ROYAUME DU MAROC

UNIVERSITÉ ABDELMALEK ESSAÂDI

FACULTÉ DES SCIENCES JURIDIQUES,

ECONOMIQUES ET SOCIALES DE TANGER

Master Data Science pour l’économie et la finance

Module Langage de programmation

(S1)

**Rapport du Projet de fin de module :**

**Application web Fullstack basée sur le Framework Django.**

Réalisé par :

**GHARBI Nesrine**

Encadré par :

**Pr. ELAACHAK LOTFI**

Année universitaire : 2023/2024

Table des matières

[I. Introduction 3](#_Toc159705502)

[II. Conception de base 4](#_Toc159705503)

[Mise en œuvre étape par étape 4](#_Toc159705504)

[III. Autres fonctionnalités et leurs mises en place 11](#_Toc159705505)

[Barre de recherche 11](#_Toc159705506)

[Panier 11](#_Toc159705507)

[Paiement via stripe 12](#_Toc159705508)

[Implémentation de jeton CSRF 12](#_Toc159705509)

[IV. Conclusion et perspectives 13](#_Toc159705510)

# Introduction

Dans le cadre de nos études à la Faculté des sciences juridiques, économique et sociales de Tanger, plus précisément pendant le premier semestre du master Data Science pour l’économie et la finance, nous sommes appelés à mettre en place une application web full Stack basée sur le Framework Django.

Ce projet est une suite à notre cours et sous l'encadrement de Monsieur ELAACHAK, pour but d’approfondir nos connaissances en développement Web.

Nous allons traiter en premier lieu la conception de l’application, ensuite nous allons expliquer l’utilité d’autres fonctionnalités et comment elles sont implémentées.

Finalement nous allons conclure ce travail et nous citerons des idées afin d’améliorer davantage cette application.

# Conception de base

Pour réaliser ce projet, nous avons choisi d'utiliser le framework Django en raison de sa robustesse, de sa simplicité de mise en œuvre et de sa capacité à fournir une application web complète. Django offre un ensemble d'outils puissants pour le développement web, y compris la gestion des bases de données, l'authentification utilisateur, et la gestion des formulaires, ce qui en fait un choix idéal pour un projet de cette envergure.

## Mise en œuvre étape par étape

* Créer un projet normal : Ouvrir l'IDE et créer un projet normal en sélectionnant Fichier -> Nouveau projet.
* Installer Django : Ensuite, nous allons installer le module Django depuis le terminal. J’ai utilisé Git Bash pour effectuer cette tâche en lançant la commande python ***-m pip install django***.
* Vérifier la version installée de Django : Pour vérifier la version de Django installée, vous pouvez exécuter la commande ***python -m django -version*** comme indiqué ci-dessous.
* Créer un projet Django : Lorsque nous exécutons la commande ***django-admin startproject***, un projet Django est créé à l'intérieur du projet normal que nous avons déjà créé ici. ***django-admin startproject mysite.***
* Vérifier la version de Python3 : python3 -version
* Exécuter le serveur web par défaut de Django:- Django fournit en interne un serveur web par défaut sur lequel nous pouvons lancer nos applications. ***python manage.py runserver*** dans le terminal. Par défaut, le serveur fonctionne sur le port 8000. Accédez au serveur web à l'URL surlignée.

On ajoute maintenant les applications qu’on a créées dans le site E-commerce dans « settings.py ».

# Application definition

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'core',

    'store',

    'userprofile',

]

urls.py : Ce fichier contient tous les modèles d'URL utilisés par le site web.

from django.contrib import admin

from django.conf import settings

from django.conf.urls.static import static

from django.urls import path, include

from core.views import frontpage, about

from userprofile import views

from store import views

urlpatterns =[

    path('', frontpage, name='frontpage'),

    path('users/', include('userprofile.urls')),

    path('store/', include('store.urls')),

    path('about/', about, name='about'),

    path('admin/', admin.site.urls),

    path('cart/', views.cart\_view, name='cart\_view'),

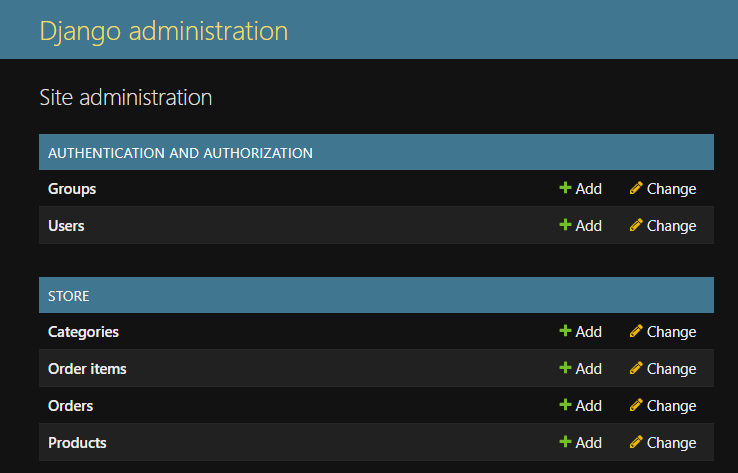
    path('cart/success/', views.success, name='success'),

    path('cart/checkout', views.checkout, name='checkout'),

    path('search/', views.search, name='search')

] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

models.py : La capture d'écran ci-dessous montre les modèles requis qu’on a créés. Ces modèles sont des tables qui seront stockées dans la base de données SQLite.



