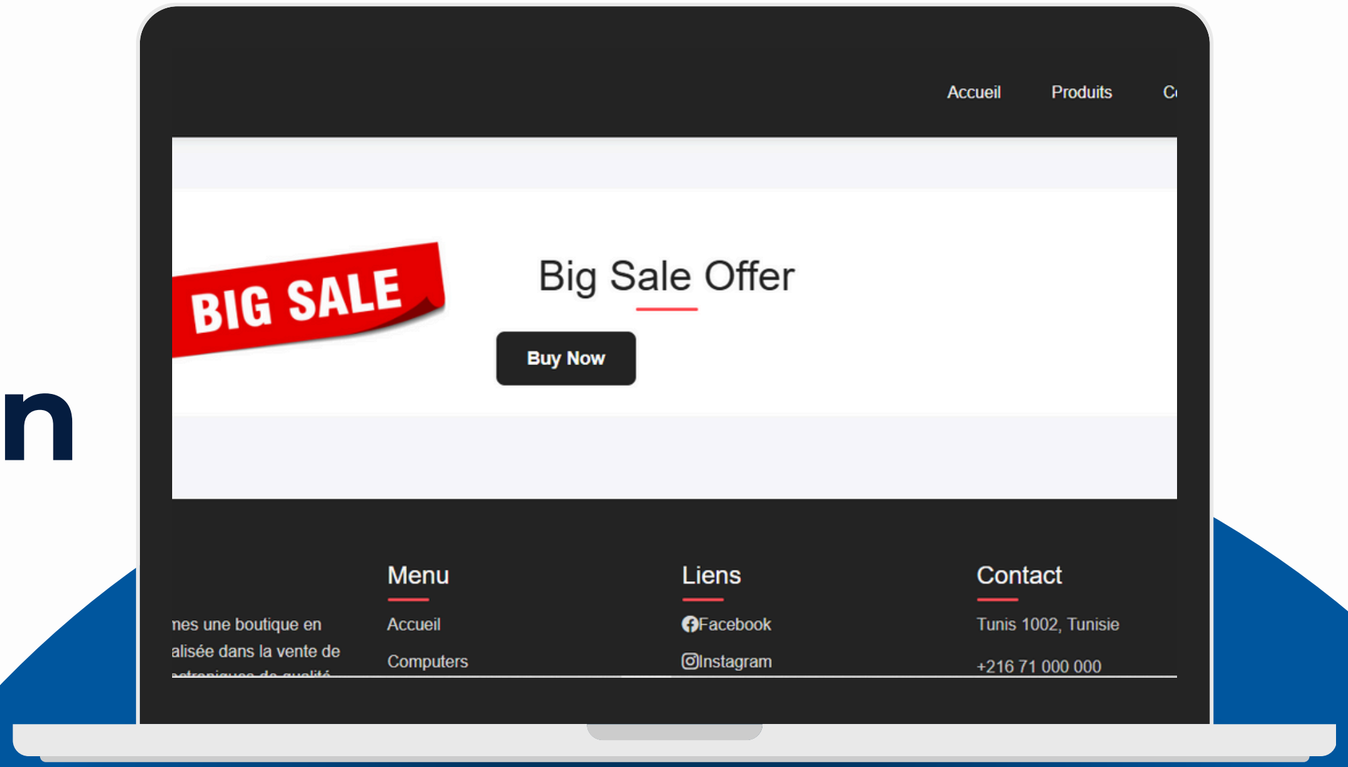
Institut supérieur de gestion
de Tunis

Projet Tuteuré Système de Recommandation



Liste de Binôme :
Sghaier Emna 2BI1
Ben Yamna Yasmine 2BI1

Introduction

Ce projet vise à développer un système de recommandation hybride pour un site e-commerce, combinant filtrage basé sur le contenu et filtrage collaboratif. L'objectif est de proposer des suggestions personnalisées aux utilisateurs en analysant leurs préférences et les tendances communautaires. Une interface simple permettra d'afficher les résultats de manière claire et efficace.

Moncode.py

Ce code Python met en place un système de recommandation de produits basé sur la similarité des descriptions textuelles dans une base de données MySQL. Voici une explication détaillée étape par étape :

Importation des bibliothèques nécessaires :

Le script utilise :

- nltk : traitement de texte (tokenisation, stopwords, stemming).
- scikit-learn : TF-IDF vectorisation.
- numpy : calculs numériques.
- scipy : distance euclidienne et corrélation de Pearson.
- mysql.connector : connexion à la base de données MySQL.

```python
import mysql.connector
import nltk
from nltk.stem.snowball import FrenchStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from numpy import dot
from numpy.linalg import norm
import numpy
from nltk.corpus import stopwords
from scipy.spatial.distance import euclidean
from scipy.stats import pearsonr

# (Optionnel) Téléchargement des ressources NLTK
nltk.download('punkt')
nltk.download('stopwords')
```

Étapes principales :

1. **Connexion MySQL :** Le script se connecte à une base MySQL nommée sysrec, dans laquelle se trouve une table appelée produit. Cette table contient les produits avec leur description (et plus tard, les recommandations)
2. **Prétraitement des descriptions :**

Récupération des descriptions depuis la BDD.
Nettoyage du texte :
Tokenisation (découpage en mots),
Passage en minuscules,
Suppression des stopwords français,
Stemming (réduction des mots à leur racine).

```python
# (Optionnel) Téléchargement des ressources NLTK
nltk.download('punkt')
nltk.download('stopwords')

# Connexion à MySQL
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="sysrec"
)
cursor = conn.cursor()
```

Moncode.py

3.Transformation TF-IDF : Les descriptions traitées sont transformées en vecteurs numériques via TF-IDF :
Plus un mot est important dans une description (et rare dans les autres), plus son poids est élevé.
Cela permet de comparer les descriptions de manière mathématique.

4. Implémentation de plusieurs méthodes de similarité:

- Cosinus : méthode principale utilisée pour les recommandations finales.
- Jaccard : mesure le taux d'intersection entre deux ensembles de mots.
- Euclidienne : mesure la distance géométrique entre deux vecteurs.
- Pearson : mesure la corrélation linéaire entre deux vecteurs.

Même si seule la similarité cosinus est utilisée pour générer les recommandations finales, les autres méthodes sont présentes dans le code.

