



République du Sénégal

Un Peuple –Un But – Une Foi

-----

Ministère de l'Enseignement Supérieur de la Recherche et de L'Innovation



UNIVERSITE IBA DER THIAM DE THIES

UFR : SCIENCES ECONOMIQUES ET SOCIALES

DEPARTEMENT : MANAGEMENT DES ORGANISATIONS (M.O)

FILIERE : SCIENCES DES DONNEES ET APPLICATIONS

OPTION : CONTROLE AUDIT ET DATA ANALYTICS.

## ***PROJET INTERPHASAGE BASE DE DONNEES AVEC PYTHON ET MYSQL***

Présenté par :

NDEYE AMINATA FALL BA

JOEL DIAHOU GUEYE

ENCADREMENT :

DOCTORANT YOUSOUF LY

**Année Académique : 2023-2024**

# SOMMAIRE

<b>INTRODUCTION .....</b>	<b>3</b>
<b>CONCEPTION DE LA BASE DE DONNEES.....</b>	<b>3</b>
<b>STRUCTURE DES TABLES .....</b>	<b>4</b>
<b>ILLUSTRATION .....</b>	<b>4</b>
<b>ARCHITECTURE DE L'APPLICATION .....</b>	<b>8</b>
<b>STRUCTURE DE L'APPLICATION .....</b>	<b>8</b>
<b>EXPLICATION DU CODE .....</b>	<b>9</b>
<b>TESTS ET DEBOGAGE .....</b>	<b>15</b>
<b>CONCLUSION.....</b>	<b>21</b>

## INTRODUCTION

Dans un contexte de gestion moderne, les supermarchés nécessitent des solutions technologiques pour gérer efficacement leurs stocks, leurs ventes et leurs fournisseurs. Ce projet consiste en la création d'une application de gestion pour un supermarché, permettant de centraliser et d'automatiser ces processus afin de répondre aux défis quotidiens liés à la gestion de l'inventaire, aux interactions avec les fournisseurs, et au suivi des ventes.

L'application vise à simplifier la gestion des produits et des transactions au sein d'un supermarché en permettant un suivi en temps réel des stocks, avec mise à jour automatique lors de chaque vente, une gestion structurée des fournisseurs, permettant de s'assurer de la disponibilité continue des produits et une visualisation claire et intuitive des produits disponibles, facilitant la gestion quotidienne du magasin.

Cette application a été développée en utilisant des technologies modernes et robustes comme *Python et Flask* pour le développement de l'interface serveur et des fonctionnalités logicielles de l'application et *MySQL* pour le stockage structuré et sécurisé des données. Nous avons aussi utilisé les langages *HTML et CSS* pour la présentation visuelle des pages et une expérience utilisateur agréable.

En répondant aux besoins spécifiques d'un supermarché, cette application offre une solution complète, permettant d'optimiser la gestion et d'améliorer l'efficacité opérationnelle.

## CONCEPTION DE LA BASE DE DONNEES

Pour garantir une gestion efficace des produits et des ventes, la base de données de l'application a été conçue autour de trois tables principales : Produit, Fournisseur, et Vente. Chacune de ces tables stocke des informations spécifiques nécessaires au bon fonctionnement du supermarché.

Nous avons d'abord connecté MySQL avec la base de données en suivant différentes étapes :

- **Host='localhost'** : indique que le serveur MySQL est hébergé localement sur votre machine. Si la base de données était hébergée sur un serveur distant, vous indiqueriez son adresse IP ou nom de domaine ici.

- **User='root'** : désigne le nom d'utilisateur de la base de données. Dans cet exemple, vous vous connectez en tant qu'utilisateur root, qui a généralement des privilèges d'administrateur.
- **Password='Atanima3026@'** : spécifie le mot de passe associé à l'utilisateur root. Assurez-vous de sécuriser votre mot de passe si vous le conservez dans le code.
- **Database='magasin'** : indique le nom de la base de données que vous souhaitez utiliser (ici, la base de données s'appelle magasin). Cette connexion se fera uniquement sur cette base de données.
- **Raise\_on\_warnings=True** : ce paramètre, lorsqu'il est activé, permet à Python de lever des exceptions pour les avertissements provenant de MySQL. Cela aide à identifier les erreurs potentielles.

## STRUCTURE DES TABLES

### 1. Table Produit

- Cette table contient les informations de base sur chaque produit disponible dans le supermarché, telles que son nom, son prix et son stock actuel. Les colonnes de cette table incluent :
  - Nom (VARCHAR) : Nom du produit.
  - Prix (DECIMAL) : Prix unitaire du produit.
  - Stock (int) : Quantité en stock du produit.

**Exemple d'utilisation** : La table Produit permet de gérer facilement les niveaux de stock et de mettre à jour les informations sur le prix des produits en cas de changements.

**ILLUSTRATION :**

```

16 def creer_tables():
17     cursor.execute("""
18     CREATE TABLE IF NOT EXISTS Produit (
19         id INT AUTO_INCREMENT PRIMARY KEY,
20         nom VARCHAR(100) NOT NULL,
21         prix DECIMAL(10, 2) NOT NULL,
22         stock INT NOT NULL
23     );
24     """)
25     cursor.execute("""

```

Il est à noter que les produits peuvent être modifiés ou supprimés afin de permettre la mise à jour dans le logiciel

- Modification du produit

```

# Modifier un produit
@app.route('/produit/modifierproduit/<int:produit_id>', methods=['GET', 'POST'])
def modifierproduit(produit_id):
    conn = creer_connexion()
    if conn:
        cursor = conn.cursor()
        if request.method == 'POST':
            nom = request.form['nom']
            prix = request.form['prix']
            stock = request.form['stock']
            try:
                cursor.execute("UPDATE Produit SET nom=%s, prix=%s, stock=%s WHERE id=%s",
                                (nom, prix, stock, produit_id))
                conn.commit()
                flash("Produit mis à jour avec succès.")
            except Error as e:
                conn.rollback()
                flash(f"Erreur : {e}")
            finally:

```

Il est connecté avec le fichier HTML Modifierproduit qui contient les mêmes tables que la base SQL Produit.

