|  |
| --- |
| Projet Site de Ecommerce  FastBuy |

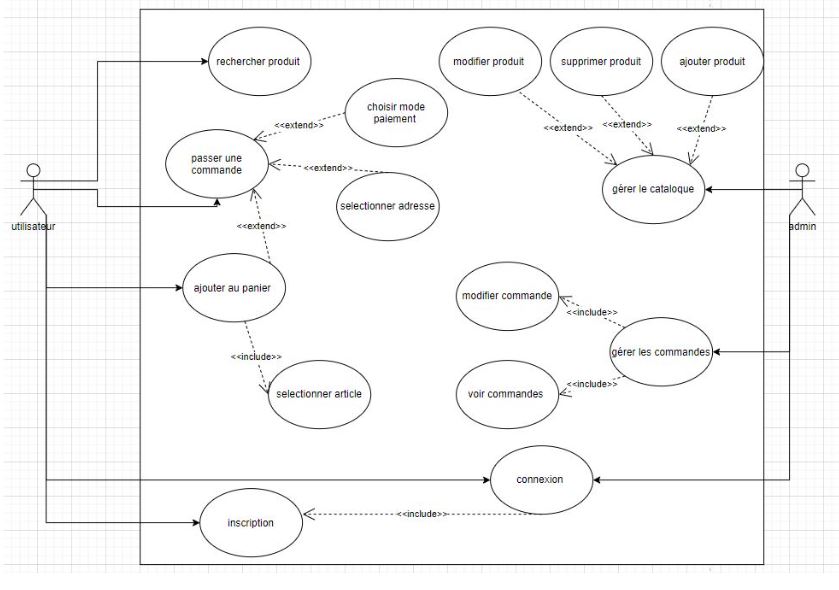
1. Modélisation et Scénarisation de l'Application

1.1 Objectifs Principaux

Au cœur de notre projet de commerce électronique se trouve l'objectif majeur de créer une plateforme en ligne spécialisée dans la vente de vêtements. Notre ambition est d'offrir une expérience d'achat diversifiée et conviviale, permettant aux utilisateurs de découvrir et d'acheter une variété de produits, notamment des chemises, des jeans, des t-shirts, des chaussures et des vestes. Les fonctionnalités clés incluent la navigation à travers une gamme de vêtements, la possibilité de passer des commandes en ligne.

1.2 Modélisation

Diagramme de cas d’utilisation :



Le diagramme de cas d'utilisation représente les interactions entre les utilisateurs (acteurs) et le système, identifiant les différentes fonctionnalités offertes par le système. Voici une description du diagramme de cas d'utilisation pour le projet Django :

Acteurs :

Utilisateur : Représente les personnes visitant le site, s'inscrivant, achetant des produits, etc.

Administrateur : Représente un administrateur du site, responsable de la gestion des produits, des commandes et des utilisateurs.

Cas d'utilisation :

S'inscrire : (Utilisateur)

Un utilisateur peut créer un compte en fournissant des informations telles que le nom, l'adresse e-mail, le mot de passe, etc.

Se connecter : (Utilisateur)

Un utilisateur peut se connecter à son compte en fournissant son adresse e-mail et son mot de passe.

Consulter les produits : (Utilisateur, Administrateur)

Les utilisateurs et les administrateurs peuvent consulter les produits disponibles.

Rechercher des produits : (Utilisateur, Administrateur)

Les utilisateurs et les administrateurs peuvent rechercher des produits en utilisant des mots-clés.

Ajouter des produits au panier : (Utilisateur)

Les utilisateurs peuvent ajouter des produits à leur panier d'achat.

Gérer le panier : (Utilisateur)

Les utilisateurs peuvent voir et modifier le contenu de leur panier.

Passer une commande : (Utilisateur)

Les utilisateurs peuvent finaliser leur commande en fournissant les détails d'expédition et effectuer le paiement.

