config.php` :

Le fichier `config.php` est organisé en plusieurs sections, chacune définissant des constantes pour différents aspects de la configuration de l'application.

1. Informations de connexion à la base de données

La première section du fichier `config.php` définit les constantes liées aux informations de connexion à la base de données pour deux environnements distincts : local et l'environnement non local. Cela permet de basculer facilement entre différents environnements de développement sans modifier le code de l'application.

Figure 13 : Informations de connexion

```
<?php

namespace App\Configuration;

// Informations de connexion à la base de données
define('DB_HOST_LOCAL', 'localhost');
define('DB_HOST_NONLOCAL ', 'webinfo');
define('DB_NAME_LOCAL', 'myapp');
define('DB_NAME_NONLOCAL ', 'terriera');
define('DB_PORT_LOCAL', '3306');
define('DB_PORT_NONLOCAL ', '3316');
define('DB_USER_LOCAL', 'root');
define('DB_USER_NONLOCAL ', 'terriera');
define('DB_PASS_LOCAL', '');
define('DB_PASS_NONLOCAL ', '');
```

- DB_HOST_LOCAL et DB_HOST_NONLOCAL : Définissent l'hôte de la base de données pour les environnements local et non local respectivement.
- DB_NAME_LOCAL et DB_NAME_NONLOCAL : Noms des bases de données utilisées dans chaque environnement.
- DB_PORT_LOCAL et DB_PORT_NONLOCAL : Ports sur lesquels la base de données MySQL écoute dans chaque environnement.
- DB_USER_LOCAL et DB_USER_NONLOCAL : Noms d'utilisateur pour la connexion à la base de données.
- DB_PASS_LOCAL et DB_PASS_NONLOCAL : Mots de passe associés aux utilisateurs des bases de données.

Avantage : Cette séparation permet de développer localement avec une configuration simplifiée (souvent sans mot de passe pour la base de données) tout en étant capable de déployer la même application dans un environnement plus sécurisé.

2. Autres configurations globales

Cette section définit des constantes qui sont utilisées à travers l'application pour des fonctionnalités générales.

Figure 14 : Configuration globale

```
// Autres configurations globales
define('APPROOT', dirname(dirname( path: __FILE__)));
define('URLROOT', 'http://localhost/myapp');
define('SITENAME', 'My App');
define('dureeExpiration', '3600');
```

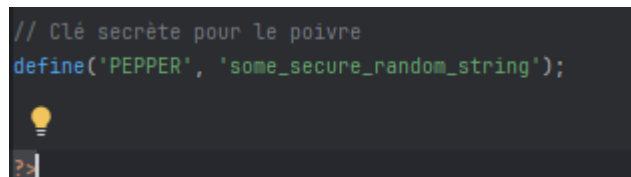
- APPROOT : Définit le chemin absolu vers le répertoire racine de l'application. Cette constante est souvent utilisée pour inclure des fichiers ou pour définir des chemins relatifs.
- URLROOT : Détermine l'URL de base de l'application. Elle est utilisée pour générer des liens internes de manière dynamique.
- SITENAME : Nom du site ou de l'application, utilisé pour afficher le titre dans la barre de titre du navigateur ou dans d'autres parties de l'interface utilisateur.
- dureeExpiration : Définit la durée d'expiration (en secondes) pour certaines fonctionnalités, comme les sessions.

Avantage : En centralisant ces configurations, il devient facile de modifier les paramètres globaux de l'application sans toucher au code source, facilitant ainsi la gestion de différentes versions de l'application (développement, production, etc.).

3. Clé secrète pour le poivre

La clé `PEPPER` est une chaîne de caractères secrète utilisée pour sécuriser davantage les mots de passe des utilisateurs. Elle est combinée avec les mots de passe avant de les hacher, ce qui ajoute une couche de sécurité supplémentaire en rendant plus difficile toute attaque basée sur des rainbow tables.

Figure 15 : PEPPER

A screenshot of a code editor with a dark background. It shows two lines of code: a comment in French and a PHP define statement. The comment reads '// Clé secrète pour le poivre' and the code line is 'define('PEPPER', 'some_secure_random_string');'. There is a small lightbulb icon on the left side of the code editor, indicating a tip or important note.

```
// Clé secrète pour le poivre
define('PEPPER', 'some_secure_random_string');
```

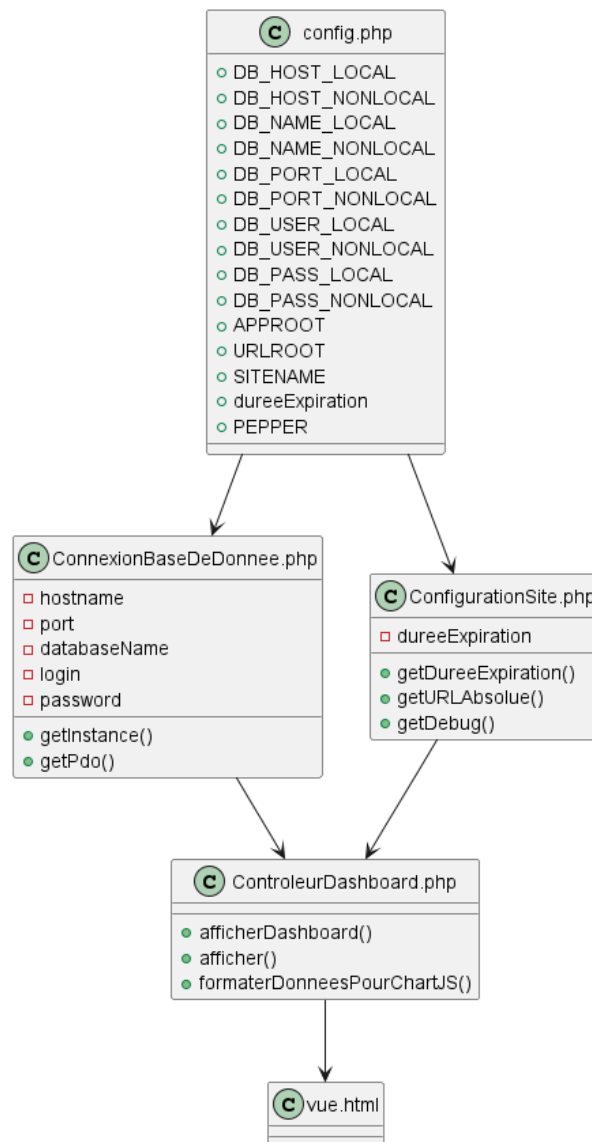
- PEPPER : Cette constante contient une chaîne aléatoire utilisée comme "poivre" dans le processus de hachage des mots de passe.

Avantage : L'utilisation du poivre (en plus du sel, qui est stocké avec le mot de passe haché) rend les attaques par force brute ou par rainbow tables beaucoup plus difficiles. Même si un attaquant parvient à obtenir la base de données et le sel, il lui manquerait toujours le poivre pour recréer les mots de passe.

Schéma d'interaction

Voici un schéma illustrant comment les configurations définies dans `config.php` interagissent avec les différents composants de l'application :

Diagramme 4 : Interaction du fichier `config.php` avec les composants de l'application.



Conclusion de la configuration :

Le fichier `config.php` est une pièce maîtresse de l'application, centralisant les paramètres critiques dans un endroit unique et facile à gérer. En utilisant des constantes pour stocker ces paramètres, l'application devient plus flexible, plus sécurisée, et plus facile à déployer dans différents environnements. Cette approche modulaire permet également de minimiser les erreurs de configuration et de simplifier le processus de développement et de maintenance.