Ce fichier nous permettra facilement de mettre à jour les informations sur les produits.

```
# fournisseurs.css  < produit.html  < bienvenue.html  # style.css  < modifierproduit.html X
templates > < modifierproduit.html > html > body > form
2  <html lang="en">
3  <head>
8  </head>
9  <body>
10  <h1>Modifier le produit</h1>
11  <form method="POST">
12      <label>Nom :
13      |   <input type="text" name="nom" value="{{ produit[1] }}" required/><br>
14      </label>
15
16      <label>Prix :
17      |   <input type="number" name="prix" value="{{ produit[2] }}" step="0.01" required/>
18      </label>
19
20      <label>Stock :
21      |   <input type="number" name="stock" value="{{ produit[3] }}" required/><br>
22      </label>
23
24      <button type="submit">Mettre à jour</button>
25  </form>
```

- Suppression du produit

```
# Supprimer un produit
@app.route('/produit/supprimerproduit/<int:produit_id>')
def supprimerproduit(produit_id):
    conn = creer_connexion()
    if conn:
        cursor = conn.cursor()
        try:
            cursor.execute("DELETE FROM Produit WHERE id = %s", (produit_id,))
            conn.commit()
            flash("Produit supprimé avec succès.")
        except Error as e:
            conn.rollback()
            flash(f"Erreur : {e}")
        finally:
            conn.close()
    return redirect(url_for('afficher_produits'))
```

Il est directement lié au fichier HTML supprimerproduit et au fichier produit afin de gerer les modifications dans la base de donnée.

## 2. Table Fournisseur

Elle Contient les informations sur les fournisseurs associés aux produits.

- **Id** (INT, clé primaire) : Identifiant unique pour chaque fournisseur.

- **Nom** (VARCHAR) : Nom du fournisseur.
- **Contact** (VARCHAR) : Coordonnées du fournisseur.

```

25 cursor.execute("""
26 CREATE TABLE IF NOT EXISTS Fournisseur (
27     id INT AUTO_INCREMENT PRIMARY KEY,
28     nom VARCHAR(100) NOT NULL,
29     contact VARCHAR(100)
30 );
31 """)

```

### 3. Table Vente

Cette table enregistre toutes les ventes réalisées dans le supermarché. À chaque vente, elle note le produit vendu et la quantité, et permet de déduire automatiquement cette quantité du stock disponible.

- **Id** (INT, clé primaire) : Identifiant unique pour chaque vente.
- **Produit\_id** (INT, clé étrangère) : Identifiant du produit vendu, référencé dans Produit.
- **Quantite** (INT) : Quantité du produit vendu.
- **Date\_vente** (DATETIME) : Date et heure de la vente.

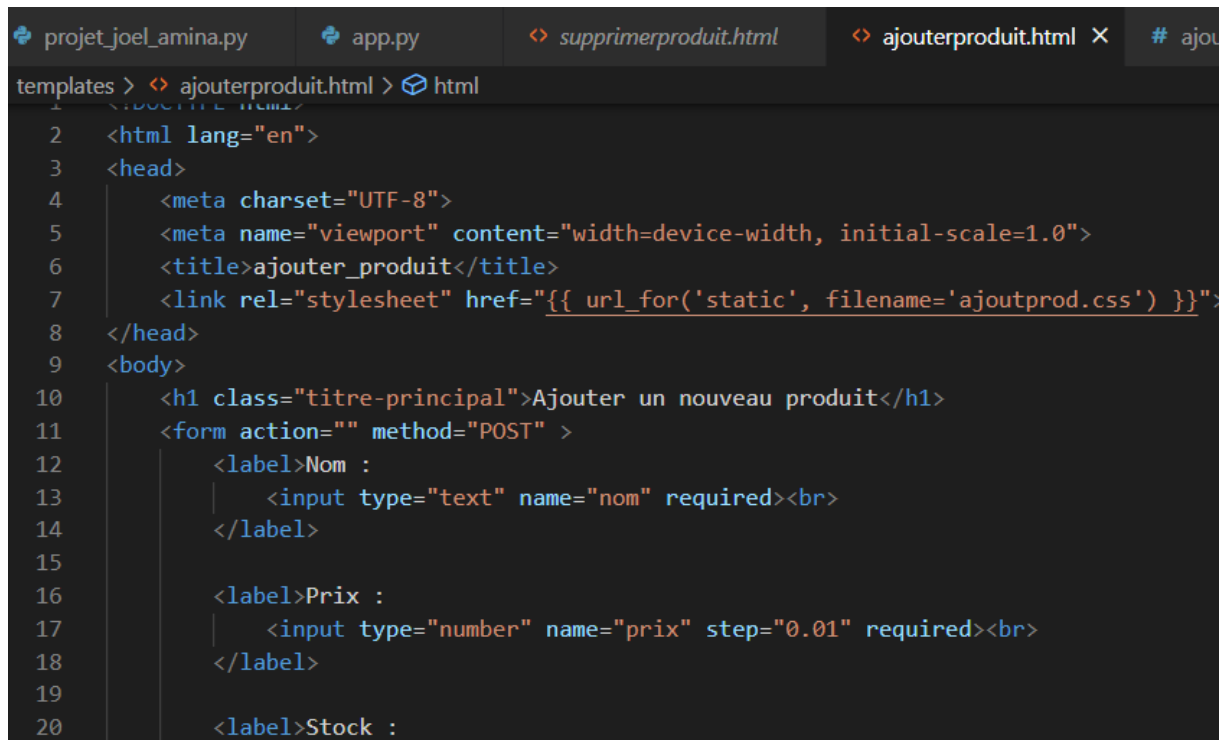
```

cursor.execute("""
CREATE TABLE IF NOT EXISTS Vente (
    id INT AUTO_INCREMENT PRIMARY KEY,
    produit_id INT,
    quantite INT,
    date_vente DATETIME DEFAULT NOW(),
    FOREIGN KEY (produit_id) REFERENCES Produit(id)
);
""")

```

**Exemple d'utilisation** : La table Vente permet de suivre les tendances de vente des différents produits, ce qui aide le supermarché à analyser la demande et à ajuster les stocks en fonction des périodes de forte affluence.