5. Recommandation :
Pour chaque produit :
Le script cherche les trois autres produits les plus similaires,
Il enregistre leurs identifiants dans les colonnes **top1**, **top2**, **top3** de la base.

```python
# Fonctions de similarité
def SimilariteCosinus(i, j, matriceTFIDF):
    vecteur_i = matriceTFIDF[i]
    vecteur_j = matriceTFIDF[j]
    if norm(vecteur_i) == 0 or norm(vecteur_j) == 0:
        return 0
    return dot(vecteur_i, vecteur_j) / (norm(vecteur_i) * norm(vecteur_j))

def SimilariteJaccard(i, j, matriceTFIDF):
    set_i = set(matriceTFIDF[i])
    set_j = set(matriceTFIDF[j])
    intersection = len(set_i.intersection(set_j))
    union = len(set_i.union(set_j))
    return intersection / union if union != 0 else 0
```

```python
def SimilariteEuclidienne(i, j, matriceTFIDF):
    return euclidean(matriceTFIDF[i], matriceTFIDF[j])

def SimilaritePearson(i, j, matriceTFIDF):
    corr, _ = pearsonr(matriceTFIDF[i], matriceTFIDF[j])
    return corr if not numpy.isnan(corr) else 0

# Récupération des produits
cursor.execute("SELECT * FROM produit")
Produits = cursor.fetchall()

dictProduits = {}
idProduits = []
StopList = set(stopwords.words('french'))
stemer = FrenchStemmer()

# Prétraitement des descriptions
for p in Produits:
    idPdt = int(p[0])
    idProduits.append(idPdt)
    Description = p[1]
    Mots = nltk.word_tokenize(Description)
    MotsStems = [stemer.stem(m.lower()) for m in Mots]
    ListFinalMots = [m for m in MotsStems if m not in StopList]
    dictProduits[idPdt] = ListFinalMots

# TF-IDF
descriptions = [' '.join(dictProduits[idPdt]) for idPdt in idProduits]
vectorizer = TfidfVectorizer()
matriceTFIDF = vectorizer.fit_transform(descriptions).toarray()

# Matrices de similarité (Cosinus ici)
matriceSimilariteCosinus = numpy.zeros((len(idProduits), len(idProduits)))
for i in range(len(idProduits)):
    for j in range(len(idProduits)):
        matriceSimilariteCosinus[i][j] = SimilariteCosinus(i, j, matriceTFIDF)
```

```python
# Calcul et insertion des recommandations
N = 3
for i in range(len(idProduits)):
    idProduitCible = idProduits[i]
    similarites = matriceSimilariteCosinus[i]
    produits_similaires = sorted(
        [(j, similarites[j]) for j in range(len(similarites)) if j != i],
        key=lambda x: x[1], reverse=True
    )
    recommandations = [idProduits[j] for j, _ in produits_similaires[:N]]
    cursor.execute("""
        UPDATE produit
        SET top1 = %s, top2 = %s, top3 = %s
        WHERE id_produit = %s
    """, (
        recommandations[0] if len(recommandations) > 0 else None,
        recommandations[1] if len(recommandations) > 1 else None,
        recommandations[2] if len(recommandations) > 2 else None,
        idProduitCible
    ))

conn.commit()

# Affichage des recommandations
cursor.execute("SELECT id_produit, top1, top2, top3 FROM produit")
for row in cursor.fetchall():
    print(f"Produit ID {row[0]}: Top 1 - {row[1]}, Top 2 - {row[2]}, Top 3 - {row[3]}")

# Fermeture
conn.close()
```

Exécution :

```
Liste des termes TF-IDF :
['108' '12' '128' '13' '15' '16' '17' '18' '20x' '24'
'256' '3060' '32'
 '5000mah' '512' '55mm' '5g' 'a15' 'amoled' 'android'
'batter' 'bionic'
 'bord' 'bureau' 'bureaut' 'camer' 'capteur' 'cart'
'chanti' 'charg'
 'compact' 'coqu' 'cor' 'doubl' 'détect' 'en' 'entré'
'full' 'gaming'
 'gamm' 'garant' 'go' 'graphiqu' 'graphism' 'haut' 'hd'
'i7' 'idéal'
 'incass' 'integr' 'intel' 'ip' 'iphon' 'main' 'min'
'mois' 'montag'
 'mouv' 'mp' 'nocturn' 'numer' 'object' 'optiqu'
'ordin' 'orient' 'parf'
 'pc' 'portabl' 'pouc' 'principal' 'pro' 'processeur'
'puc' 'ram' 'rapid'
 'recondition' 'reflex' 'renforc' 'rtx' 'résolu' 'ser'
'sim' 'smartphon'
 'ssd' 'stabilis' 'stockag' 'surveil' 'tableau' 'tout'
'tripl' 'téléphon'
 'un' 'usag' 'vidéo' 'vision' 'vlogger' 'voitur' 'wi'
'zoom' 'écran'
 'étudi']
```

```
Matrice TF-IDF :

Produit 17 :
  15 : 0.2798
  16 : 0.2798
  512 : 0.2798
  cor : 0.2798
  go : 0.4066
  i7 : 0.2798
  intel : 0.2798
  ordin : 0.2219
  portabl : 0.2219
  pouc : 0.2219
  processeur : 0.2798
  ram : 0.2460
  ssd : 0.2798

Produit 18 :
  3060 : 0.3346
  32 : 0.3346
  bureau : 0.3346
  cart : 0.3346
  gaming : 0.3346
  go : 0.2431
  graphiqu : 0.3346
  ordin : 0.2654
  ram : 0.2942
  rtx : 0.3346
```

```
Produit ID 17: Top 1 - 18, Top 2 - 33, Top 3 - 27
Produit ID 18: Top 1 - 17, Top 2 - 33, Top 3 - 27
Produit ID 19: Top 1 - 30, Top 2 - 20, Top 3 - 33
Produit ID 20: Top 1 - 30, Top 2 - 19, Top 3 - 21
Produit ID 21: Top 1 - 22, Top 2 - 20, Top 3 - 19
Produit ID 22: Top 1 - 32, Top 2 - 25, Top 3 - 21
Produit ID 23: Top 1 - 32, Top 2 - 25, Top 3 - 30
Produit ID 24: Top 1 - 35, Top 2 - 32, Top 3 - 26
Produit ID 25: Top 1 - 23, Top 2 - 22, Top 3 - 17
Produit ID 26: Top 1 - 35, Top 2 - 32, Top 3 - 24
Produit ID 27: Top 1 - 17, Top 2 - 30, Top 3 - 23
Produit ID 28: Top 1 - 17, Top 2 - 27, Top 3 - 18
Produit ID 29: Top 1 - 31, Top 2 - 36, Top 3 - 32
Produit ID 30: Top 1 - 20, Top 2 - 19, Top 3 - 27
Produit ID 31: Top 1 - 36, Top 2 - 32, Top 3 - 29
Produit ID 32: Top 1 - 23, Top 2 - 22, Top 3 - 35
Produit ID 33: Top 1 - 17, Top 2 - 19, Top 3 - 21
Produit ID 34: Top 1 - 31, Top 2 - 36, Top 3 - 32
Produit ID 35: Top 1 - 32, Top 2 - 24, Top 3 - 26
Produit ID 36: Top 1 - 31, Top 2 - 32, Top 3 - 29
```

Filtrage.py :

Ce code recommande des produits à un utilisateur en se basant sur les notes données par d'autres utilisateurs similaires. Il utilise une matrice de similarité cosinus pour estimer dans quelle mesure deux utilisateurs ont des goûts similaires.

```python
import pymysql.cursors
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

class CollaborativeFiltering:
    def __init__(self):
        self.conn = None
        self.users = []
        self.products = []
        self.ratings_matrix = None
        self.user_index = {}
        self.product_index = {}

    def connect_db(self):
        try:
            self.conn = pymysql.connect(
                host="localhost",
                user="root",
                password="",
                database="sysrec",
                autocommit=False
            )
            return True
        except pymysql.MySQLError as err:
            print(f"Erreur MySQL: {err}")
            return False
```

1. Connexion à la base de données

- Le code se connecte à la base sysrec (même que dans le code précédent).
- Il récupère les identifiants des utilisateurs (user) et des produits (produit).