Création d'un Trigger

Développement du Trigger :

Le trigger `update_statistiques_after_vente` joue un rôle crucial dans l'application de gestion des ventes en automatisant la mise à jour des statistiques associées à chaque utilisateur dès qu'une nouvelle vente est enregistrée. Ce trigger est conçu pour s'exécuter après l'insertion d'une nouvelle transaction dans la table `vends`, garantissant que les données dans la table `Statistiques` sont toujours à jour, reflétant le nombre de paires vendues, le bénéfice total, et le chiffre d'affaires total (CATotal).

Structure et Fonctionnement du Trigger

Ce trigger est déclenché après une opération d'insertion (AFTER INSERT) dans la table `vends`. Cette exécution post-insertion permet d'utiliser les nouvelles données insérées pour mettre à jour immédiatement les statistiques de l'utilisateur concerné.

Déclarations de Variables

Le trigger commence par déclarer plusieurs variables locales, destinées à stocker temporairement les informations nécessaires pour effectuer les mises à jour :

Figure 16 : Déclarations de Variables

```
CREATE TRIGGER update_statistiques_after_vente
AFTER INSERT ON vends
FOR EACH ROW
BEGIN
  DECLARE prixAchat DECIMAL(10, 2);
  DECLARE prixVente DECIMAL(10, 2);
  DECLARE benefice DECIMAL(10, 2);
```

- **prixAchat** : Stocke le prix d'achat de l'article vendu, récupéré à partir de la table `possede`.
- **prixVente** : Contient le prix de vente de l'article, extrait de la table `Ventes`.
- **benefice** : Représente le bénéfice généré par la vente, calculé comme la différence entre le prix de vente et le prix d'achat.

Récupération des Données Nécessaires

Ensuite, le trigger procède à la récupération des informations nécessaires pour le calcul des statistiques. Deux requêtes SQL sont exécutées :

1. Récupération du prix d'achat :

Figure 17 : Récupération des Données

```
-- Récupérer le prix d'achat de l'article vendu avec la bonne taille et statut  
SELECT prixAchat INTO prixAchat  
FROM possede  
WHERE email = NEW.email  
AND id = NEW.idArticle  
AND taille = NEW.taille  
AND statut = 'vendu'  
LIMIT 1;
```

Cette requête interroge la table possede pour obtenir le prix d'achat de l'article spécifique vendu. Elle identifie l'article en fonction de l'email de l'utilisateur, de l'ID de l'article, de la taille, et du statut de l'article.

2. Récupération du prix de vente :

Figure 18 : Récupération des Données

```
-- Récupérer le prix de vente de l'article vendu  
SELECT prixVente INTO prixVente  
FROM Ventes  
WHERE id = NEW.idVente  
LIMIT 1;
```

Cette requête extrait le prix de vente de l'article à partir de la table Ventes, en se basant sur l'ID de la vente nouvellement insérée.

Calcul du Bénéfice et Mise à Jour des Statistiques

Après avoir récupéré les données nécessaires, le trigger calcule le bénéfice réalisé sur la vente et met à jour les statistiques associées à l'utilisateur :

Figure 19 : Calcul et mise à jour des statistiques

```
-- Calculer le bénéfice pour 1 article vendu
SET benefice = prixVente - prixAchat;

-- Mettre à jour la table Statistiques
INSERT INTO Statistiques (email, nbPaireVendu, BeneficeTotal, CATotal)
VALUES (NEW.email, 1, benefice, prixVente)
ON DUPLICATE KEY UPDATE
    nbPaireVendu = nbPaireVendu + 1,
    BeneficeTotal = BeneficeTotal + benefice,
    CATotal = CATotal + prixVente;
```

Le bénéfice est simplement la différence entre le prix de vente et le prix d'achat de l'article vendu. Cette commande INSERT ... ON DUPLICATE KEY UPDATE est particulièrement utile car elle permet soit d'insérer de nouvelles données si l'utilisateur n'a pas encore d'entrée dans la table Statistiques, soit de mettre à jour les données existantes en incrémentant les valeurs des ventes, du bénéfice et du chiffre d'affaires.

Mise à Jour de la Table possede

En parallèle, le trigger met également à jour la table possede pour décrémenter la quantité de l'article vendu :

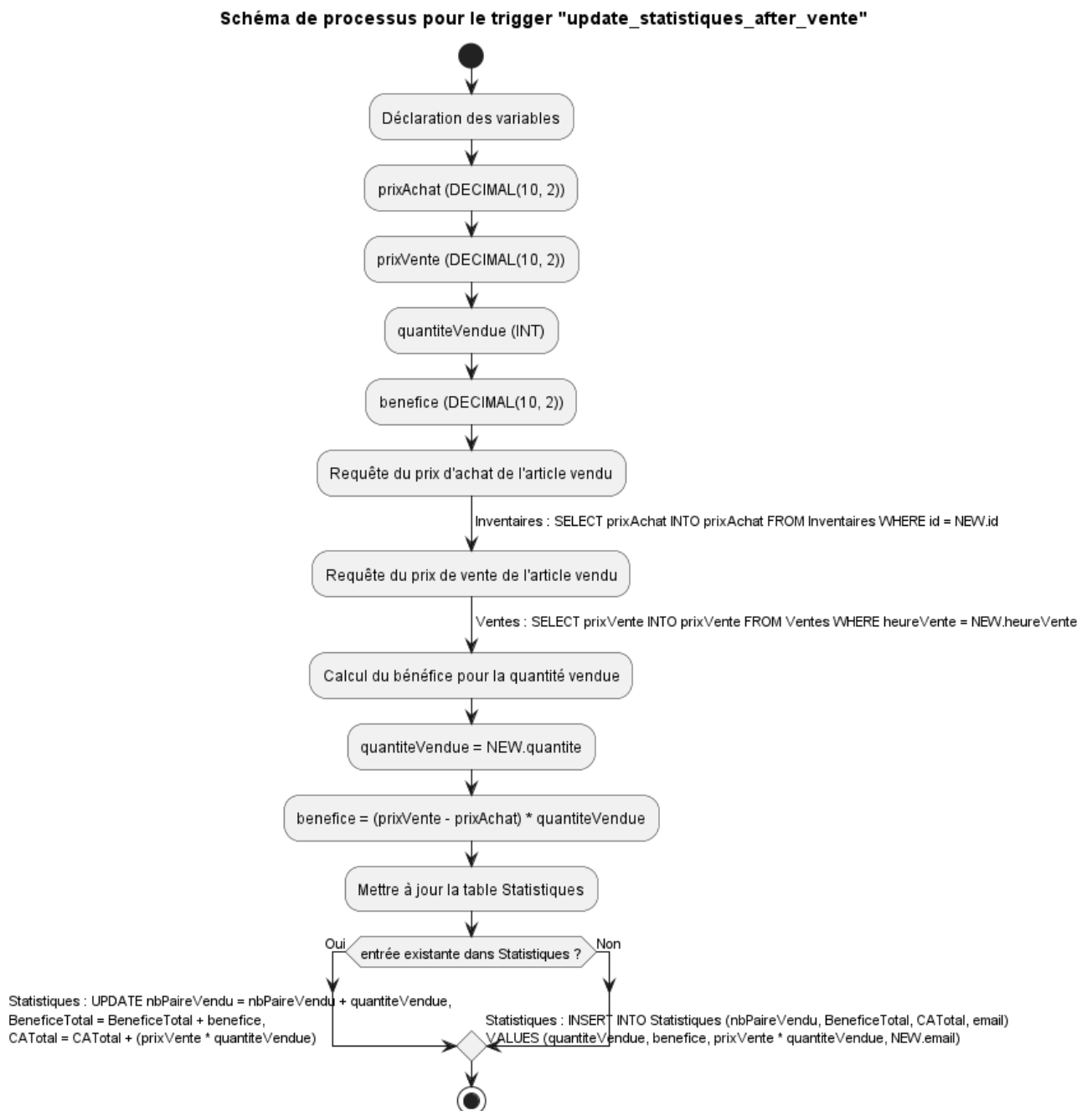
Figure 20 : Mise à jour des données

```
-- Décrémenter la quantité de l'article dans la table possede
UPDATE possede
SET quantite = quantite - 1
WHERE email = NEW.email
    AND id = NEW.idArticle
    AND taille = NEW.taille
    AND statut = 'vendu';
```