Il est lié au fichier html ajoutervente.html qui affichera les informations requises afin que l'utilisateur puisse les soumettre.

A screenshot of a code editor with a dark theme. The top bar shows several tabs: 'projet\_joel\_amina.py', 'app.py', 'supprimerproduit.html', 'ajouterproduit.html' (which is active), and a partial tab '# ajou'. The editor content shows the HTML structure of 'ajouterproduit.html'. It starts with a DOCTYPE declaration, followed by an HTML tag with lang="en". The head section includes a meta charset="UTF-8", a meta viewport="width=device-width, initial-scale=1.0", a title "ajouter\_produit", and a link to a stylesheet using a Jinja2 template. The body section contains a main heading "Ajouter un nouveau produit" and a form with three fields: "Nom" (text input), "Prix" (number input with a step of 0.01), and "Stock" (text input).

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>ajouter_produit</title>
7   <link rel="stylesheet" href="{{ url_for('static', filename='ajoutprod.css') }}">
8 </head>
9 <body>
10  <h1 class="titre-principale">Ajouter un nouveau produit</h1>
11  <form action="" method="POST">
12    <label>Nom :
13    |   <input type="text" name="nom" required><br>
14    </label>
15
16    <label>Prix :
17    |   <input type="number" name="prix" step="0.01" required><br>
18    </label>
19
20    <label>Stock :
```

La conception de cette base de données est structurée de manière à répondre aux besoins de gestion d'un supermarché. Elle permet une gestion simplifiée des stocks, un suivi des fournisseurs et une analyse des ventes, facilitant ainsi la prise de décisions pour l'achat et le réapprovisionnement.

## ARCHITECTURE DE L'APPLICATION

L'architecture de notre application est basée sur le modèle MVC (Modèle-Vue-Contrôleur), ce qui permet de séparer la logique de gestion des données, l'interface utilisateur, et le contrôle des actions. Cette architecture rend le code plus organisé et maintenable.

### STRUCTURE DE L'APPLICATION

L'application est organisée autour des composants suivants :

#### 1. **Modèle (Base de Données) :**

- La base de données MySQL est au cœur de l'application, contenant les tables essentielles comme Produit, Fournisseur, et Vente.



- La connexion à la base de données est gérée via la fonction `creer_connexion()`, qui établit une communication avec MySQL.
- Les requêtes SQL permettent de manipuler les données en ajoutant, modifiant, supprimant ou lisant les informations dans les tables.

## 2. **Vue (Templates HTML) :**

- Les fichiers HTML utilisés pour chaque vue (comme `produit.html`, `ajouterproduit.html`, `modifierproduit.html`, etc.) affichent les données et fournissent des formulaires pour les interactions de l'utilisateur.
- Flask utilise `render_template` pour relier les fichiers HTML aux routes. Ces vues montrent les messages flash (notifications) pour informer les utilisateurs des actions (ajout, modification, suppression).

## 3. **Contrôleur (Routes et Logique d'Application) :**

- Les routes définies dans l'application Flask agissent en tant que contrôleurs. Par exemple, `/produit` affiche les produits, et `/produit/modifierproduit/<int:produit_id>` permet de modifier un produit.
- Les contrôleurs gèrent la logique de traitement des données et font appel aux modèles pour interagir avec la base de données, puis redirigent vers les vues pour afficher les résultats.
- Chaque route est associée à une fonction qui exécute une action spécifique, comme ajouter, modifier, ou supprimer un produit.

## 4. **Gestion des Erreurs et Notifications :**

- Les erreurs de connexion ou d'exécution des requêtes SQL sont capturées par des blocs `try-except`, avec des messages affichés aux utilisateurs grâce à flash.
- Cela permet de gérer de manière fluide les erreurs et de notifier l'utilisateur de la réussite ou de l'échec des opérations.

Cette architecture modulaire assure une bonne organisation du code, favorise la réutilisation, et simplifie le développement de nouvelles fonctionnalités pour l'application.

## **EXPLICATION DU CODE**

Ce code ci dessous nous permettra d'importer des bibliothèques.

```
projet_joel_amina.py  app.py  x  <> supprimerproduit.html  <> ajouterproduit.html
app.py > ❷ creer_connexion
1  from flask import Flask, render_template, request, redirect, url_for, flash
2  import mysql.connector
3  from mysql.connector import Error
4
```

**Flask** : Framework web pour créer des applications web en Python.

**mysql.connector** : Module pour se connecter à une base de données MySQL.

**Error** : Classe pour gérer les erreurs de connexion à la base de données.

Nous pouvons maintenant initialiser le code

```
app = Flask(__name__)
app.secret_key = 'votre_cle_secrete'
```

**app** : Instance de l'application Flask.

**secret\_key** : Utilisée pour sécuriser les sessions de l'application et les messages flash.

Ensuite nous allons connecter MySQL avec la base de donnée

```
# Fonction pour établir la connexion à la base de données
def creer_connexion():
    try:
        conn = mysql.connector.connect(
            host='localhost',
            user='root',
            password='Atanima3026@',
            database='magasin',
        )
        return conn
    except Error as e:
        print(f"Erreur lors de la connexion à la base de données : {e}")
        return None
```

**creer\_connexion** : Fonction qui établit une connexion à la base de données MySQL. Si la connexion échoue, elle renvoie None et affiche l'erreur.

Route de la Page d'Accueil

