Mercadona

Documentation technique

CLASSES

Classe Admins :

- **Description** :
  Cette classe définit le modèle d'un administrateur avec les attributs identifiant et mot de passe.
- **Attributs** :
  identifiant : Champ de base de données de type String correspondant à l'identifiant de l'admin
  password : Champ de base de données de type String correspondant au mot de passe de l'administrateur (crypté).
- **Propriétés et Méthodes** :
  identifiant() : Getter pour l'identifiant de l'administrateur.
  password() : Getter pour le mot de passe de l'administrateur.
  password.setter : Setter pour le mot de passe de l'administrateur.
  to_json() : Convertit l'objet Admins en format JSON.
  from_json() : Créer une instance Admins à partir d'un dictionnaire JSON.

Classe Category :

- **Description** :
  Cette classe définit le modèle d'une catégorie avec les attributs identifiant de la catégorie, le libellé, et la relation entre cette catégorie et les produits.
- **Attributs** :
  id_category : Champ de base de données de type Integer correspondant à l'id autoincrementé de la catégorie.
  libelle : Champ de base de données de type String correspondant au libellé de la catégorie
  devices : Relation one-to-many vers la classe Products.
- **Propriétés et Méthodes** :
  category() : Getter pour l'identifiant de la catégorie.
  libelle() : Getter pour le libellé de la catégorie.
  libelle.setter : Setter pour le libellé de la catégorie.
  to_json() : Convertit l'objet Category en format JSON.
  from_json() : Créer une instance Category à partir d'un dictionnaire JSON.

Classe Products :

- **Description** :
  Cette classe définit le modèle d'un produit avec les attributs identifiant du produit, le nom, la catégorie, la description, le prix, les données de l'image, la date de début de promotion, la date de fin de promotion, et le montant de la promotion.
- **Attributs** :
  id_product : Identifiant du produit (clé primaire de la table).
  name : Nom du produit.
  description : Description du produit.
  price : Prix du produit.
  image_data : Données de l'image du produit.
  category_name : Nom de la catégorie du produit.
  promo_start : Champ de type Date correspondant à la date de début de la promotion.
  promo_end : Champ de type Date correspondant à la date de fin de la promotion
  promo_amount : Champ de type Integer correspondant au pourcentage de promotion
- **Propriétés et Méthodes** :
  Des propriétés et des méthodes getter/setter sont utilisées pour accéder et modifier les attributs de la classe.
  to_json() : Convertit l'objet Products en format JSON.
  from_json() : Créer une instance Products à partir d'un dictionnaire JSON.

FONCTIONS

create_user() :

- **Description** :

Cette fonction est utilisée pour créer un nouvel utilisateur dans la base de données. Elle reçoit les données via une requête POST provenant du formulaire de création d'utilisateur.

- **Fonctionnement** :

Vérifie si l'utilisateur est identifié comme administrateur (session.get('user') == 'admin'). Si ce n'est pas le cas, l'utilisateur est redirigé vers la page de connexion.
Récupère les données du formulaire (l'identifiant et le mot de passe) envoyées via la requête POST.
Elle utilise ces données pour créer un nouvel objet de la classe Admins à partir de la méthode de classe from_json() (qui est une méthode personnalisée de la classe Admins pour créer des instances à partir de données JSON).
Elle ajoute cet utilisateur nouvellement créé à la base de données.
Elle redirige ensuite vers la vue « new_admin »

create_category() :

- **Description** :

  Cette fonction permet la création d'une nouvelle catégorie dans la base de données via le formulaire de création de catégorie.

- **Fonctionnement** :

  Vérifie si l'utilisateur est identifié comme administrateur (session.get('user') == 'admin'). Si ce n'est pas le cas, l'utilisateur est redirigé vers la page de connexion.
  Récupère le libellé de la catégorie à créer à partir des données du formulaire de création de catégorie.
  Transforme le libellé pour mettre la première lettre en majuscule.
  Prépare les données pour créer une nouvelle instance de Category en utilisant la méthode statique from_json() de la classe Category.
  Ajoute cette nouvelle catégorie à la base de données.
  Si l'ajout est réussi, redirige vers une vue nommée 'new_category.new_category' en passant un paramètre added_category comme True pour indiquer que la catégorie a été ajoutée avec succès. En cas d'erreur, redirige également vers la même vue mais avec added_category égal à False. La valeur de added_category permette le message affiché à la redirection.

create_product() :

- **Description** :

  Cette fonction permet la création d'un nouveau produit dans la base de données.