Schéma du Processus du Trigger :

Pour mieux comprendre le flux de ce processus, le schéma ci-dessous illustre les étapes suivies par le trigger `update_statistiques_after_vente` :

Diagramme 5 : Processus du trigger `update_statistiques_after_vente`



Avantages et Limitations du Trigger :

Avantages

L'automatisation des mises à jour est un élément crucial pour assurer que les statistiques restent constamment à jour sans nécessiter d'intervention humaine. En mettant en place un trigger qui se déclenche automatiquement, les données sont actualisées en temps réel, ce qui non seulement garantit une précision optimale, mais réduit également le risque d'erreurs humaines qui pourraient survenir lors de mises à jour manuelles. Cette approche permet de maintenir l'intégrité des données et de répondre rapidement aux changements, offrant ainsi une fiabilité accrue.

L'utilisation de la clause ``ON DUPLICATE KEY UPDATE`` dans les requêtes SQL ajoute une couche d'efficacité supplémentaire. Cette clause permet de gérer les cas où une ligne existe déjà dans la base de données et doit être mise à jour plutôt qu'insérée à nouveau. En évitant les doublons et en mettant directement à jour les enregistrements existants, cette méthode optimise les performances, en particulier lors de nombreuses transactions. Cela est particulièrement avantageux dans des environnements où les bases de données sont fortement sollicitées, car cela réduit le temps de traitement et les ressources nécessaires, tout en garantissant que les informations sont toujours cohérentes et à jour.

Limitations

Les triggers, en raison de leur nature automatisée, s'exécutent en arrière-plan chaque fois qu'une condition prédéfinie est remplie, par exemple lors de l'insertion, la mise à jour ou la suppression de données dans une table. Cependant, cette automatisation, bien que puissante, présente une complexité notable lorsqu'il s'agit de déboguer. En effet, le fait que les triggers s'activent sans intervention explicite de l'utilisateur rend leur suivi et leur analyse plus difficiles. Cela devient encore plus complexe lorsque ces triggers interagissent avec plusieurs tables ou sont associés à des requêtes SQL complexes. Dans de tels cas, identifier la source d'un dysfonctionnement ou d'un comportement inattendu peut nécessiter un effort considérable, car il faut souvent analyser l'ensemble du processus d'exécution pour repérer l'origine du problème.

L'utilisation de triggers peut poser des défis en matière de scalabilité. Lorsqu'une application est confrontée à un volume croissant de transactions, les triggers, qui sont conçus pour traiter automatiquement des événements spécifiques, peuvent devenir des bombes à retardement. À mesure que le nombre d'opérations augmente, le temps de traitement requis pour exécuter les triggers peut s'accumuler, ce qui pourrait ralentir l'ensemble du système. Si ces ralentissements deviennent significatifs, ils pourraient entraîner une dégradation des performances, notamment sous des charges de travail plus élevées. Dans un tel scénario, il serait

impératif de réévaluer l'architecture en place. Il pourrait être nécessaire d'optimiser ou de repenser l'implémentation des triggers, voire de les remplacer par des solutions alternatives, pour garantir que le système reste performant et capable de s'adapter à la croissance continue du volume de transactions.

Ainsi, bien que les triggers puissent offrir une automatisation précieuse dans la gestion des bases de données, leur mise en œuvre nécessite une attention particulière tant sur le plan du débogage que sur celui de la performance, surtout dans un contexte où la scalabilité est un facteur crucial.

Conclusion :

Le trigger `update_statistiques_after_vente` est un élément technique clé qui assure la mise à jour en temps réel des statistiques de vente dans l'application. Son rôle est essentiel pour garantir que les données utilisées par les administrateurs et les utilisateurs reflètent toujours les informations les plus récentes, permettant ainsi une gestion efficace et précise des ventes. Malgré les défis, ce trigger est un atout majeur pour la robustesse et la fiabilité du système de gestion des ventes.

Récupération des statistiques pour les afficher à jour

Récupération des Statistiques pour les Afficher à Jour

L'une des fonctionnalités clés de l'application de gestion des stocks et des ventes développées pour MSDM Horizon est la capacité à récupérer et afficher en temps réel des statistiques essentielles pour les utilisateurs. Ces statistiques incluent le nombre total de paires vendues, le bénéfice total généré, et le chiffre d'affaires total. De plus, une visualisation des ventes mensuelles est également proposée via un graphique interactif.

1. Récupération des Statistiques Utilisateur

Pour chaque utilisateur, les statistiques sont récupérées à partir de la base de données à l'aide du **StatistiquesRepository**. Ce dépôt interagit avec la table `Statistiques` pour extraire les données nécessaires. La méthode `getStatistiquesByUserId` est chargée de récupérer les statistiques spécifiques à l'utilisateur connecté, identifiées par son `userId`.

Figure 21 : Récupération des statistiques par utilisateurs

```
public function getStatistiquesByUserId($userId): ?Statistiques
{
    $sql = "SELECT * FROM Statistiques
           WHERE idStat = (SELECT idStat FROM Membres WHERE email = :userId)";
    $values = ['userId' => $userId];

    $pdoStatement = ConnexionBaseDeDonnee::getPdo()->prepare($sql);
    $pdoStatement->execute($values);
    $resultat = $pdoStatement->fetch();

    if ($resultat === false) {
        return null;
    }

    return $this->construireDepuisTableau($resultat);
}
```

Cette méthode exécute une requête SQL qui sélectionne les statistiques basées sur l'identifiant de l'utilisateur. Si aucune donnée n'est trouvée, elle renvoie `null`, sinon, elle construit un objet `Statistiques` à partir des données récupérées.

2. Affichage des Statistiques sur le Tableau de Bord

Dans le contrôleur `ControleurDashboard`, les statistiques récupérées sont ensuite passées à la vue `dashboard.php` pour être affichées sur le tableau de bord de l'utilisateur. Les données incluent le nombre de paires vendues, le bénéfice total, et le chiffre d'affaires total.