```

22 # Page de bienvenue avec instructions
23 @app.route('/')
24 def bienvenue():
25     message = "Bienvenue! Utilisez le menu pour ajouter ou gérer des produits et fournisseurs"
26     return render_template('bienvenue.html', message=message)
27

```

**@app.route('/') :** Définition de la route pour la page d'accueil.

**bienvenue :** Fonction qui renvoie un message de bienvenue à la vue du fichier html  
bienvenue.html.

Nous afficherons tous les produits disponibles

```

# Afficher tous les produits
@app.route('/produit')
def afficher_produits():
    conn = creer_connexion()
    if conn:
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT * FROM produit")
        produit = cursor.fetchall()
        cursor.close()
        conn.close()
        return render_template('produit.html', produit=produit)
    else:
        flash("Erreur de connexion à la base de données.")
        return redirect(url_for('bienvenue'))

```

**@app.route('/produit') :** Route pour afficher tous les produits.

**afficher\_produits :** Fonction qui récupère tous les produits de la base de données et les passe à la vue du fichier produit.html. En cas d'erreur de connexion, elle redirige vers la page d'accueil avec un message d'erreur.

Nous pouvons ajouter un produit

```

# Ajouter un produit
@app.route('/ajouterproduit', methods=['GET', 'POST'])
def ajouterproduit():
    if request.method == 'POST':
        nom = request.form['nom']
        prix = request.form['prix']
        stock = request.form['stock']
        conn = creer_connexion()
        if conn:
            cursor = conn.cursor()
            try:
                cursor.execute("INSERT INTO Produit (nom, prix, stock) VALUES (%s, %s, %s)"
                                % (nom, prix, stock))
                conn.commit()
                flash("Produit ajouté avec succès.")
            except Error as e:
                conn.rollback()
                flash(f"Erreur : {e}")
            finally:
                conn.close()
            return redirect(url_for('afficher_produits'))
    return render_template('ajouterproduit.html')

```

**@app.route('/ajouterproduit', methods=['GET', 'POST'])** : Route pour ajouter un produit, acceptant les requêtes GET et POST.

**ajouterproduit** : Fonction qui traite les données du formulaire. Si une requête POST est reçue, elle récupère les données, ajoute le produit à la base de données et redirige vers la page des produits. En cas d'erreur, elle affiche un message.

Les produits peuvent être modifiés et supprimés

Modification d'un produit

```

# Modifier un produit
@app.route('/produit/modifierproduit/<int:produit_id>', methods=['GET', 'POST'])
def modifierproduit(produit_id):
    conn = creer_connexion()
    if conn:
        cursor = conn.cursor()
        if request.method == 'POST':
            nom = request.form['nom']
            prix = request.form['prix']
            stock = request.form['stock']
            try:
                cursor.execute("UPDATE Produit SET nom=%s, prix=%s, stock=%s WHERE id=%s",
                                conn.commit()
                                flash("Produit mis à jour avec succès.")
            except Error as e:
                conn.rollback()
                flash(f"Erreur : {e}")
            finally:
                conn.close()
                return redirect(url_for('afficher_produits'))
        cursor.execute("SELECT * FROM Produit WHERE id = %s", (produit_id,))
        produit = cursor.fetchone()
        conn.close()
        return render_template('modifierproduit.html', produit=produit)

```

**@app.route('/produit/modifierproduit/int:produit\_id', methods=['GET', 'POST']) :**

Route pour modifier un produit en fonction de son ID.

**modifierproduit :** Fonction qui récupère les détails du produit à modifier. Si la méthode est POST, elle met à jour les données dans la base de données et affiche un message de succès. Si la méthode est GET, elle affiche le formulaire de modification avec les données existantes.

Suppression :

```

# Supprimer un produit
@app.route('/produit/supprimerproduit/<int:produit_id>')
def supprimerproduit(produit_id):
    conn = creer_connexion()
    if conn:
        cursor = conn.cursor()
        try:
            cursor.execute("DELETE FROM Produit WHERE id = %s", (produit_id,))
            conn.commit()
            flash("Produit supprimé avec succès.")
        except Error as e:
            conn.rollback()
            flash(f"Erreur : {e}")
        finally:
            conn.close()
    return redirect(url_for('afficher_produits'))

```

**@app.route('/produit/supprimerproduit/int:produit\_id')** : Route pour supprimer un produit en fonction de son ID.

**supprimerproduit** : Fonction qui supprime le produit correspondant dans la base de données et affiche un message de succès ou d'erreur.

Enregistrer une vente

```
def Enregistrer une vente
@app.route('/ajouterventes', methods=['GET', 'POST'])
def enregistrer_vente():
    if request.method == 'POST':
        produit_id = request.form['produit_id']
        quantite = int(request.form['quantite'])
        conn = creer_connexion()
        if conn:
            cursor = conn.cursor()
            try:
                cursor.execute("SELECT stock FROM Produit WHERE id = %s", (produit_id,))
                stock_actuel = cursor.fetchone()[0]
                if stock_actuel >= quantite:
                    cursor.execute("INSERT INTO Vente (produit_id, quantite) VALUES (%s, %s)"
                                   cursor.execute("UPDATE Produit SET stock = stock - %s WHERE id = %s", (q
                    conn.commit()
                    flash("Vente enregistrée avec succès et stock mis à jour.")
                else:
                    flash("Stock insuffisant pour cette vente.")
            except Error as e:
                conn.rollback()
                flash(f"Erreur : {e}")
            finally:
                conn.close()
            return redirect(url_for('afficher_produits'))
        return render_template('ajouterventes.html')
```

**@app.route('/ajouterventes', methods=['GET', 'POST'])** : Route pour enregistrer une vente.

**enregistrer\_vente** : Fonction qui traite les données de la vente, vérifie le stock disponible, enregistre la vente dans la base de données, et met à jour le stock du produit. En cas de stock insuffisant ou d'erreur, elle affiche un message approprié.

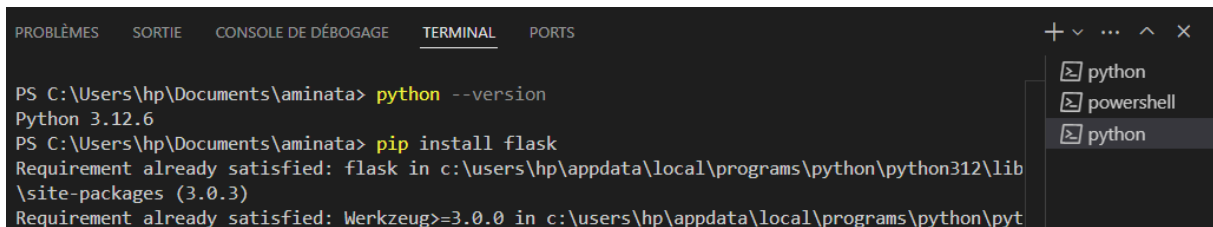
Exécution de l'Application