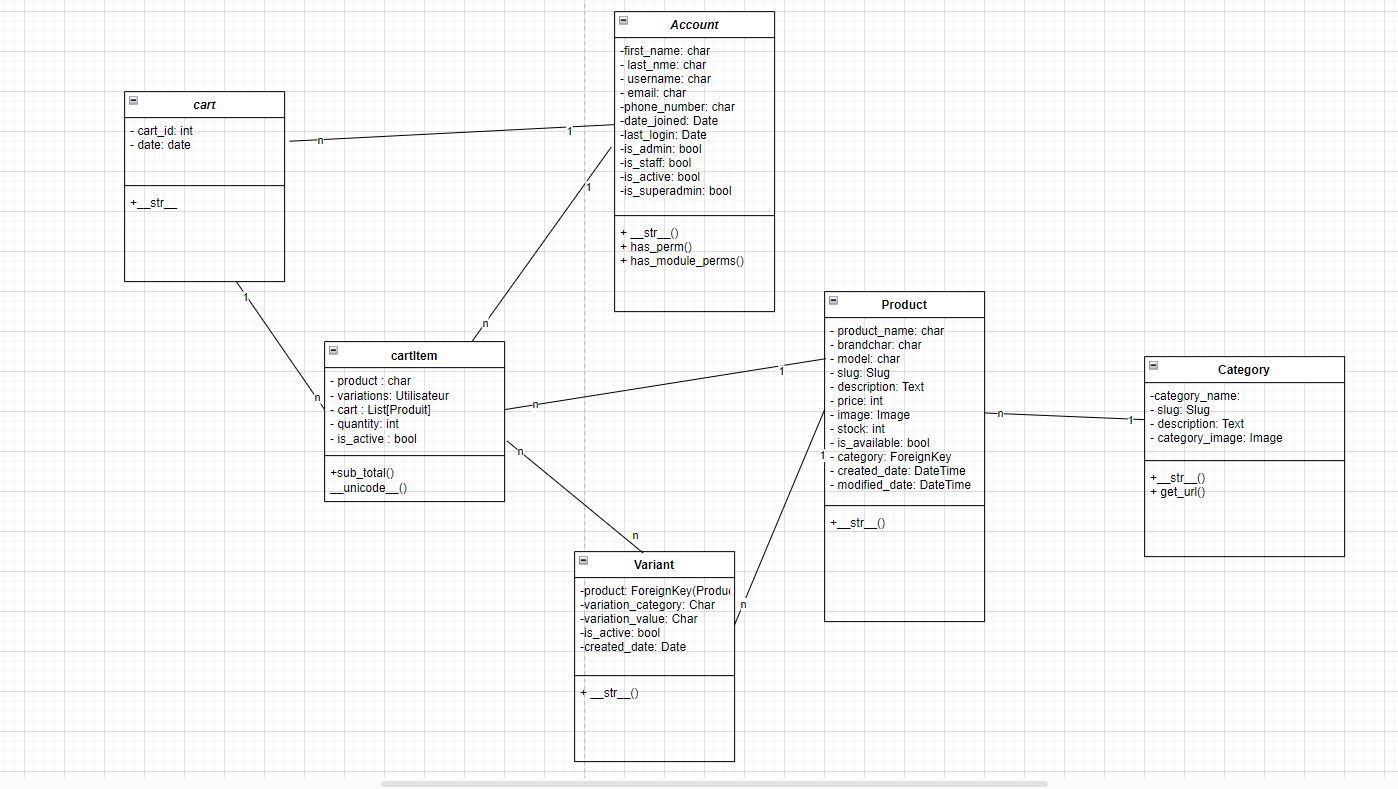
Gérer les produits : (Administrateur)

L'administrateur peut ajouter, modifier ou supprimer des produits et des catégories.

Gérer les commandes : (Administrateur)

L'administrateur peut voir et gérer les commandes passées par les utilisateurs.

Diagramme de class :



Le diagramme de classes représente la structure statique et les relations entre les différentes classes au sein d'un système logiciel. Voici une description des éléments présents dans le diagramme de classe que nous avons défini pour le projet :

**Account:** La classe représentant les utilisateurs du système.

Possède des champs tels que first\_name, last\_name, username, email, phone\_number, etc.

Hérite de la classe AbstractBaseUser de Django pour la gestion avancée des utilisateurs.

Utilise le gestionnaire de modèle MyAccountManager pour la création d'utilisateurs.

**Cart:** Représente le panier d'achat pour un utilisateur.

A un champ cart\_id pour identifier de manière unique chaque panier.

Possède une relation Many-to-One avec la classe Account, liant un panier à un utilisateur.

A une relation One-to-Many avec la classe CartItem, indiquant qu'un panier peut contenir plusieurs articles.

**CartItem:** Représente un article spécifique dans le panier.

A une relation Many-to-One avec la classe Product, indiquant l'article auquel il est associé.

A une relation Many-to-One avec la classe Cart, indiquant le panier auquel il appartient.

Contient des informations telles que la quantité et l'état d'activité.

**Category:** Représente les catégories auxquelles les produits appartiennent.

A une relation One-to-Many avec la classe Product, indiquant qu'une catégorie peut avoir plusieurs produits.

**Product:** Représente un produit disponible dans le magasin.

A une relation Many-to-One avec la classe Category, indiquant la catégorie à laquelle il appartient.