2. Chargement des données

- Création d'une matrice de notes de taille (nombre d'utilisateurs × nombre de produits).
- Chaque case contient la note donnée par un utilisateur à un produit (ou 0 si aucune note).
- Les notes proviennent de la table notes.

```python
def load_data(self, conn):
    try:
        cursor = conn.cursor(pymysql.cursors.DictCursor)
        cursor.execute("SELECT id_user FROM user ORDER BY id_user")
        self.users = [row['id_user'] for row in cursor.fetchall()]
        self.user_index = {user_id: idx for idx, user_id in enumerate(self.users)}

        cursor.execute("SELECT id_produit FROM produit ORDER BY id_produit")
        self.products = [row['id_produit'] for row in cursor.fetchall()]
        self.product_index = {prod_id: idx for idx, prod_id in enumerate(self.products)}

        self.ratings_matrix = np.zeros((len(self.users), len(self.products)))

        cursor.execute("SELECT id_user, id_produit, note FROM notes")
        for row in cursor.fetchall():
            try:
                user_idx = self.user_index[row['id_user']]
                prod_idx = self.product_index[row['id_produit']]
                if row['note'] is None or row['note'] == 0:
                    self.ratings_matrix[user_idx][prod_idx] = 0
                else:
                    self.ratings_matrix[user_idx][prod_idx] = row['note']
            except KeyError:
                continue

        return True
    except Exception as e:
        print(f"Erreur chargement données: {e}")
        return False
```

3.Calcul de la similarité entre utilisateurs

- On calcule une matrice de similarité cosinus entre les utilisateurs :
  - On centre les notes (on soustrait la moyenne de chaque utilisateur).
  - Puis on applique la fonction cosine_similarity() de Scikit-learn.

4. Génération des recommandations pour un utilisateur donné

- Pour chaque produit que l'utilisateur n'a pas noté :
  - Le système cherche les autres utilisateurs similaires qui ont noté ce produit.
  - Il calcule une note prédite en faisant une moyenne pondérée selon la similarité.
- Seuls les produits avec une note prédite suffisamment élevée sont conservés.

Filtrage.py :

5. Fonction recommander_produits(user_id)

- C’est la fonction principale à appeler : elle retourne les ID des produits recommandés pour un utilisateur donné.
- Elle appelle les fonctions précédentes (chargement, similarité, prédictions).

- Ce système repose sur l’analyse des comportements des utilisateurs (collaboratif), contrairement à l’autre script qui s’appuyait sur le contenu des produits (descriptions textuelles). Dans le premier code, plusieurs métriques étaient disponibles (cosinus, Jaccard, Euclidienne, Pearson), mais seule la similarité cosinus était appliquée au final pour les recommandations.

```python
def calculate_similarities(self):
    try:
        # Remplace les 0 par NaN pour ignorer dans le calcul de la moyenne
        masked_ratings = np.where(self.ratings_matrix > 0, self.ratings_matrix, np.nan)
        user_means = np.nanmean(masked_ratings, axis=1, keepdims=True)
        norm_ratings = self.ratings_matrix - np.nan_to_num(user_means)
        return cosine_similarity(norm_ratings)
    except Exception as e:
        print(f"Erreur calcul similarités: {e}")
        return None
```

```python
def get_recommendations(self, user_id, similarity_matrix, k=3):
    if user_id not in self.user_index:
        return []

    user_idx = self.user_index[user_id]
    predictions = []

    for prod_idx, product_id in enumerate(self.products):
        if self.ratings_matrix[user_idx][prod_idx] == 0:  # Produit non noté
            similar_users = []
            for other_user_idx in range(len(self.users)):
                if other_user_idx != user_idx and self.ratings_matrix[other_user_idx][prod_idx] > 0:
                    similarity = similarity_matrix[user_idx][other_user_idx]
                    rating = self.ratings_matrix[other_user_idx][prod_idx]
                    similar_users.append((similarity, rating))

            similar_users.sort(reverse=True)
            similar_users = similar_users[:k]

            if similar_users:
                sum_sim = sum(sim for sim, _ in similar_users)
                if sum_sim > 0:
                    pred_rating = sum(sim * rating for sim, rating in similar_users) / sum_sim
                    print(f"Produit {product_id} - Prédiction: {pred_rating} (utilisé par {len(similar_users)} utilisateurs similaires)
                    if pred_rating >= 0:
                        predictions.append((product_id, pred_rating))
                else:
                    print(f"Produit {product_id} - Similarité nulle ou négative")
            else:
                print(f"Produit {product_id} - Aucun utilisateur similaire avec une note")
    return sorted(predictions, key=lambda x: x[1], reverse=True)
```

app.py :

Application Flask de Recommandation de Produits :

Introduction:

Cette application web est développée avec Flask, un framework Python léger, et utilise une base de données MySQL pour gérer les informations des utilisateurs, produits, évaluations et achats. Elle propose une fonctionnalité de recommandation de produits pour les utilisateurs en fonction de leurs évaluations, et permet également la gestion des évaluations et des achats de produits.

1. Configuration Flask :

Explication :

- Flask est configuré pour gérer les fichiers statiques et les templates avec un rechargement automatique.
- La clé secrète (SECRET_KEY) est utilisée pour sécuriser les sessions et autres aspects liés à la sécurité dans l'application

```python
from flask import Flask, render_template, redirect, url_for, request, flash, session, jsonify

import pymysql
from functools import wraps
from werkzeug.security import check_password_hash
from filtrage import recommander_produits

app = Flask(__name__, static_folder='static', static_url_path='/static')
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0
app.config['TEMPLATES_AUTO_RELOAD'] = True
app.config['SECRET_KEY'] = 'your_secret_key'

DB_CONFIG = {
    'host': 'localhost',
    'user': 'root',
    'password': '',
    'db': 'sysrec',
    'charset': 'utf8mb4',
    'cursorclass': pymysql.cursors.DictCursor
}
```

2. Connexion à la base de données MySQL

La connexion à la base de données MySQL est gérée par une fonction décorée db_connection_handler.

- Cette fonction décoratrice gère la connexion à la base de données, permettant de l'utiliser dans différentes routes tout en assurant la fermeture correcte de la connexion après l'exécution.
- Les erreurs MySQL sont capturées et affichées de manière appropriée.

```python
def db_connection_handler(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        conn = None
        try:
            conn = pymysql.connect(**DB_CONFIG)
            kwargs['conn'] = conn
            return f(*args, **kwargs)
        except pymysql.Error as e:
            print(f"Erreur MySQL: {e}")
            return render_template('error.html', error="Erreur de base de données"), 500
        finally:
            if conn:
                conn.close()
    return wrapper

@app.route('/')
def index():
    return render_template('index.html')
```

app.py :

3. Gestion de l'authentification (Login) :

- Cette route permet à un utilisateur de se connecter en vérifiant ses informations dans la base de données.
- Les informations de connexion sont comparées avec celles enregistrées dans la table user, et si les informations sont valides, l'utilisateur est redirigé vers la page d'accueil avec une session active.