Figure 22 : ControleurDashboard

```
public function index()
{
    $utilisateur = ConnexionUtilisateur::getLoginUtilisateurConnecte();

    $statRepo = new StatistiquesRepository();
    $statistiques = $statRepo->getStatistiquesByUserId($utilisateur);
    $vendRepo = new VendRepository();
    $ventesParMois = $vendRepo->getVentesParMois($utilisateur);

    if ($statistiques) {
        $nbVendu = $statistiques->getNbPaireVendu();
        $beneficeTotal = $statistiques->getBeneficeTotal();
        $caTotal = $statistiques->getCATotal();
    } else {
        $nbPaireVendu = 0;
        $beneficeTotal = 0.0;
        $caTotal = 0.0;
    }

    $this->afficherVue( cheminVue: 'dashboard/dashboard.php', [
        'pagetitle' => 'Dashboard',
        'nbVendu' => $nbVendu,
        'beneficeTotal' => $beneficeTotal,
        'caTotal' => $caTotal,
        'ventesParMois' => $ventesParMois
    ]);
}
```

Ce code extrait les statistiques et les ventes par mois pour l'utilisateur courant, puis les transmet à la vue. Si les statistiques sont disponibles, elles sont affichées, sinon, des valeurs par défaut sont utilisées.

3. Visualisation des Ventes Mensuelles

La vue `dashboard.php` intègre un graphique interactif pour afficher les ventes par mois. Cette visualisation est réalisée à l'aide de la bibliothèque JavaScript **Chart.js**. Les données sont passées depuis le contrôleur sous forme de tableau JSON.

Figure 23 : Vue Dashboard avec chart.js

```
<div class="statistics-container">
  <div class="statistic-item">
    <h3>Nombre de paires vendues</h3>
    <p><?= /** @var TYPE_NAME $nbVendu */
      $nbVendu; ?></p>
  </div>
  <div class="statistic-item">
    <h3>Bénéfice Total</h3>
    <p><?= /** @var TYPE_NAME $beneficeTotal */
      $beneficeTotal; ?>€</p>
  </div>
  <div class="statistic-item">
    <h3>Chiffre d'affaires Total</h3>
    <p><?= /** @var TYPE_NAME $caTotal */
      $caTotal; ?>€</p>
  </div>
</div>

<div class="chart-container">
  <canvas id="ventesParMoisGraphique"></canvas>
</div>
```

```
<script>
var ctx = document.getElementById('ventesParMoisGraphique').getContext('2d');

// Données passées depuis le contrôleur
var labels = ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet', 'Août', 'Septembre', 'Octobre', 'Novembre', 'Décembre'];
var data = <?= /** @var TYPE_NAME $ventesParMois */
    json_encode($ventesParMois); ?>;

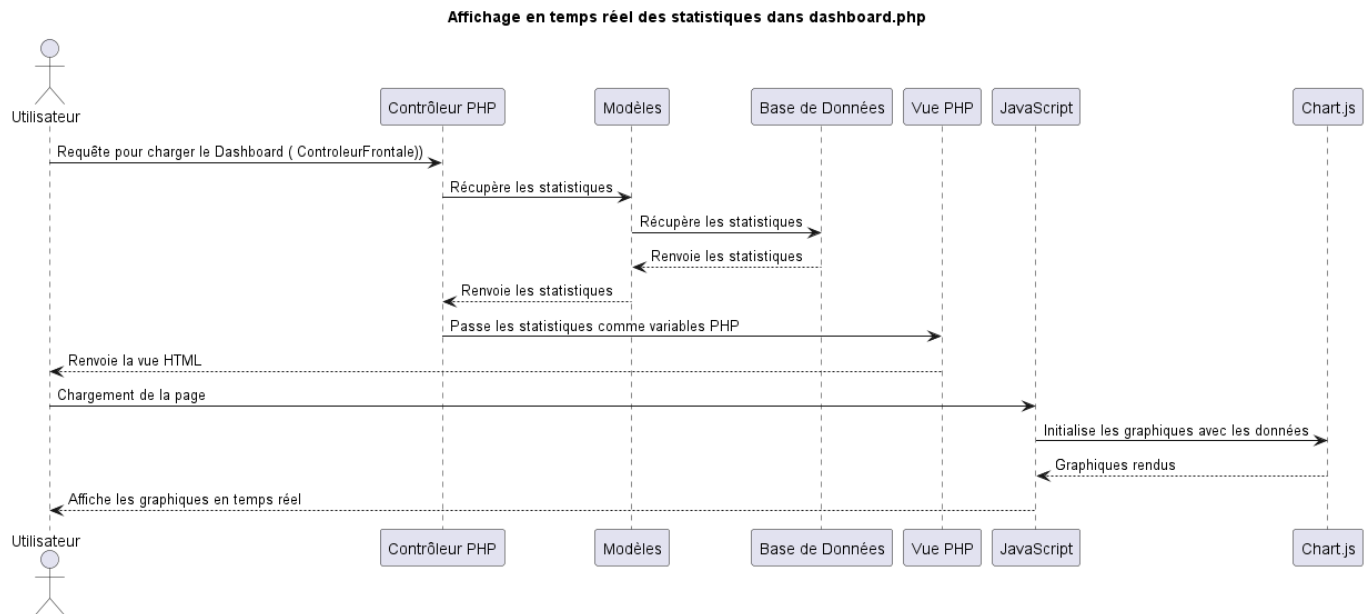
var myChart = new Chart(ctx, {
    type: 'line',
    data: {
        labels: labels,
        datasets: [{
            label: 'Ventes',
            data: data,
            backgroundColor: 'rgba(75, 192, 192, 0.2)',
            borderColor: 'rgba(75, 192, 192, 1)',
            borderWidth: 1
        }]
    },
    options: {
        responsive: true,
        maintainAspectRatio: false,
        scales: {
            y: {
                beginAtZero: true
            }
        }
    }
});
</script>
```

Ce script génère un graphique linéaire représentant le nombre de ventes par mois. Les données sont automatiquement ajustées pour inclure tous les mois, même ceux sans vente, garantissant une représentation complète de l'année.

4. Mise à Jour Dynamique

Chaque fois qu'une nouvelle vente est enregistrée, les statistiques sont automatiquement mises à jour grâce aux **triggers SQL** implémentés dans la base de données. Cela permet de maintenir les données du tableau de bord à jour en temps réel sans intervention manuelle.

Diagramme 6 : Affichage des statistiques



Conclusion

La récupération et l'affichage des statistiques à jour sont essentiels pour offrir une vue d'ensemble précise de la performance des ventes et des stocks. Grâce à l'architecture implémentée, l'application permet non seulement d'extraire ces données en temps réel, mais aussi de les visualiser de manière claire et interactive, facilitant ainsi la prise de décision pour les utilisateurs.

Un site responsif

La responsivité est un aspect crucial du développement web moderne, garantissant que les interfaces utilisateur s'adaptent correctement à différentes tailles d'écran et types d'appareils. Pour l'application de gestion des ventes, la responsivité a été soigneusement intégrée afin de garantir une expérience utilisateur fluide, que ce soit sur un ordinateur de bureau, une tablette ou un smartphone.

Structure Responsive

L'interface de l'application est divisée en deux sections principales : la barre latérale (sidebar) et le contenu principal (main-content). Cette structure en deux colonnes permet de séparer les fonctionnalités de navigation des informations affichées à l'écran. La mise en page a été conçue pour être flexible, en utilisant des pourcentages pour les largeurs, ce qui facilite l'adaptation à différentes tailles d'écran.

- **Barre latérale** : La sidebar occupe 20% de la largeur de l'écran sur les grandes résolutions, mais cette proportion change en fonction de la taille de l'écran pour s'adapter aux plus petits appareils, passant à 30% sur les tablettes et 40% sur les smartphones.
- **Contenu principal** : Le main-content occupe le reste de l'espace disponible. Il s'ajuste automatiquement pour prendre la largeur restante après la sidebar, assurant ainsi que le contenu reste bien lisible.