A une relation One-to-Many avec la classe Variant, indiquant qu'un produit peut avoir plusieurs variantes.

**Variant:** Représente une variante d'un produit sa couleur ou sa taille.

A une relation Many-to-One avec la classe Product, indiquant le produit auquel il est associé.

2. Architecture Générale du Code

2.1 Grands Concepts

Modèles (Models) :

Les modèles sont des classes Python qui définissent la structure des données du projet. Dans notre cas, les modèles incluent Account, Cart, CartItem, Category, Product, et Variant.

Les modèles interagissent avec la base de données pour stocker et récupérer des informations.

Vues (Views) :

Les vues définissent la logique métier. Elles interagissent avec les modèles pour récupérer des données, effectuer des opérations, et transmettre les résultats aux templates.

Les vues sont implémentées dans le fichier views.py par exemple, les fonctions store, by\_category dans l’application store.



Templates :

Les templates sont des fichiers HTML intégrant des balises Django pour générer du contenu dynamique à afficher à l'utilisateur.

Gestion des URLs :

Les URLconf (Configuration des URLs) dans le fichier urls.py définissent les correspondances entre les URL et les vues. Cela dirige les requêtes HTTP vers les fonctions de vue appropriées.

Les fichiers urls.py des applications (store, accounts, cart, etc.) définissent les URLs spécifiques à chaque application.

Exemple store :



Gestion du Formulaire :

Django inclut des fonctionnalités intégrées pour la gestion des formulaires. Les formulaires sont utilisés pour collecter et valider les données utilisateur.

Les formulaires sont définis dans des fichiers comme forms.py.

2.2 Explication des différents éléments du pattern MVT :

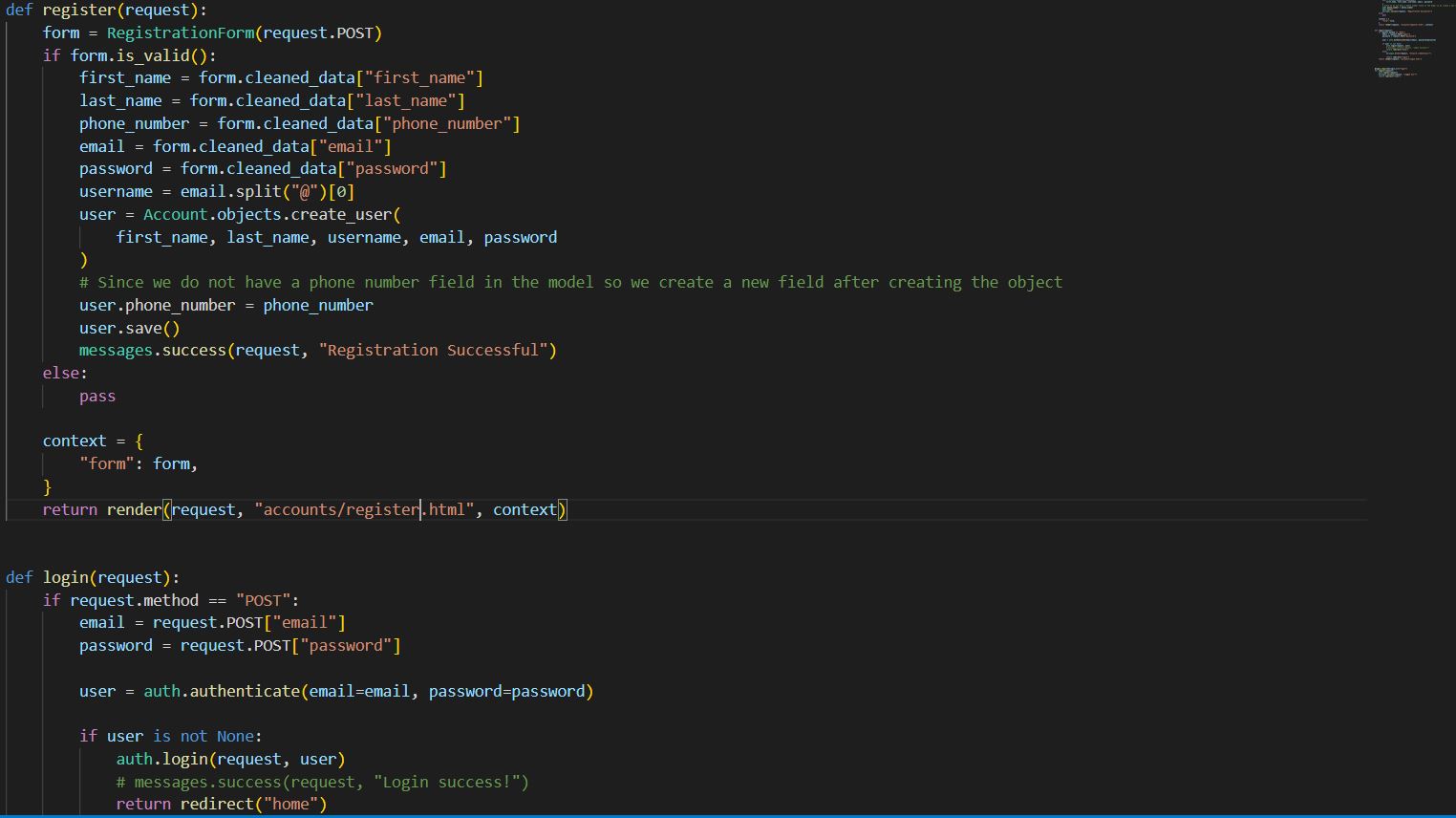
Dans notre code, nous avons structuré l'application en suivant les principes de Django MVT (Modèle-Vue-Template). Les principales composantes comprennent :