```
if __name__ == '__main__':
    app.run(debug=True)
```

**app.run(debug=True)** : Démarre le serveur Flask en mode débogage, permettant de visualiser les erreurs et d'actualiser automatiquement l'application lors de modifications de code.

## TESTS ET DEBOGAGE

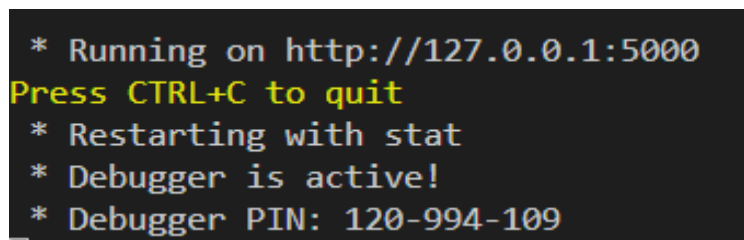
Nous allons d'abord lancer l'application flask sur le terminal au niveau du logiciel utilisé Visual Studio Code.



```
PROBLÈMES  SORTIE  CONSOLE DE DÉBOGAGE  TERMINAL  PORTS

PS C:\Users\hp\Documents\aminata> python --version
Python 3.12.6
PS C:\Users\hp\Documents\aminata> pip install flask
Requirement already satisfied: flask in c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (3.0.3)
Requirement already satisfied: Werkzeug>=3.0.0 in c:\users\hp\appdata\local\programs\python\pyt
```

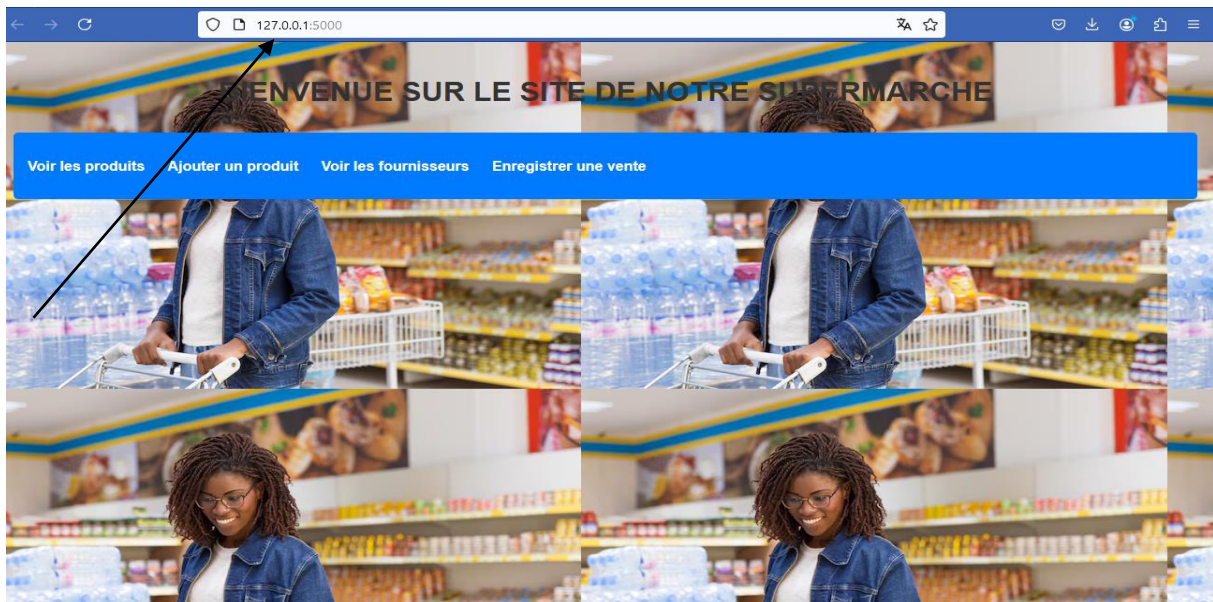
Après exécution, le terminal nous affiche que l'application flask est lancée avec succès et nous fournit le code d'accès au site.