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username'].strip()
        password = request.form['password'].strip()

        conn = pymysql.connect(**DB_CONFIG)
        try:
            with conn.cursor() as cursor:
                cursor.execute('SELECT * FROM user WHERE login = %s', (username,))
                user = cursor.fetchone()

                if user and user['password'].strip() == password.strip():
                    session['user_id'] = user['id_user']
                    session['username'] = user['login']
                    flash('Connexion réussie!', 'success')
                    return redirect(url_for('utilisateur'))
                else:
                    flash('Nom d\'utilisateur ou mot de passe incorrect', 'danger')
                    return redirect(url_for('login'))
        except Exception as e:
            print(f"Erreur : {e}")
            flash('Erreur lors de la connexion à la base de données.', 'danger')
            return redirect(url_for('login'))
        finally:
            conn.close()

    return render_template('login.html')
```

4. Page de l'utilisateur avec recommandations :

- Si l'utilisateur est connecté, cette route affiche ses produits notés, ainsi que les produits recommandés, en appelant la fonction recommander_produits, qui utilise des données sur les évaluations de l'utilisateur pour déterminer les produits similaires.
- Elle charge également tous les produits disponibles dans la base de données pour permettre à l'utilisateur d'évaluer ou d'acheter d'autres produits.

```python
                placeholders = ','.join(['%s'] * len(ids_recommandes))
                cursor.execute(f'''
                    SELECT p.*, n.note
                    FROM produit p
                    LEFT JOIN notes n ON n.id_produit = p.id_produit AND n.id_user = %s
                    WHERE p.id_produit IN ({placeholders})
                ''', (user_id, *ids_recommandes))
                produits_recommandes = cursor.fetchall()

            # Produits disponibles avec leurs notes
            cursor.execute('''
                SELECT p.*, n.note
                FROM produit p
                LEFT JOIN notes n ON n.id_produit = p.id_produit AND n.id_user = %s
            ''', (user_id,))
            produits_disponibles = cursor.fetchall()

        return render_template(
            'utilisateur.html',
            produits_notes=produits_notes,
            produits_recommandes=produits_recommandes,
            produits_disponibles=produits_disponibles
        )

    except Exception as e:
        print(f"Erreur: {e}")
        flash('Une erreur est survenue.', 'danger')
        return redirect(url_for('index'))
```

```python
/utilisateur')
on_handler
eur(conn):
_id' not in session:
('Vous devez être connecté pour accéder à cette page.', 'danger')
rn redirect(url_for('login'))


_id = session['user_id']
 conn.cursor() as cursor:
# Produits déjà notés avec leurs notes
cursor.execute('''
    SELECT p.*, n.note
    FROM produit p
    JOIN notes n ON n.id_produit = p.id_produit
    WHERE n.id_user = %s AND n.note IS NOT NULL
''', (user_id,))
produits_notes = cursor.fetchall()

# Produits recommandés
ids_recommandes = recommander_produits(user_id, conn)
print("Produits recommandés IDs:", ids_recommandes)

produits_recommandes = []
if ids_recommandes:
    placeholders = ','.join(['%s'] * len(ids_recommandes))
    cursor.execute(f'''
        SELECT p.*, n.note
        FROM produit p
        LEFT JOIN notes n ON n.id_produit = p.id_produit AND n.id_user = %s
        WHERE p.id_produit IN ({placeholders})
    ''', (user_id, *ids_recommandes))
    produits_recommandes = cursor.fetchall()
```

app.py :