Media Queries

Les media queries ont été utilisées pour gérer la responsivité du site. Elles permettent d'appliquer des styles CSS spécifiques en fonction de la largeur de l'écran, assurant que l'interface reste utilisable et attrayante sur différents appareils.

- **Écrans de Tablettes (max-width: 768px) :**
 - La largeur de la sidebar passe à 30% et celle du main-content à 70%.
 - Les éléments de texte, tels que les titres et les liens dans la sidebar, voient leur taille de police légèrement réduite pour mieux s'adapter à l'écran.
 - Les éléments de la section principale, qui sont disposés côte à côte sur les grands écrans, passent en disposition verticale pour une meilleure lisibilité.
- **Écrans de Smartphones (max-width: 480px) :**
 - La sidebar occupe 40% de la largeur, et le main-content les 60% restants.
 - Les polices sont encore réduites pour optimiser l'espace disponible.
 - La disposition des éléments devient principalement verticale, ce qui permet une navigation aisée avec une seule main, caractéristique importante sur les petits écrans.

Adaptation des Composants Interactifs

Les graphiques et autres éléments interactifs, comme ceux présents dans les sections chart-container et stat-item, sont également conçus pour être responsive :

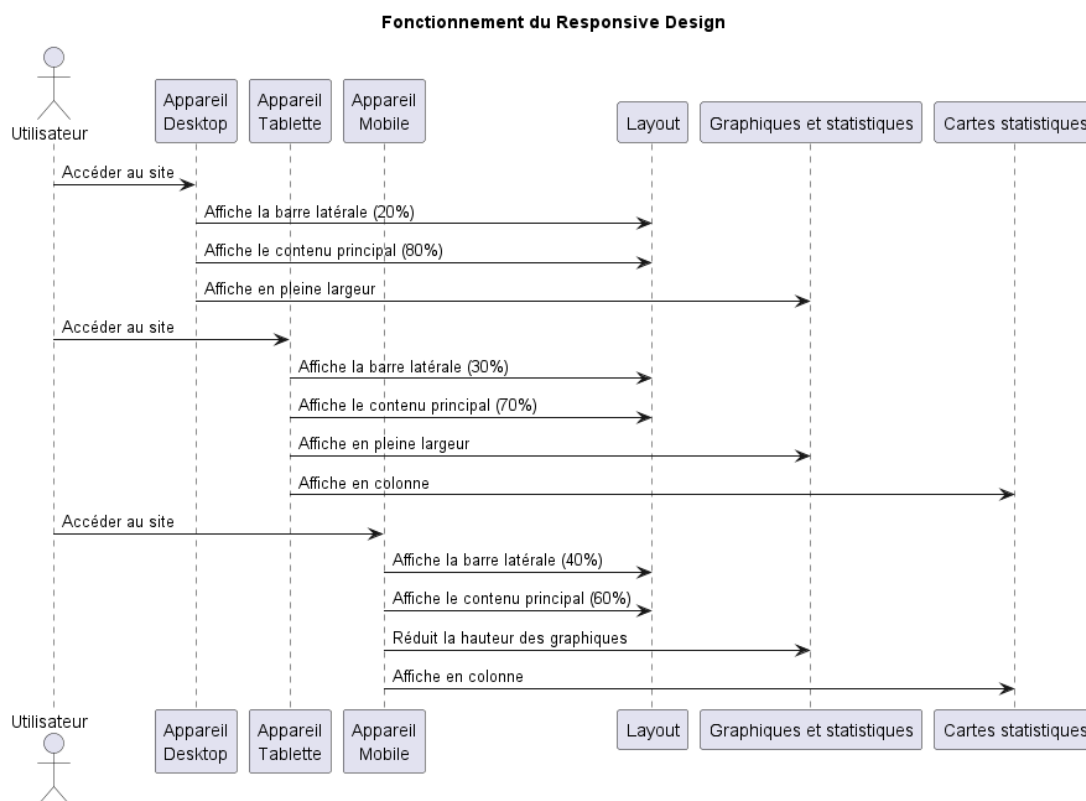
- **Graphiques** : Les graphiques utilisent des pourcentages pour leur largeur (max-width: 100%), garantissant qu'ils prennent toute la largeur disponible quel que soit l'appareil. Sur les plus petits écrans, la hauteur des graphiques est ajustée pour rester lisible sans nécessiter de défilement excessif.
- **Cartes de Statistiques** : Les statistiques passent d'une disposition côte à côte à une disposition en colonne sur les petits écrans, ce qui améliore la lisibilité et l'accessibilité des informations importantes.

Expérience Utilisateur Améliorée

En rendant le site responsive, l'objectif est de garantir que les utilisateurs puissent accéder facilement à toutes les fonctionnalités de l'application, qu'ils soient au bureau sur un écran large ou en déplacement avec un smartphone. Les media queries et les ajustements de design permettent d'offrir une expérience utilisateur cohérente et intuitive, indépendamment de la taille de l'écran ou du type d'appareil utilisé.

Cette attention particulière à la responsivité assure non seulement une meilleure accessibilité, mais contribue également à l'efficacité globale de l'application, en permettant aux utilisateurs de gérer leurs ventes et consulter leurs statistiques dans des conditions optimales, où qu'ils se trouvent.

Diagramme 7 : Fonctionnement du responsive



Rapport d'activité

Contexte du Stage

Le stage s'est déroulé au sein de l'entreprise **MSDM Horizon**, une société par actions simplifiée (SAS) inscrite sous le numéro Siren 979471539 et ayant pour siège social 1080 Route de Dieulefit, 26160 La Bégude-de-Mazenc, Drôme, France. MSDM Horizon est spécialisée dans le commerce de gros et de détail de chaussures, prêt-à-porter, et accessoires de mode, avec une présence en ligne via un site e-commerce. Fondée le 12 septembre 2023, l'entreprise cible les détaillants de mode, les boutiques spécialisées, ainsi que les consommateurs finaux à travers sa plateforme en ligne.

Enjeux du Stage

L'accueil d'un stagiaire représentait pour MSDM Horizon une opportunité de renforcer ses processus de vente en ligne et d'optimiser la gestion de ses stocks. Le stage s'inscrit dans une stratégie globale de croissance et d'efficacité opérationnelle, avec pour objectif de développer une application web de gestion des stocks et des ventes.

Analyse du Sujet

La mission principale a consisté en la conception et le développement d'une application web de gestion des stocks et des ventes. Cette application visait à améliorer la gestion des articles en stock, à suivre les ventes et à générer des statistiques pour faciliter la prise de décision. Les tâches principales incluaient la conception de la base de données, le développement du backend en PHP en utilisant le modèle MVC, la création de l'interface utilisateur en HTML, CSS et JavaScript, ainsi que l'implémentation de fonctionnalités de gestion des utilisateurs, d'inventaire, de ventes et de statistiques.

Analyse de l'Environnement Technique

L'application a été développée selon une architecture MVC, avec une base de données MySQL pour gérer les données. L'interface utilisateur a été conçue pour être intuitive et réactive, avec une attention particulière portée à la sécurité des données, notamment à travers le hachage et le salage des mots de passe.

Modélisation de la Base de Données

La base de données a été structurée pour inclure des tables représentant les utilisateurs, les articles en stock, les ventes, et les statistiques. Un schéma relationnel a été créé pour assurer une gestion efficace des données et une automatisation des statistiques grâce à des triggers SQL.