```
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 120-994-109
```

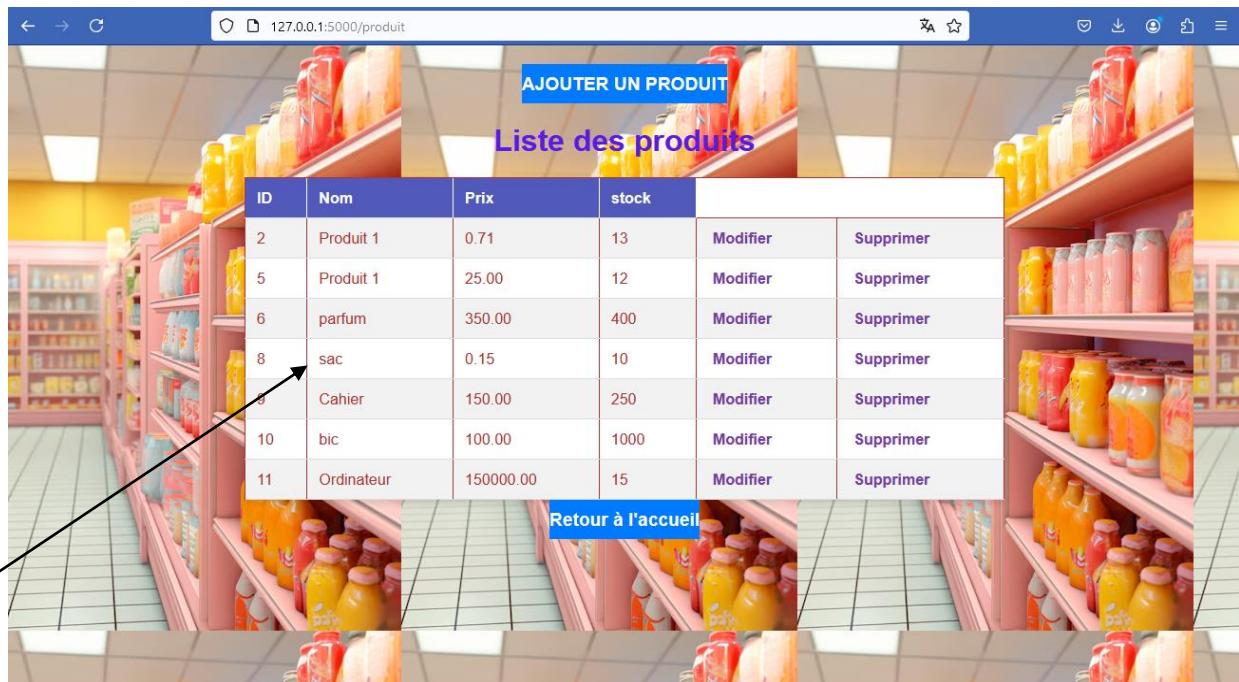
Nous pouvons maintenant ouvrir le navigateur web et accéder à l'adresse suivante :

<http://127.0.0.1:5000>. Cela devrait vous montrer la page de bienvenue de notre application, selon notre code.



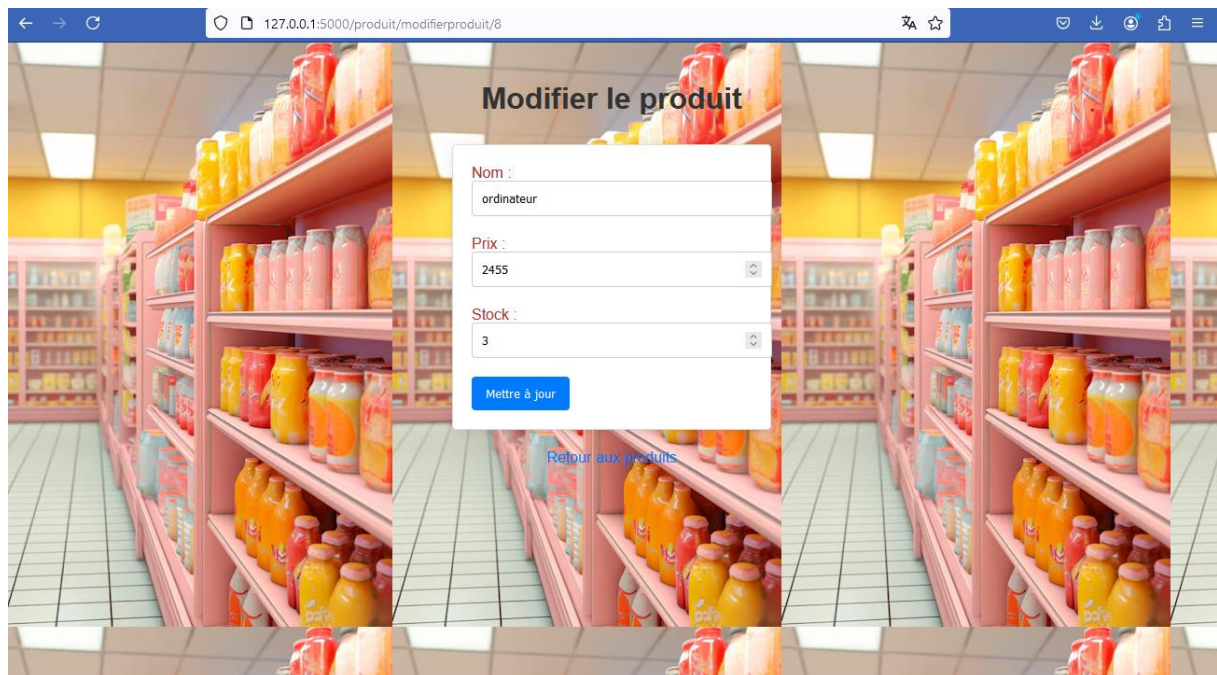
Une fois dans le site web, il est maintenant possible d'accéder aux fichiers produit, fournisseurs et ventes.





Il est possible de modifier et de supprimer un produit.

Nous allons essayer de modifier toutes les informations concernant le produit numero 8.



Resultat :



AJOUTER UN PRODUIT

Liste des produits

ID	Nom	Prix	stock		
2	Produit 1	0.71	13	Modifier	Supprimer
5	Produit 1	25.00	12	Modifier	Supprimer
6	parfum	350.00	400	Modifier	Supprimer
8	ordinateur	2455.00	3	Modifier	Supprimer
9	Cahier	150.00	250	Modifier	Supprimer
10	bic	100.00	1000	Modifier	Supprimer
11	Ordinateur	150000.00	15	Modifier	Supprimer

Retour à l'accueil

Essayons maintenant de supprimer le produit numero 5 :

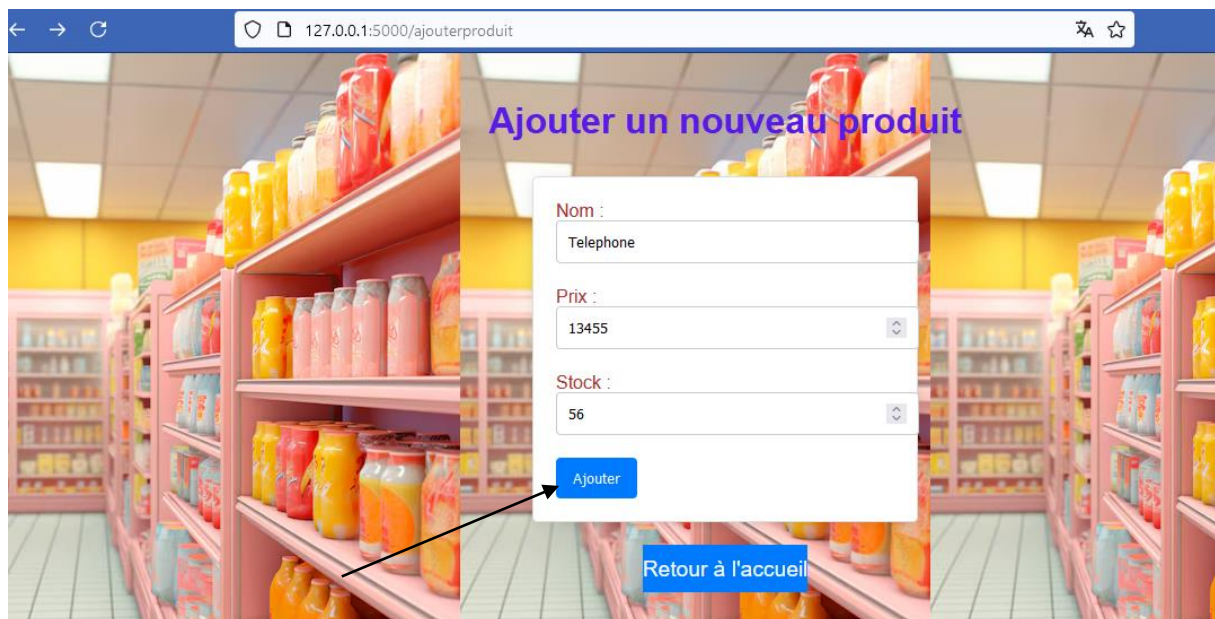
AJOUTER UN PRODUIT

Liste des produits

ID	Nom	Prix	stock		
2	Produit 1	0.71	13	Modifier	Supprimer
6	parfum	350.00	400	Modifier	Supprimer
8	ordinateur	2455.00	3	Modifier	Supprimer
9	Cahier	150.00	250	Modifier	Supprimer
10	bic	100.00	1000	Modifier	Supprimer