```python
@app.route('/logout')
def logout():
    session.clear()
    flash('Vous êtes déconnecté avec succès.', 'success')
    return redirect(url_for('login'))
```

5. Déconnexion (Logout) :
Cette route permet à l'utilisateur de se déconnecter en supprimant la session en cours et en affichant un message de succès.

6. Soumission des évaluations de produits:

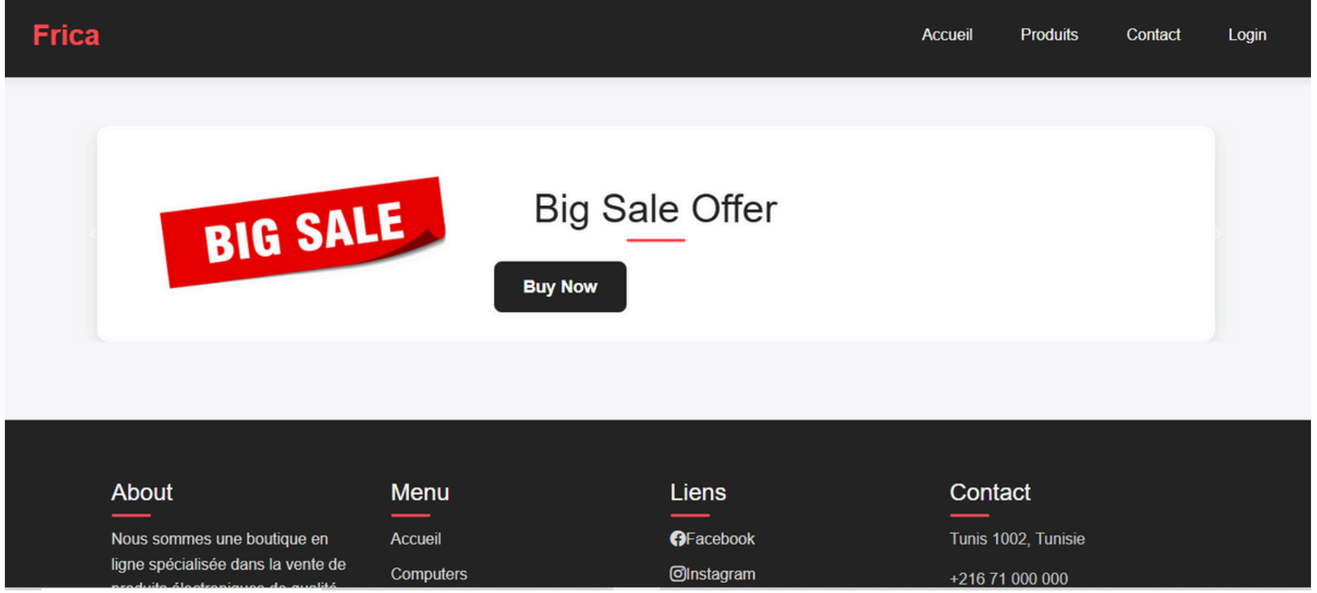
- Cette route permet à un utilisateur de soumettre une note pour un produit.
- La note est validée, puis soit insérée dans la base de données si elle n'existe pas déjà, soit mise à jour si elle existe déjà.

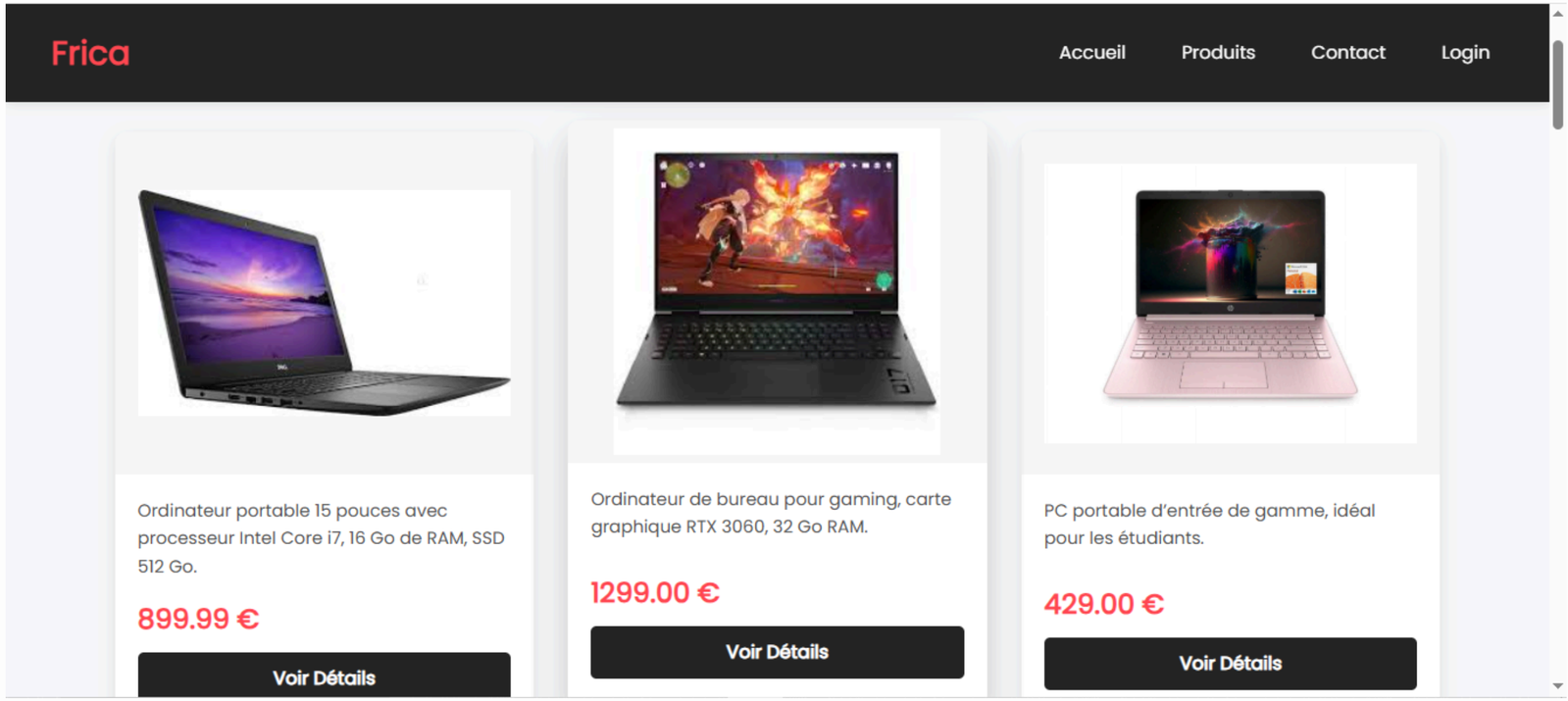
```python
@app.route('/submit-rating', methods=['POST'])
@db_connection_handler
def submit_rating(conn):
    if 'user_id' not in session:
        return jsonify({'success': False, 'message': 'Utilisateur non connecté'}), 401

    try:
        data = request.get_json()
        if not data:
            return jsonify({'success': False, 'message': 'Données manquantes'}), 400

        product_id = data.get('productId')
        rating = data.get('rating')
        user_id = session['user_id']

        if product_id is None or rating is None:
            return jsonify({'success': False, 'message': 'Produit ou note manquant'}), 400

        # Validation des données
        try:
            product_id = int(product_id)
            rating = int(rating)
        except ValueError:
            return jsonify({'success': False, 'message': 'Types invalides'}), 400

        if rating < 1 or rating > 5:
            return jsonify({'success': False, 'message': 'Note invalide (1 à 5 requis)'}), 400

        with conn.cursor() as cursor:
            # Vérification des clés étrangères
            cursor.execute("SELECT * FROM produit WHERE id_produit = %s", (product_id,))
            if not cursor.fetchone():
                return jsonify({'success': False, 'message': 'Produit introuvable'}), 400
```

7. Soumission des achats de produits :
Cette route permet à un utilisateur de soumettre un achat, enregistrant le produit et la quantité achetée dans la base de données.

```python
@app.route('/submit-purchase', methods=['POST'])
@db_connection_handler
def submit_purchase(conn):
    # Vérification si l'utilisateur est connecté
    if 'user_id' not in session:
        return jsonify({'success': False, 'message': 'Utilisateur non connecté'}), 401

    user_id = session['user_id']  # Récupération de l'ID utilisateur depuis la session
    product_id = request.json.get('productId')  # Récupération de l'ID du produit depuis la requête
    quantity = request.json.get('quantity', 1)  # Quantité (par défaut à 1)

    # Insertion dans la base de données
    try:
        with conn.cursor() as cursor:
            cursor.execute('''
                INSERT INTO achats (id_user, id_produit, quantite)
                VALUES (%s, %s, %s)
            ''', (user_id, product_id, quantity))
            conn.commit()

        return jsonify({'success': True, 'message': 'Achat effectué avec succès'}), 200
    except Exception as e:
        conn.rollback()
        return jsonify({'success': False, 'message': str(e)}), 500