- **Fonctionnement** :

  Vérifie si l'utilisateur est identifié comme administrateur. Si ce n'est pas le cas, l'utilisateur est redirigé vers la page de connexion.
  Vérifie la présence et la validité de l'image envoyée dans la requête.
  Récupère les données du formulaire pour créer un nouveau produit en utilisant la méthode from_json() de la classe Products.
  Ajoute ce nouveau produit à la base de données.

modify_product(id_product: int) :

- **Description** :

  Cette fonction permet de modifier les informations d'un produit sélectionné.

- **Fonctionnement** :

  Récupère l'ancien produit à partir de la base de données à l'aide de son identifiant (id_product).
  Gère la mise à jour des informations du produit, y compris la gestion de l'image : stocke la nouvelle image si elle est téléchargée ou récupère l'ancienne image.
  Supprime l'ancien produit de la base de données et crée un nouveau produit avec les informations mises à jour.

delete_product(id_product: int) :

- **Description** :

  Cette fonction permet la suppression d'un article sélectionné.

- **Fonctionnement** :

  Supprime l'article correspondant à l'identifiant de la base de données.

homepage() :

- **Description** :

  Cette fonction gère l'affichage de la page d'accueil.

- **Fonctionnement** :

  Lorsqu'un utilisateur accède à la racine de l'application (page d'accueil), cette fonction est déclenchée.
  Elle rend le template 'index.html' en utilisant la fonction render_template de Flask pour afficher la page d'accueil de l'application.

catalogue() :

- **Description** :

  Cette fonction est responsable de l'affichage du catalogue complet des articles, avec les informations sur les produits et les promotions.

- **Fonctionnement** :

  Récupère la date actuelle.
  Obtient la liste de toutes les catégories et de tous les produits à afficher en utilisant les fonctions get_all_category() et get_all_product() provenant d'un fichier.
  Rend le template 'catalogue.html' en passant les produits, les catégories et la date actuelle pour l'affichage.

catalogue_update() :

- **Description** :

  Cette fonction gère la mise à jour dynamique du catalogue en fonction du choix de tri et de la catégorie sélectionnée par l'utilisateur.

- **Fonctionnement** :

  Récupère les données JSON envoyées par une requête POST, notamment la catégorie sélectionnée et l'ordre de tri.
  Utilise ces informations pour mettre à jour la liste des produits affichés en appelant la fonction get_sorted_product() avec la catégorie sélectionnée et le sens de tri.
  Rend dynamiquement un template en fonction de la mise à jour des produits.
  Retourne la réponse sous forme de JSON contenant le HTML mis à jour pour les produits.

login() :

- **Description** :

  Cette fonction affiche la page de connexion au back office.

- **Fonctionnement** :

  Rend le template 'login.html' pour afficher le formulaire de connexion au back office.

traitement() :

- **Description** :

  Cette fonction traite les identifiants envoyés par l'utilisateur pour vérifier s'il s'agit d'un administrateur.

- **Fonctionnement** :

  Vérifie si les identifiants envoyés par l'utilisateur correspondent à un administrateur en appelant la fonction get_all_users() pour obtenir tous les utilisateurs.
  Compare les identifiants entrés avec les identifiants stockés dans la liste des utilisateurs.
  Si les identifiants sont corrects, définit une session avec l'utilisateur en tant qu'administrateur et redirige vers la vue du back office.
  Si les identifiants sont incorrects, supprime la session de l'utilisateur et renvoie à la page de connexion avec une variable connexion égale à False pour afficher un message d'erreur.

logout() :

- **Description** :

  Cette fonction permet la déconnexion de l'administrateur.

- **Fonctionnement** :

  Supprime la clé 'user' de la session en utilisant session.pop, ce qui déconnecte l'administrateur.

  Redirige l'utilisateur vers la vue du catalogue en utilisant.

back_office() :

- **Description** :

  Cette fonction permet d'afficher la page du back office pour les administrateurs.