11	Ordinateur	150000.00	15	Modifier	Supprimer

Retour à l'accueil

Cliquer sur « ajouter un produit » sur la barre de navigation dans « bienvenue » nous mènera directement à la page d'ajout de produit. Ainsi nous pourrons ajouter un produit en saisissant son nom, son prix et le stock disponible.



Resultat :

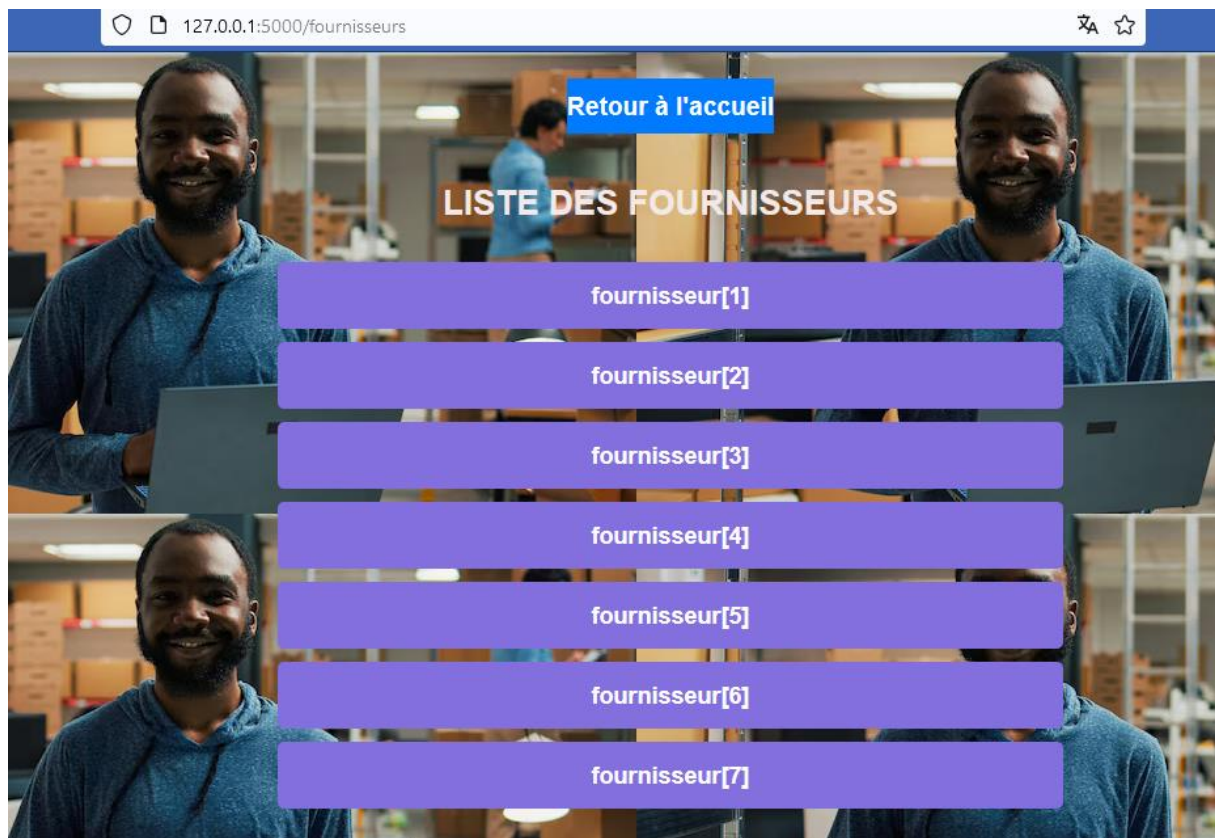
AJOUTER UN PRODUIT

Liste des produits

ID	Nom	Prix	stock		
2	Produit 1	0.71	13	Modifier	Supprimer
6	parfum	350.00	400	Modifier	Supprimer
8	ordinateur	2455.00	3	Modifier	Supprimer
9	Cahier	150.00	250	Modifier	Supprimer
10	bic	100.00	1000	Modifier	Supprimer
11	Ordinateur	150000.00	15	Modifier	Supprimer
12	Telephone	13455.00	56	Modifier	Supprimer
13	Telephone	13455.00	56	Modifier	Supprimer

Retour à l'accueil

Voici la liste des fournisseurs



L'enregistrement de la vente est directement lié à la liste des produits. A chaque vente le site demande d'enregistrer la quantité et l'ID pour mettre à jour les produits et stocks disponibles.

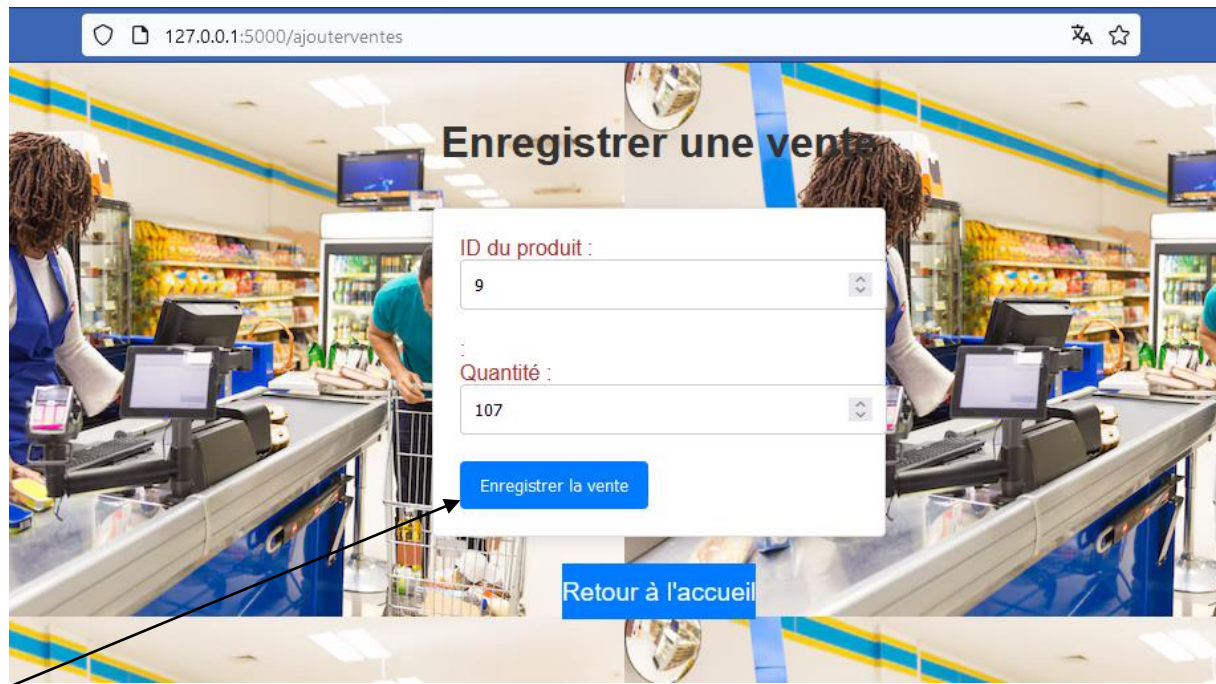
Démonstration :

L'exemple concernera l'ID numero 9 ;

9	Cahier	150.00	250	Modifier	Supprimer
---	--------	--------	-----	----------	-----------

Nous constatons avoir vendu 107 cahiers qui doivent etre enregistrées dans la vente.





Resultat :

AJOUTER UN PRODUIT

Liste des produits