if __name__ == '__main__':
    app.run(debug=False, port=5001, use_reloader=False)
```

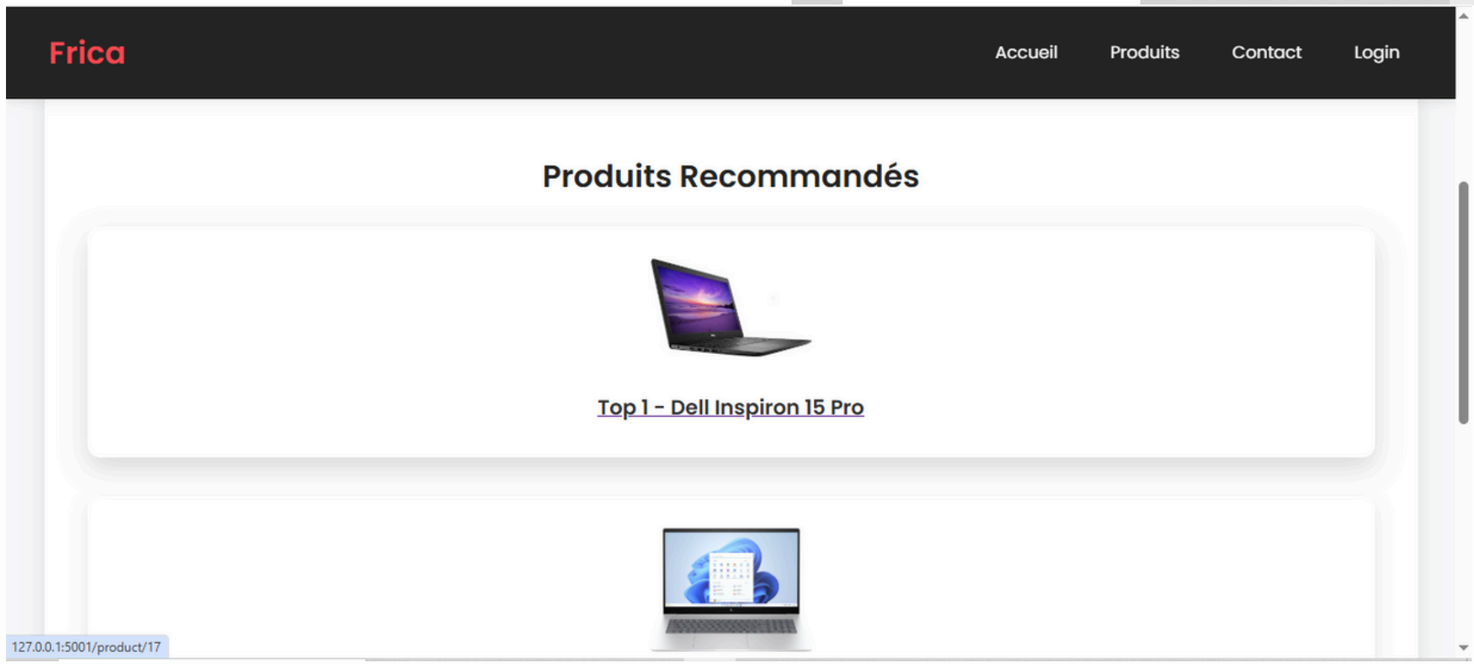
Le site WEB :


Frica
Accueil
Produits
Contact
Login
BIG SALE
Big Sale Offer
Buy Now
About
Nous sommes une boutique en ligne spécialisée dans la vente de
Menu
Accueil
Computers
Liens
Facebook
Instagram
Contact
Tunis 1002, Tunisie
+216 71 000 000

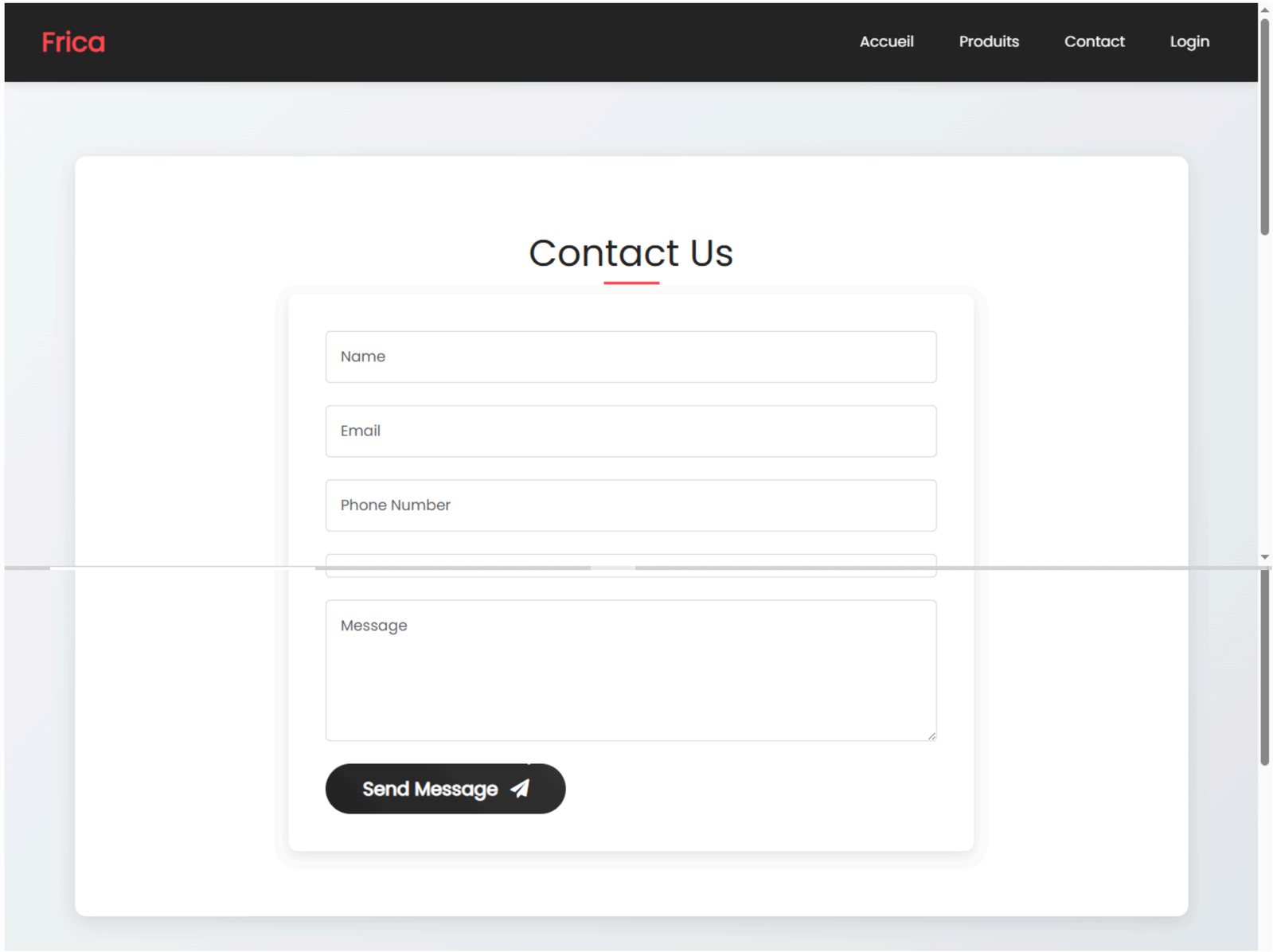

Frica
Accueil
Produits
Contact
Login
Ordinateur portable 15 pouces avec processeur Intel Core i7, 16 Go de RAM, SSD 512 Go.
899.99 €
Voir Détails
Ordinateur de bureau pour gaming, carte graphique RTX 3060, 32 Go RAM.
1299.00 €
Voir Détails
PC portable d'entrée de gamme, idéal pour les étudiants.
429.00 €
Voir Détails

Le site WEB :

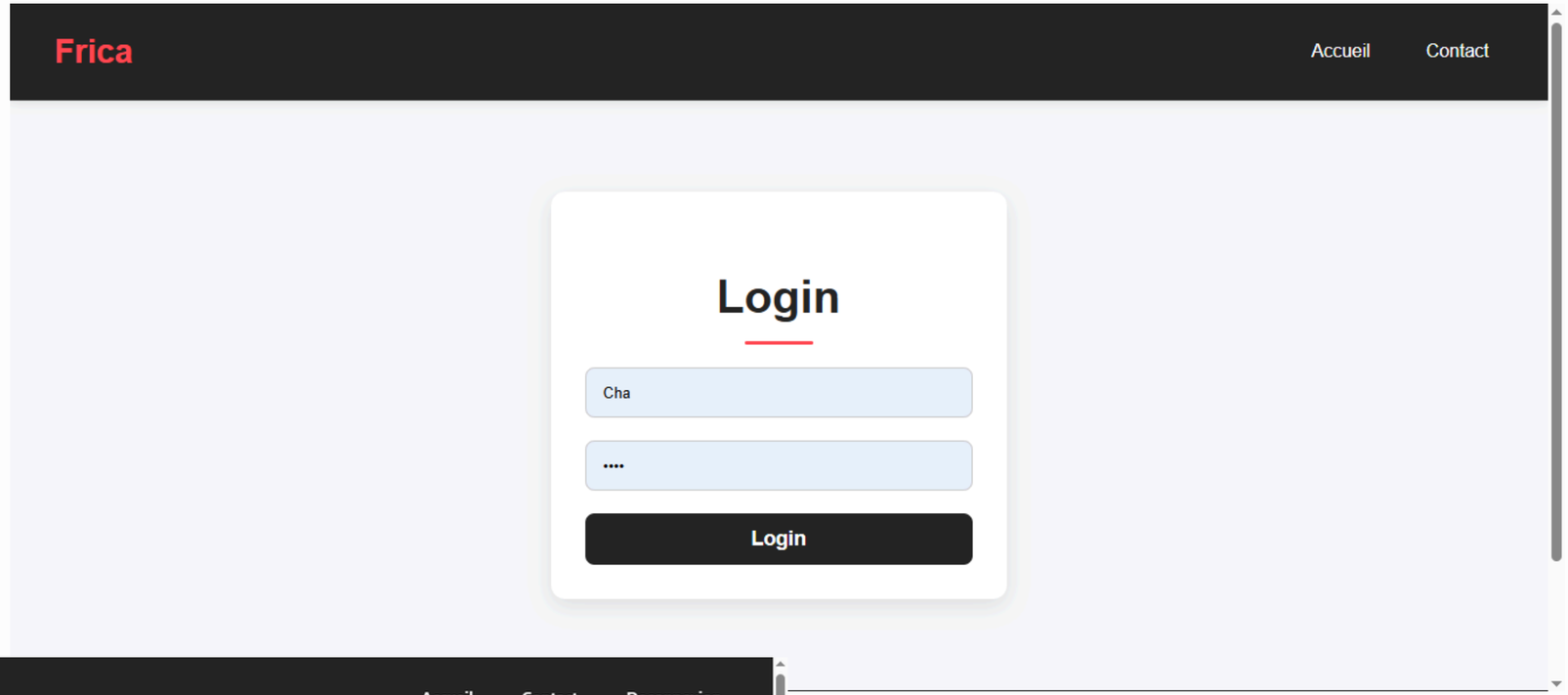

Frica
Accueil
Produits
Contact
Login
Détails du Produit
HP Omen Gamer X