Développement des Fonctionnalités

Les fonctionnalités principales de l'application incluent la gestion des utilisateurs (création de comptes, connexion), la gestion des articles (ajout, modification, suppression,), la gestion des ventes (enregistrement et suivi), et la génération de statistiques de vente. Un effort particulier a été porté sur l'amélioration de l'interface utilisateur.

Focus sur des Aspects Techniques

L'intégration de Chart.js a permis de visualiser les statistiques de vente sur le tableau de bord de l'application. La gestion des données dynamiques a été simplifiée en utilisant un préchargement via PHP, ce qui a amélioré la performance et la robustesse de l'application.

Conclusion

Le stage a permis de développer une application robuste et sécurisée pour la gestion des stocks et des ventes, répondant aux besoins spécifiques de MSDM Horizon. Ce projet a non seulement contribué à l'efficacité opérationnelle de l'entreprise, mais a également offert une expérience précieuse en développement web et en gestion de projet.

Proposition d'évolution de l'application

Afin d'améliorer encore l'efficacité et la fonctionnalité de l'application de gestion des stocks et des ventes, plusieurs évolutions sont proposées :

1. Ajout d'un Trigger sur Suppression et Modification de Vente pour les Statistiques

Actuellement, un trigger SQL met à jour automatiquement les statistiques de vente après chaque nouvelle transaction. Toutefois, pour garantir une précision optimale des données, il est recommandé d'étendre cette automatisation aux cas de suppression et de modification de ventes. En ajoutant un trigger spécifique pour ces opérations, les statistiques seraient mises à jour en temps réel chaque fois qu'une vente est annulée ou modifiée, assurant ainsi que le bénéfice total, le chiffre d'affaires et le nombre d'articles vendus reflètent toujours la réalité du stock et des transactions.

2. Ajout d'un Compte Acheteur avec Accès à Toutes les Paires en Vente

Une autre évolution potentiellement bénéfique serait l'introduction de comptes spécifiques pour les acheteurs. Ces comptes permettraient aux utilisateurs enregistrés en tant qu'acheteurs d'accéder directement à toutes les paires en vente sur la plateforme, facilitant ainsi le processus d'achat et élargissant la base de clients potentiels. Cette fonctionnalité pourrait inclure des options de filtrage et de recherche avancées pour permettre aux acheteurs de trouver rapidement les articles qui les intéressent, tout en offrant une expérience d'achat plus personnalisée et fluide.

Ces propositions visent à renforcer la robustesse de l'application tout en offrant de nouvelles opportunités de croissance pour MSDM Horizon.

Méthodologie et organisation du projet

Organisation du travail dans l'entreprise :

Approche Agile en travail individuel

Bien que j'aie travaillé seule sur le projet, j'ai adopté une méthodologie agile pour structurer mon travail et maximiser l'efficacité de chaque phase de développement. L'approche agile, habituellement utilisée dans les équipes pour faciliter la collaboration, a été adaptée à mon contexte de travail individuel pour maintenir un rythme soutenu et itératif.

Structure des Sprints

J'ai organisé mon travail en sprints de deux semaines. Chaque sprint débutait par une planification où je définissais les objectifs précis à atteindre. Les tâches à accomplir étaient listées dans un backlog, que je priorisais en fonction de l'urgence et de l'importance des fonctionnalités à développer. Cette approche m'a permis de segmenter le projet en étapes gérables et de maintenir une vision claire des objectifs à court terme.

Gestion du Backlog

Le backlog servait de tableau de bord central pour suivre l'avancement du projet. J'y notais toutes les fonctionnalités à développer, les bugs à corriger, et les améliorations à apporter. Chaque élément du backlog était détaillé avec des critères d'acceptation clairs, ce qui facilitait la gestion des tâches et permettait de mesurer objectivement l'accomplissement des objectifs à la fin de chaque sprint.

Revue de Sprint (Sprint Review)

À la fin de chaque sprint, je réalisais une revue de sprint pour évaluer les progrès accomplis. Cette étape consistait à vérifier si les fonctionnalités développées répondaient aux critères définis et si elles étaient prêtes pour une éventuelle mise en production. Lors de cette revue, j'identifiais également les tâches non terminées ou les problèmes rencontrés, que j'intégrais au backlog du sprint suivant. Cela m'a permis de maintenir une continuité dans le projet et de m'assurer que chaque fonctionnalité était pleinement opérationnelle avant de passer à la suivante.

Adaptation et amélioration continue

L'un des principes fondamentaux de la méthodologie agile est l'amélioration continue. Après chaque sprint, j'évaluais non seulement les résultats obtenus, mais aussi les processus utilisés. Si un aspect du développement ou de la planification nécessitait des ajustements, je modifiais mon approche pour le sprint suivant. Par exemple, si une tâche s'avérait plus complexe que prévu, je prenais soin de mieux la décomposer ou de réévaluer ses priorités dans le backlog. Cette flexibilité m'a permis d'adapter mon rythme de travail aux réalités du projet, tout en restant concentrée sur les objectifs finaux.

Autogestion et discipline

Travailler seule tout en appliquant une méthodologie agile a nécessité une grande discipline. Chaque jour, je faisais un point sur mes avancées, identifiant les obstacles potentiels et ajustant mon planning si nécessaire. Cette autodiscipline a été essentielle pour respecter les délais que je m'étais fixés et pour assurer une progression régulière tout au long du projet.

En somme, l'application de la méthodologie agile m'a permis de structurer mon travail de manière efficace, de rester flexible face aux imprévus, et de maintenir un haut niveau de productivité. Cette approche a également facilité la gestion des priorités et m'a aidée à livrer un produit final de qualité, conforme aux attentes définies au début du projet.

Méthodes de développement et de travail personnel :

Autonomie et proactivité

Au cours de mon stage, j'ai bénéficié d'un haut degré d'autonomie, ce qui m'a permis de prendre en charge diverses responsabilités, notamment la conception de la base de données et le développement du back-end de l'application. Cette autonomie m'a permis de développer une capacité à résoudre des problèmes de manière proactive, en recherchant des informations pertinentes et en proposant des solutions adaptées aux besoins du projet.

Démarche personnelle

Lorsque des problèmes techniques se sont posés, j'ai pris l'initiative de les analyser en profondeur, en consultant la documentation officielle et en regardant des tutoriels en ligne pour explorer différentes solutions possibles.

Initiatives prises

J'ai également pris l'initiative de renforcer la sécurité de l'application en implémentant des mécanismes de hachage et de salage des mots de passe, ainsi qu'en améliorant l'interface utilisateur pour une meilleure expérience utilisateur. Ces initiatives ont non seulement contribué à la robustesse de l'application, mais ont aussi permis de rendre le système plus convivial et sécurisé.

Travail effectué en plus du projet principal :

En parallèle du développement de l'application de gestion des stocks et des ventes, j'ai également participé à d'autres tâches diversifiées, comme l'aide à différents besoins de l'entreprise notamment de la logistique ou des conseils plus variés.

Cette section du rapport met en évidence mon implication active dans le projet, ainsi que ma capacité à travailler de manière autonome tout en étant un membre efficace d'une équipe. Elle illustre également les méthodes de travail rigoureuses et collaboratives en vigueur chez MSDM Horizon, ainsi que mon adaptation à ces méthodes pour contribuer au succès du projet.