**APP accounts :**

Modèle (Model): Les modèles Django Account, définissent la structure de données pour les comptes utilisateur. Ces modèles sont responsables de la gestion des données et de leur interaction avec la base de données.

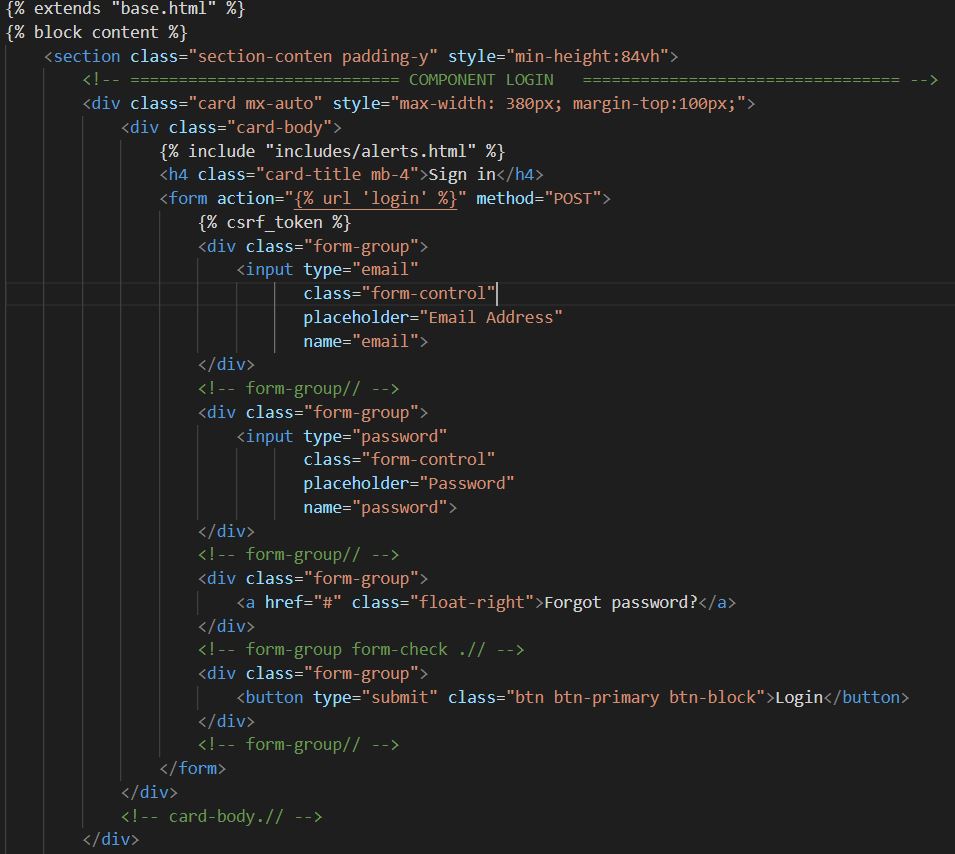


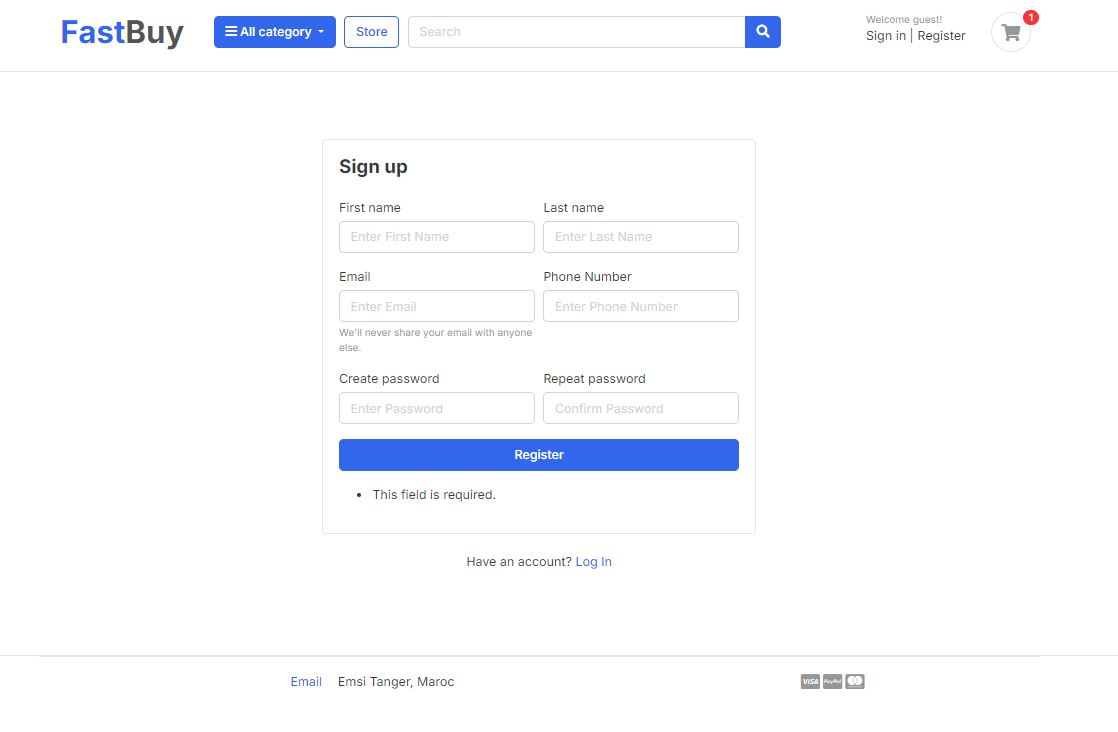
Vue (View): Les vues login et register définissent la logique métier pour les actions associées à la connexion et à l'inscription. Ces vues utilisent les modèles Django (Account) pour créer de nouveaux utilisateurs, authentifier les utilisateurs existants.



Template (Template): Les templates Django login.html et register.html, sont responsables de la présentation des données aux utilisateurs.

login.html: Ce template présente un formulaire de connexion avec des champs pour l'adresse e-mail et le mot de passe. En cas d'erreur lors de la connexion, il affiche un message d'erreur. Il propose également un lien pour s'inscrire s'il n'y a pas encore de compte.



register.html: Ce template présente un formulaire d'inscription avec des champs pour le prénom, le nom, l'adresse e-mail, le numéro de téléphone, le mot de passe, et la confirmation du mot de passe. En cas d'erreur lors de l'inscription, il affiche des messages d'erreur. Il propose également un lien pour se connecter si l'utilisateur a déjà un compte. 

**App cart :**

Modèle (Model): Les modèles Cart et CartItem définissent la structure de données pour le panier d'achat. Ils sont responsables de la gestion des données liées au panier.

Vue (View): Les vues définissent la logique métier associée au panier d'achat.

\_cart\_id: Cette fonction est un utilitaire qui obtient ou crée un identifiant de session (cart\_id) pour représenter le panier d'achat. Cela permet d'associer le panier d'un utilisateur à sa session.