Ordinateur de bureau pour gaming, carte graphique RTX 3060, 32 Go RAM.
1299.00 €
Produits Recommandés

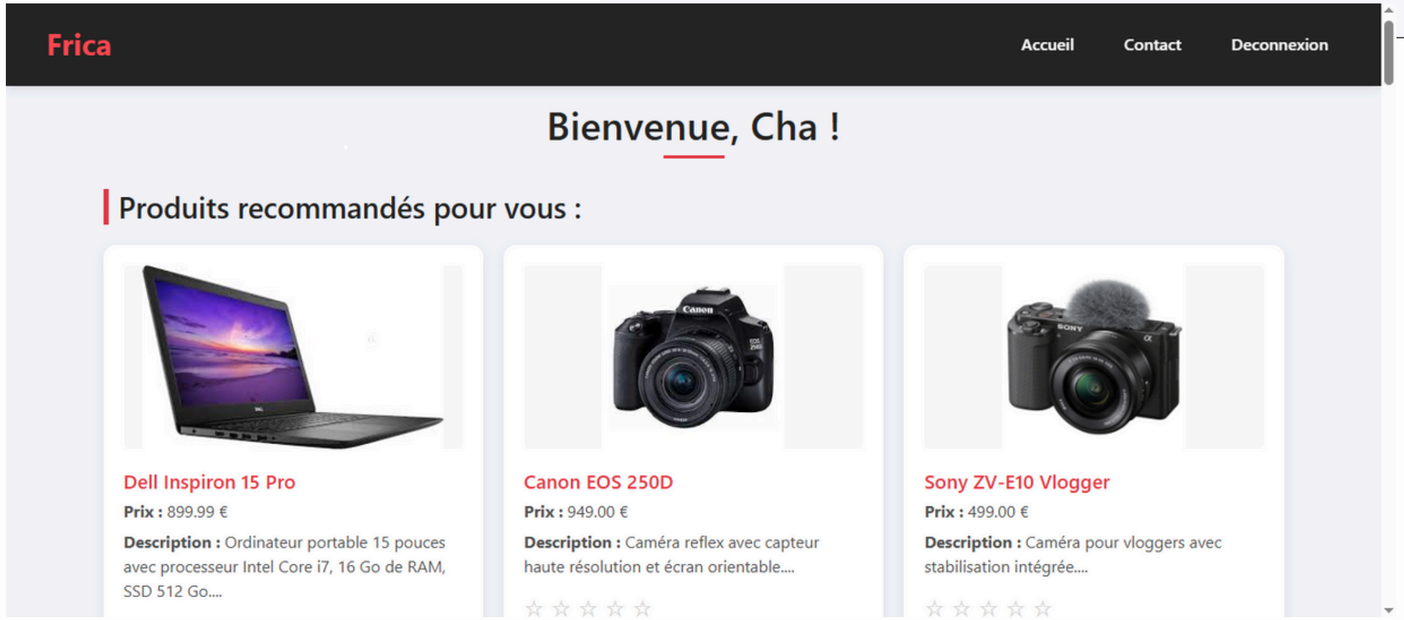

Frica
Accueil
Produits
Contact
Login
Produits Recommandés
Top 1 - Dell Inspiron 15 Pro
127.0.0.1:5001/product/17

Le site WEB :

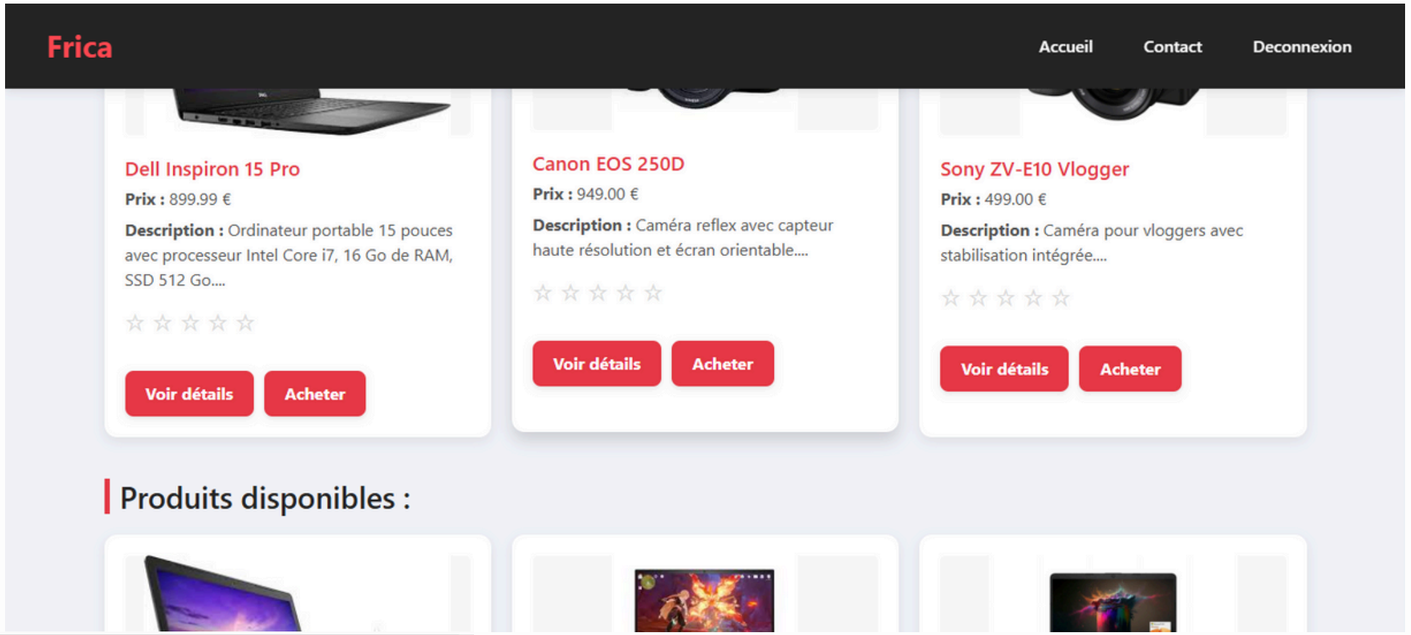

Frica
Accueil
Produits
Contact
Login
Contact Us
Name
Email
Phone Number
Message
Send Message

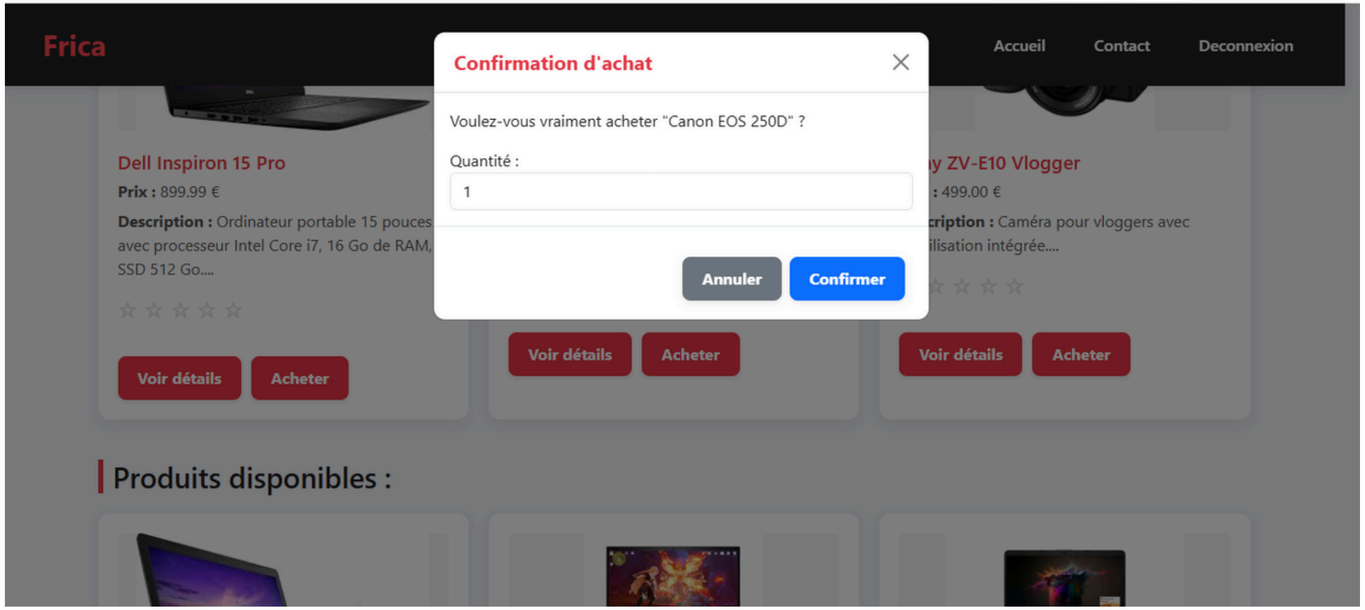
Le site WEB :


Frica
Accueil
Contact
Login
Cha
••••
Login


Frica
Accueil
Contact
Deconnexion
Bienvenue, Cha !
Produits recommandés pour vous :
Dell Inspiron 15 Pro
Prix : 899.99 €
Description : Ordinateur portable 15 pouces avec processeur Intel Core i7, 16 Go de RAM, SSD 512 Go....
Canon EOS 250D
Prix : 949.00 €
Description : Caméra reflex avec capteur haute résolution et écran orientable....
Sony ZV-E10 Vlogger
Prix : 499.00 €
Description : Caméra pour vloggers avec stabilisation intégrée....

Le site WEB :


Frica
Accueil
Contact
Deconnexion
Dell Inspiron 15 Pro
Prix : 899.99 €
Description : Ordinateur portable 15 pouces avec processeur Intel Core i7, 16 Go de RAM, SSD 512 Go....
Voir détails
Acheter
Canon EOS 250D
Prix : 949.00 €
Description : Caméra reflex avec capteur haute résolution et écran orientable....
Voir détails
Acheter
Sony ZV-E10 Vlogger
Prix : 499.00 €
Description : Caméra pour vloggers avec stabilisation intégrée....
Voir détails
Acheter
Produits disponibles :


Frica
Accueil
Contact
Deconnexion
Confirmation d'achat
Voulez-vous vraiment acheter "Canon EOS 250D" ?
Quantité :
1
Annuler
Confirmer
Dell Inspiron 15 Pro
Prix : 899.99 €
Description : Ordinateur portable 15 pouces avec processeur Intel Core i7, 16 Go de RAM, SSD 512 Go....
Voir détails
Acheter
Voir détails
Acheter
Voir détails
Acheter
Produits disponibles :

MERCI POUR VOTRE ATTENTION