ID	Nom	Prix	stock		
2	Produit 1	0.71	0	Modifier	Supprimer
6	parfum	350.00	400	Modifier	Supprimer
8	ordinateur	2455.00	3	Modifier	Supprimer
9	Cahier	150.00	143	Modifier	Supprimer
10	bic	100.00	1000	Modifier	Supprimer
11	Ordinateur	150000.00	15	Modifier	Supprimer
12	Telephone	13455.00	22	Modifier	Supprimer
13	Telephone	13455.00	56	Modifier	Supprimer

Retour à l'accueil

Après enregistrement de la vente, il nous reste à présent 143 cahiers.

## **CONCLUSION**

En conclusion, ce projet de développement d'une application de gestion pour un supermarché a permis de concevoir une solution efficace et intuitive pour la gestion des produits, des fournisseurs et des ventes. En suivant le modèle de conception MVC (Modèle-Vue-Contrôleur), nous avons structuré notre application de manière à séparer clairement la logique métier, la présentation et le contrôle des données. Cette architecture a non seulement facilité le développement et la maintenance du code, mais a également permis d'améliorer l'expérience utilisateur.

La conception de la base de données a été un aspect clé de notre application, permettant de stocker et de gérer efficacement les informations sur les produits et les transactions. Grâce à l'intégration de MySQL, nous avons pu garantir la persistance et l'intégrité des données tout en offrant des performances optimales.