- **Fonctionnement** :

  Vérifie si l'utilisateur a une session active en tant qu'administrateur.

  Si l'utilisateur est un administrateur, récupère la liste de toutes les catégories et de tous les produits en utilisant des fonctions get_all_category() et get_all_product() provenant.

  Rend le template 'back-office.html' pour afficher la page du back office, en passant les produits, les catégories, la date actuelle, ainsi que des paramètres pour la gestion d'erreurs ou de succès lors d'opérations.

new_admin() :

- **Description** :

  Cette fonction permet d'afficher la page pour ajouter un nouvel utilisateur à la base de données.

- **Fonctionnement** :

  Vérifie si l'utilisateur a une session active en tant qu'administrateur.

  Si l'utilisateur est un administrateur, rend le template 'create-user.html' pour afficher la page d'ajout d'un nouvel utilisateur.

  Si l'utilisateur n'est pas un administrateur, redirige vers la vue de connexion.

new_category() :

- **Description** :

  Cette fonction permet d'afficher la page pour ajouter de nouvelles catégories.

- **Fonctionnement** :

  Vérifie si l'utilisateur a une session active en tant qu'administrateur.
  Si l'utilisateur est un administrateur, récupère la liste de toutes les catégories en utilisant la fonction get_all_category().
  Rend le template 'new_category.html' pour afficher la page d'ajout de nouvelles catégories en passant la liste des catégories.
  Si l'utilisateur n'est pas un administrateur, il est redirigé vers la vue de connexion.

product_update(product_id: int) :

- **Description** :

  Cette fonction permet de récupérer les informations d'un produit sélectionné pour les afficher dans la page de modification du produit.

- **Fonctionnement** :

  Vérifie si l'utilisateur a une session active en tant qu'administrateur
  Récupère les informations du produit sélectionné en utilisant l'identifiant product_id.
  Si le produit existe, prépare un dictionnaire product_to_modify contenant les informations du produit pour les afficher dans la page de modification.
  Si le produit n'existe pas, redirige vers la vue du back office.
  Si l'utilisateur n'est pas un administrateur, il est redirigé vers la vue de connexion.

encode_password(password: str) -> bytes :

- **Description** :

  Cette fonction prend un mot de passe en texte brut et renvoie le mot de passe haché.

- **Fonctionnement :**

  Elle utilise la bibliothèque bcrypt pour hacher le mot de passe en utilisant bcrypt.hashpw.

- **Renvoi :**

  Un mot de passe haché sous forme de bytes.

password_check(password_input, hashed_password) :

- **Description** :

  Cette fonction vérifie si un mot de passe en texte brut correspond à un mot de passe haché donné.

- **Fonctionnement** :

  Elle utilise bcrypt.checkpw pour comparer le mot de passe en texte brut avec le mot de passe haché.

- **Renvoi** :

  True si les mots de passe correspondent, False sinon.

get_all_category()

- **Description** :
  Récupère toutes les catégories présentes en base de données.
- **Fonctionnement** :
  Utilise Category.query.all() pour récupérer toutes les catégories, puis les formate dans une liste.
- **Renvoi** :
  Une liste des noms des catégories.

get_all_product()

- **Description** :
  Récupère tous les produits sous forme de liste JSON pour affichage.
- **Fonctionnement** :
  Utilise Products.query.all() pour récupérer tous les produits, puis les formate dans une liste JSON.
- **Renvoi** :
  Une liste de dictionnaires représentant les informations des produits.

get_sorted_product(categorie, sens)

- **Description** :
  Récupère les produits triés selon la catégorie et le sens de tri choisis par l'utilisateur.
- **Fonctionnement** :
  Utilise des requêtes filtrées avec éventuellement un tri décroissant ou croissant.
- **Renvoi** :
  Une liste de dictionnaires représentant les informations des produits triés.

get_all_users()

- **Description** :
  Récupère la liste de tous les utilisateurs présents en base de données.
- **Fonctionnement** :
  Utilise Admins.query.all() pour récupérer tous les utilisateurs, puis les formate dans une liste de dictionnaires.
- **Renvoi** :
  Une liste de dictionnaires représentant les informations des utilisateurs.