Conclusion

Au terme de mon stage chez MSDM Horizon, j'ai pu développer une application web de gestion des stocks et des ventes qui répond pleinement aux besoins de l'entreprise. Cette application a été conçue pour être facilement reprenable et maintenue par l'entreprise, grâce à une architecture MVC claire et une documentation soignée. Le projet a significativement amélioré le système d'information de MSDM Horizon en optimisant la gestion des articles en stock, en automatisant la génération de statistiques. Ces améliorations ont permis d'accroître l'efficacité opérationnelle de l'entreprise et d'offrir une meilleure expérience.

Sur le plan personnel, ce stage a été particulièrement formateur dans plusieurs domaines clés. Tout d'abord, j'ai renforcé mes compétences en **développement back-end**, notamment en PHP avec le modèle MVC, ce qui m'a permis de structurer mon code de manière modulaire et maintenable. Par exemple, la mise en place de triggers SQL pour automatiser les mises à jour des statistiques de vente a été une expérience enrichissante qui m'a permis de mieux comprendre les interactions complexes entre différentes couches d'une application.

Ensuite, j'ai approfondi mes connaissances en **sécurité informatique**, en mettant en œuvre des techniques avancées de protection des données, comme le hachage et le salage des mots de passe. Cette approche m'a sensibilisé à l'importance de sécuriser les informations utilisateurs, un aspect crucial dans le développement d'applications web modernes.

En outre, ce projet m'a permis de développer une **expertise en gestion de bases de données** relationnelles. J'ai conçu et optimisé la base de données de manière à assurer une performance efficace et une manipulation fluide des données, même sous des charges de travail importantes. La création de schémas relationnels et l'implémentation de triggers SQL ont été des points forts de cette expérience.

Enfin, j'ai acquis une solide compétence en **gestion de projet** et en méthodologie agile, en organisant mon travail de manière structurée avec des sprints réguliers et des révisions de sprint. Cela m'a permis de livrer un produit final de qualité avec un minimum de fonctionnalités tout en respectant les délais imposés, tout en m'adaptant aux imprévus et en optimisant continuellement le processus de développement.

En conclusion, ce stage a non seulement été une opportunité de contribuer de manière tangible à l'amélioration du système d'information de MSDM Horizon, mais il m'a également permis de développer des compétences techniques et professionnelles essentielles qui seront précieuses dans ma future carrière en informatique.

Visa du maître de stage

Vue le 19/08/2024

Bibliographie

- [1]
Manuel PHP, PHP Documentation Group : <https://www.php.net/manual/en/index.php>
- [2]
Documentation MySQL, Oracle Corporation. MySQL Reference Manual :
<https://dev.mysql.com/doc/refman/en/>
- [3]
Documentation Chart.js : <https://www.chartjs.org/docs/>
- [4]
D.M. Selfa; M. Carrillo; M. Del Rocio Boone (13 mars 2006). A Database and Web Application Based on MVC Architecture : <https://ieeexplore.ieee.org/abstract/document/1604744>
- [5]
Laurent Szpirglas (mars 2022). Cryptage, hachage et salage - Quelle est la différence ? :
<https://www.globalsecuritymag.fr/Cryptage-hachage-et-salage-Quelle,20220309,122892.html>
- [6]
Jean-Luc Hainaut (2015). Bases de données - Concepts, utilisation et développement. Dunod
- [7]
Janet Valade (2002). PHP et MySQL pour les Nuls
- [8]
Ecole du web. (27 février 2022) Chart.js Tutorial - Codez des graphiques facilement avec Chart.js. YouTube : <https://www.youtube.com/watch?v=NSCJ9jIUnSI>

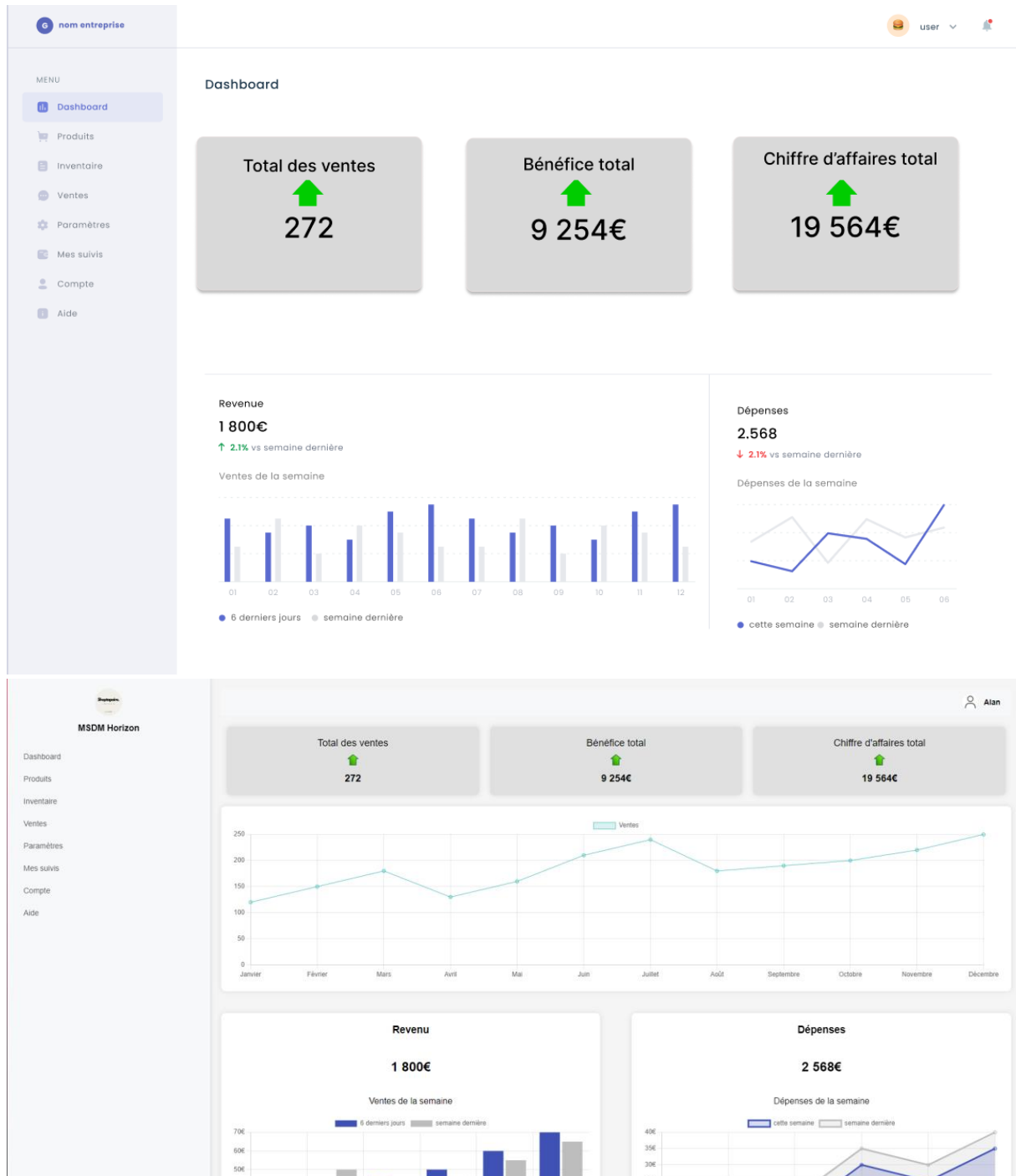
Annexes

Conception de l'interface graphique sur Figma vs ce qui a été réalisé:

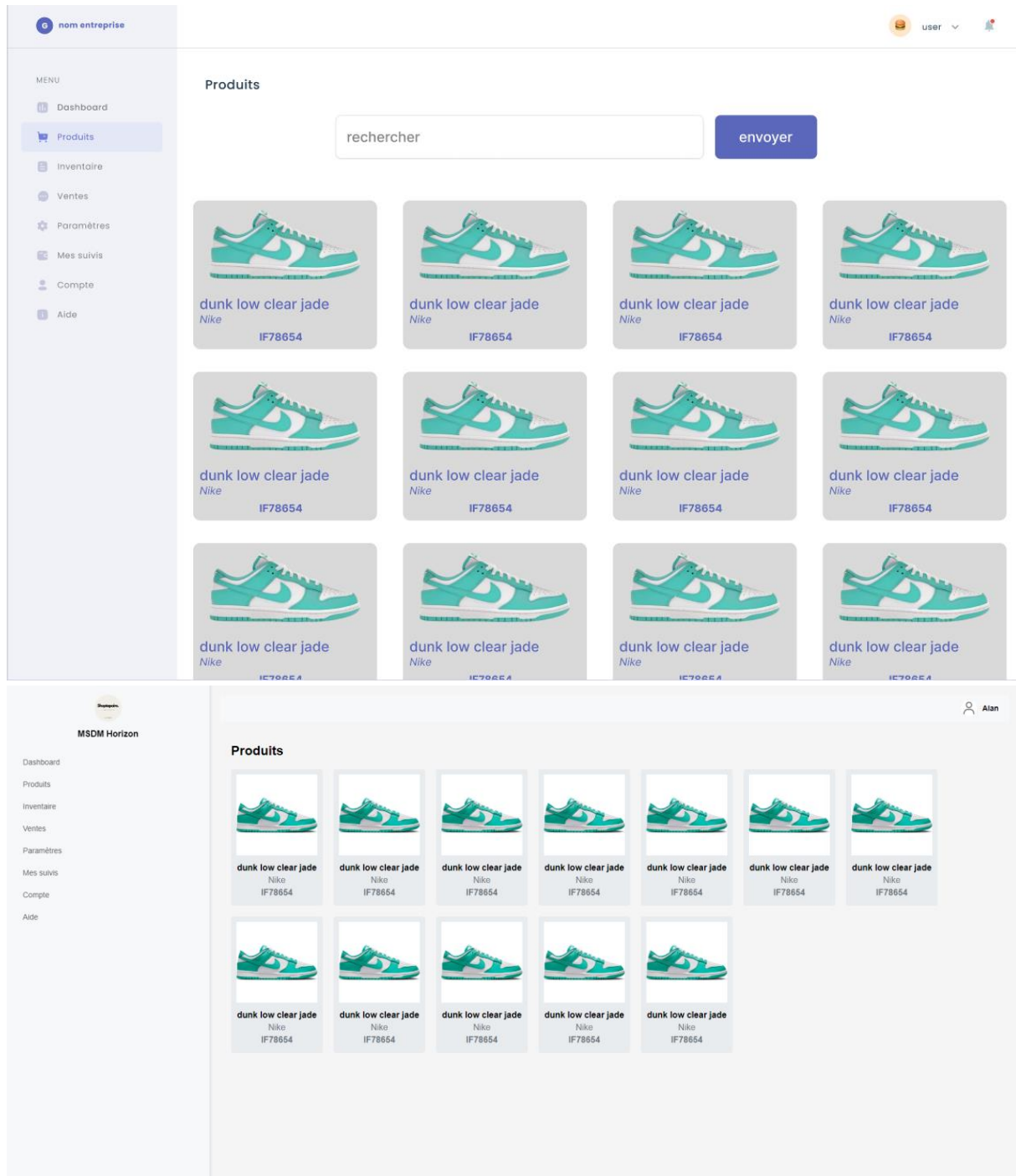
Lien vers Figma : [https://www.figma.com/design/MPyBGMZ2PHno30PotBLX61/Dashboard-\(Community\)?node-id=125-146](https://www.figma.com/design/MPyBGMZ2PHno30PotBLX61/Dashboard-(Community)?node-id=125-146)

Développement d'une Application Web de Gestion des Stocks et des Ventes pour MSDM

Horizon – Alan TERRIER



Développement d'une Application Web de Gestion des Stocks et des Ventes pour MSDM Horizon – Alan TERRIER



Développement d'une Application Web de Gestion des Stocks et des Ventes pour MSDM
Horizon – Alan TERRIER

MSDM Horizon

Dashboard

Produits

Inventaire

Ventes

Paramètres

Mes suivis

Compte

Aide

Alan

Ventes

Produit	Taille	Quantité	Achat	Vente	Lieu	Date Achat	Date Vente	Bénéfice	Statut	Suivi
Dunk low clear jade	39	1	100€	140€	Vinted	01/01/23	01/01/24	40€	reçu	suivre
Adidas campus 00s dark green	37	2	85€	120€	Discord	25/11/23	11/05/24	70€	en attente	suivre

Développement d'une Application Web de Gestion des Stocks et des Ventes pour MSDM
Horizon – Alan TERRIER

Mes suivis

Dashboard

Produits

Inventaire

Ventes

Paramètres

Mes suivis

Compte

Aide

Alan

Mes suivis

Produit	Taille	Quantité	Achat / Vente	Suivis	Statut	Type	Action
Dunk low clear jade	39	1	01/01/23	6A42635005792	Il est trié sur la plateforme de départ.	vente	<button>suivre</button>
Adidas campus 00s dark green	37	2	25/11/23	6A42635005792	Il est trié sur la plateforme de départ.	achat	<button>suivre</button>

localhost/marketplace/public/controlleurFrontal.php?action=index&controlleur=inventaire

Développement d'une Application Web de Gestion des Stocks et des Ventes pour MSDM Horizon – Alan TERRIER

6 nom entreprise

MENU

- Dashboard
- Produits
- Inventaire
- Ventes
- Paramètres
- Mes suivis
- Compte
- Aide

user

Compte

prenom

nom

mail

téléphone

ville

facebook

discord

mettre à jour

changer mot de passe

supprimer mon compte

déconnexion

MSDM Horizon

Dashboard

Produits

Inventaire

Ventes

Paramètres

Mes suivis

Compte

Aide

Compte

prenom

nom

mail

téléphone

ville

facebook

discord

mettre à jour

changer mot de passe

supprimer mon compte

déconnexion

Développement d'une Application Web de Gestion des Stocks et des Ventes pour MSDM Horizon – Alan TERRIER

6 nom entreprise

MENU

- Dashboard
- Produits
- Inventaire
- Ventes
- Paramètres
- Mes suivis
- Compte
- Aide

user

Aide

Contactez nous :

Ou :

mail

contact@entreprise.com

téléphone

0697536450

Hésitez pas à consulter notre FAQ :

FAQ

MSDM Horizon

Dashboard

Produits

Inventaire

Ventes

Paramètres

Mes suivis

Compte

Aide

Aide

Contactez-nous :

Envoyer

Ou :

mail

contact@entreprise.com


téléphone

0697536450

N'hésitez pas à consulter notre FAQ :

FAQ

localhost/marketplace/public/controleurfrontal.php?action=index&controleur=Dashboard

 nom entreprise

Connexion

mail

mot de passe


connexion

Connexion

mail

mot de passe

connexion

 nom entreprise

Inscription

☐

créter
compte

☐ J'accepte les conditions d'utilisations *

Inscription

☐ J'accepte les conditions d'utilisations *

créter
compte