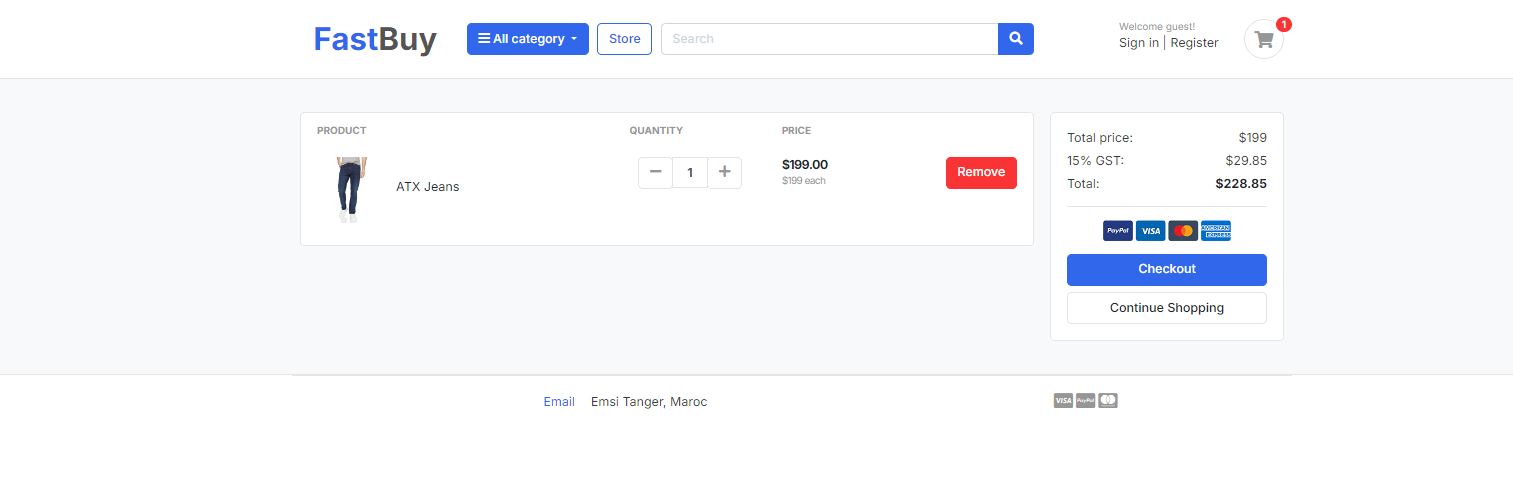
add\_to\_cart: Cette vue gère l'ajout d'un produit au panier d'achat. Elle utilise les modèles Product, Variant, Cart, et CartItem pour effectuer différentes opérations, telles que la création de nouveaux éléments de panier ou la mise à jour de la quantité des éléments existants.

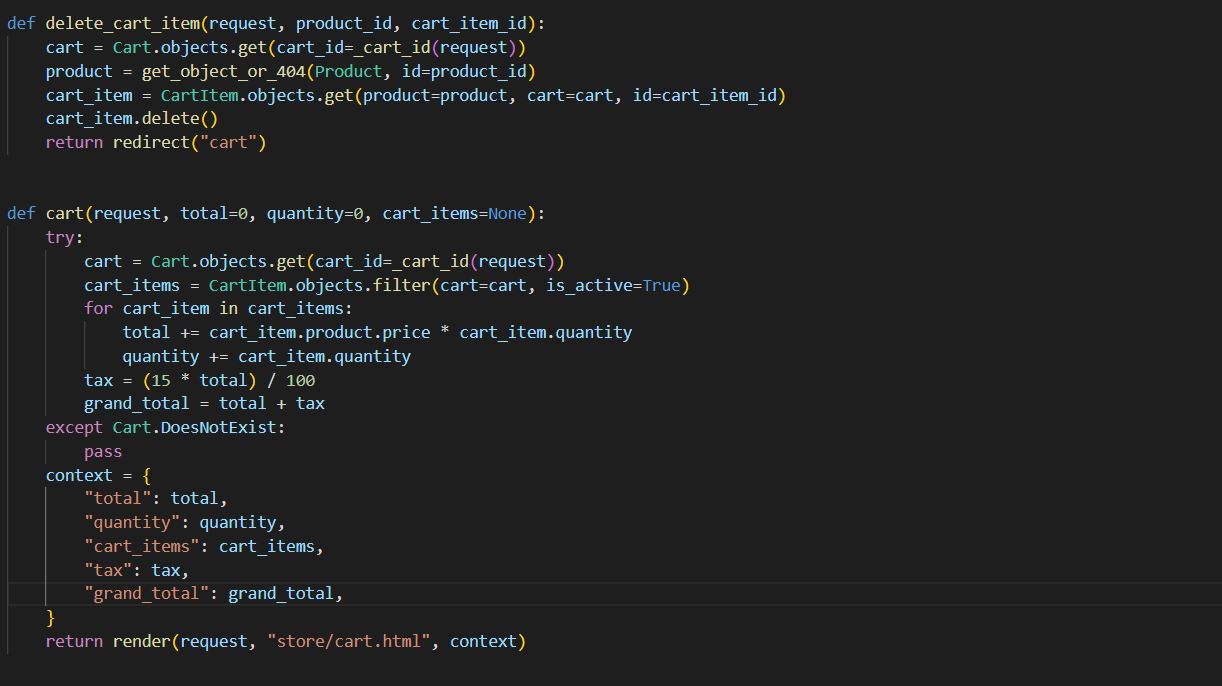
remove\_from\_cart: Cette vue gère la réduction de la quantité d'un produit dans le panier d'achat. Elle utilise les modèles Product, Cart, et CartItem pour effectuer la mise à jour appropriée.

delete\_cart\_item: Cette vue gère la suppression complète d'un élément du panier d'achat. Elle utilise les modèles Product, Cart, et CartItem pour effectuer la suppression.

cart: Cette vue génère la page du panier d'achat. Elle utilise les modèles Cart et CartItem pour obtenir les détails du panier, tels que la liste des articles, le total, la taxe, et le total général.

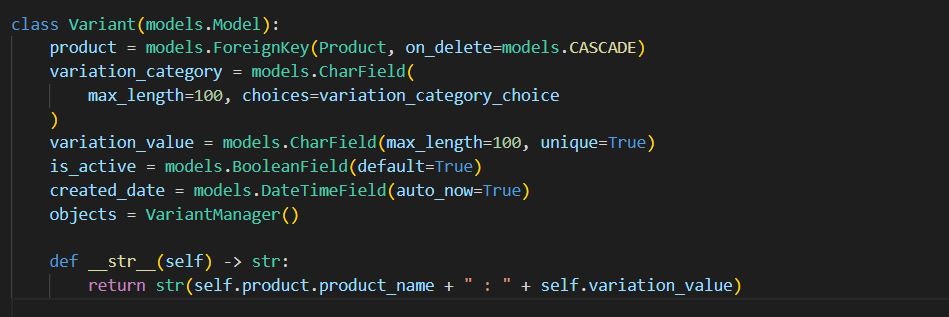
Template (Template): Le template cart.html est responsable de la présentation des données du panier d'achat aux utilisateurs. Il utilise des boucles pour afficher les détails de chaque article dans le panier, ainsi que le total, la taxe, et le total général. Il propose également des liens pour continuer les achats ou passer à la caisse.



****

**App Store :**Modèle (Model): Les modèles Product et Variant définissent la structure de données pour les produits et les variations. Ils sont responsables de la gestion des données liées aux produits et aux variantes.



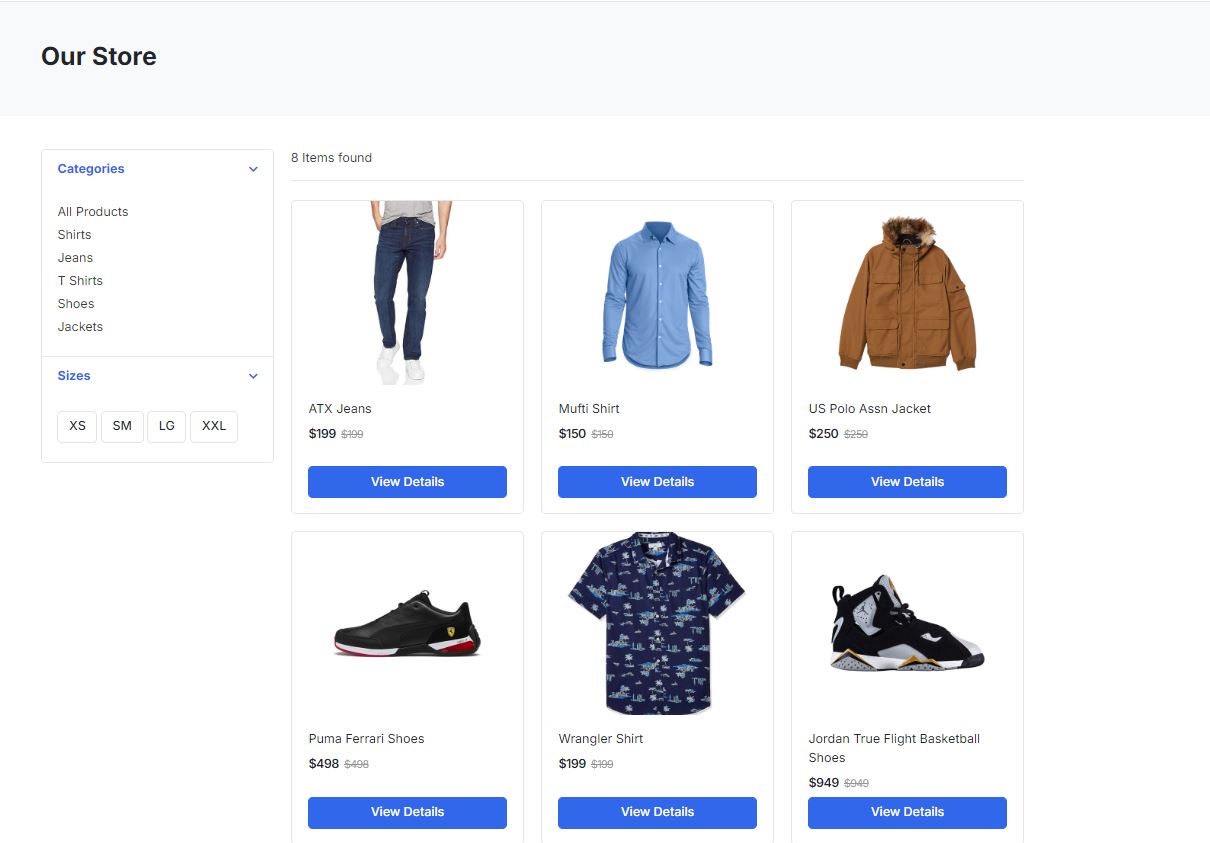


Vue : Les vues store, by\_category, product\_detail, et search interagissent avec les modèles Product et Category. Ces vues récupèrent les données nécessaires à partir de la base de données et les présentent à l'utilisateur. La vue store affiche tous les produits, la vue by\_category affiche les produits d'une catégorie spécifique, la vue product\_detail affiche les détails d'un produit, et la vue search effectue une recherche de produits.



La pagination affiche six produits dans une page.

Template (Template): Le template store.html est responsable de la présentation des données liées aux produits. Il inclut une section pour afficher les catégories, une section pour afficher les filtres de recherche, et une section pour afficher la liste des produits. Il utilise également la pagination pour organiser les produits sur plusieurs pages.

****

3. Gestion du Code

3.1 Gestion du Contrôle de Version (SCM)

Pour la gestion du contrôle de version de notre projet, GitHub a été utilisé comme plateforme principale. Bien que le projet ne soit pas développé en équipe, l'utilisation de GitHub offre plusieurs avantages, même pour un développeur unique.

Le dépôt GitHub associé à ce projet est accessible à l'adresse suivante : [https://github.com/hamzaymd/Django\_Group3EcommFastBuy].

3.2 Avantages du SCM

L'utilisation de GitHub en tant qu'outil de gestion du contrôle de version offre plusieurs avantages :

Historique des modifications : GitHub maintient un historique détaillé des modifications apportées au code, facilitant le suivi des évolutions du projet et la résolution des problèmes.

Gestion des versions : Les balises (tags) sont utilisées pour marquer des versions spécifiques du projet, ce qui simplifie la gestion des versions de l'application.

Sauvegardes et sécurité : GitHub sert également de sauvegarde externe du projet, assurant une certaine sécurité en cas de perte de données locales.

Le lien vers le dépôt GitHub est [insérez le lien ici]. Vous pouvez explorer le dépôt pour obtenir plus de détails sur le code et le processus de développement.

4. Perspectives

Plusieurs améliorations et fonctionnalités peuvent être envisagées pour finaliser et améliorer le projet. Certaines perspectives d'amélioration incluent :

Dashboard Client : L'implémentation d'un tableau de bord client peut considérablement améliorer l'expérience utilisateur en offrant aux utilisateurs enregistrés un moyen convivial de gérer leurs informations. Les fonctionnalités potentielles pourraient inclure la modification du profil, le suivi des commandes, la gestion des préférences, etc.

Optimisation du Code : Une optimisation continue du code est cruciale pour maintenir la performance et la maintenabilité de l'application. Certains aspects d'optimisation peuvent inclure :

Amélioration de l'Expérience Utilisateur (UX) : Effectuer une analyse approfondie de l'expérience utilisateur pour identifier les zones d'amélioration et apporter des ajustements au design et à l'ergonomie de l'application.

Fonctionnalités Additionnelles : Ajouter de nouvelles fonctionnalités qui pourraient enrichir l'application, en fonction des retours utilisateurs et des besoins émergents.

Sécurité : Effectuer des audits de sécurité réguliers, mettre à jour les dépendances, et implémenter les meilleures pratiques en matière de sécurité pour protéger l'application contre les menaces potentielles.

Tests Approfondis : Élargir la couverture des tests unitaires et d'intégration pour garantir la stabilité de l'application au fur et à mesure des évolutions.

Optimisation de la Base de Données : Examiner et ajuster les requêtes pour garantir des performances optimales de la base de données.

Ajout du Paiement en Ligne : L'intégration d'un système de paiement en ligne rendra l'application plus polyvalente et pratique pour les utilisateurs. Cela pourrait inclure l'intégration de passerelles de paiement populaires, la sécurisation des transactions, et la gestion des factures et des reçus.

Amélioration de la Sécurité :Avec l'ajout de fonctionnalités sensibles telles que le paiement en ligne, renforcer la sécurité de l'application devient une priorité. Cela pourrait impliquer la mise en œuvre de protocoles de sécurité tels que HTTPS, la protection contre les attaques CSRF, etc.