**Categories (Catégories)** :

* + Il représente une catégorie de produits.
  + Il a un titre et un slug pour identifier la catégorie.
  + Chaque catégorie peut contenir plusieurs produits.

**Product (Produit)** :

* + Il représente un produit disponible à la vente.
  + Il est associé à un utilisateur (vendeur), une catégorie, et possède des champs comme titre, description, prix, image, etc.
  + Il a une méthode **get\_display\_price()** qui retourne le prix.
  + Il a également une méthode **get\_thumbnail()** pour obtenir une miniature du produit.
  + Les images sont enregistrées dans un répertoire spécifique.

**Order (Commande)** :

* + Il représente une commande passée par un utilisateur.
  + Contient des informations sur le client (prénom, nom, adresse, etc.), le montant payé, et le statut de paiement.
  + Est associé à l'utilisateur qui a passé la commande.

**OrderItem (Élément de commande)** :

* + Il représente un produit spécifique commandé dans une quantité donnée.
  + Est associé à une commande et à un produit.
  + Contient des informations sur le prix et la quantité du produit commandé.

views.py : On a 3 fichiers views localisés dans « core », « store » et « userprofile » :

* « core » :

**Frontpage** : Cette vue récupère tous les produits à partir de la base de données à l'aide du modèle **Product** et les transmet au modèle de rendu. Elle vérifie également si l'utilisateur est authentifié et affiche quelques informations de débogage via des impressions dans la console.

def frontpage(request):

    products = Product.objects.all()

    return render(request, 'core/frontpage.html', {

        'products': products

    })

**About** : Cette vue est plus simple. Elle rend simplement un modèle nommé **'core/about.html'**.

def about(request):

    return render(request, 'core/about.html')

* « store » :

**add\_to\_cart** : Cette vue est utilisée pour ajouter un produit au panier. Elle récupère l'identifiant du produit et l'ajoute au panier, puis redirige vers la vue de visualisation du panier.

def add\_to\_cart(request, product\_id):

    cart = Cart(request)

    cart.add(product\_id)

    return redirect('cart\_view')

**Success** : Cette vue affiche une page de succès après qu'une commande a été passée avec succès.

def success(request):

    return render(request, 'store/success.html')

**Change\_quantity** : Cette vue permet de changer la quantité d'un produit dans le panier. Elle prend l'action (augmenter ou diminuer la quantité) en paramètre et ajuste la quantité du produit correspondant dans le panier.

def change\_quantity(request, product\_id):

    action = request.GET.get('action', '')

    if action:

        quantity = 1

        if action == 'decrease':

            quantity = -1

        cart = Cart(request)

        cart.add(product\_id, quantity, True)

    return redirect('cart\_view')

**Remove\_from\_cart** : Cette vue permet de supprimer un produit du panier. Elle récupère l'identifiant du produit et le supprime du panier, puis redirige vers la vue de visualisation du panier.

def remove\_from\_cart(request, product\_id):

    cart = Cart(request)

    cart.remove(product\_id)

    return redirect('cart\_view')

**Cart\_view** : Cette vue affiche le contenu actuel du panier.

def cart\_view(request):

    cart = Cart(request)

    return render(request, 'store/cart\_view.html', {

        'cart': cart

    })

**Checkout** : Cette vue gère le processus de paiement. Elle récupère les détails de la commande, calcule le montant total, crée une session de paiement Stripe, enregistre la commande et les éléments de commande associés, puis vide le panier. Enfin, elle retourne un objet JSON contenant des informations sur la session de paiement.

@login\_required

def checkout(request):

cart = Cart(request)

    if request.method == 'POST':

        print('-----------')

        data = json.loads(request.body)

***(saut dans le code)***

stripe.api\_key = settings.STRIPE\_SECRET\_KEY

        session = stripe.checkout.Session.create(

            payment\_method\_types=['card'],

            line\_items=items,

            mode='payment',

            success\_url=f'{settings.WEBSITE\_URL}cart/success/',

            cancel\_url=f'{settings.WEBSITE\_URL}cart/',

        )

        payment\_intent = session.payment\_intent

return JsonResponse({'session': session, 'order': payment\_intent})

    else:

        form = OrderForm()

***(saut dans le code)***

    return render(request, 'store/checkout.html', {

        'cart': cart,

        'form': form,

        'pub\_key': settings.STRIPE\_PUB\_KEY,

    })

**Category\_detail** : Cette vue affiche les détails d'une catégorie spécifique et les produits associés à cette catégorie.

def category\_detail(request, slug):

    category = get\_object\_or\_404(Category, slug=slug)

    products = category.products.all()

    return render(request, 'store/category\_detail.html', {

        'category': category,

        'products': products

    })

**Product\_detail** : Cette vue affiche les détails d'un produit spécifique.

def product\_detail(request, category\_slug, slug):

    product = get\_object\_or\_404(Product, slug=slug)

    return render(request, 'store/product\_detail.html', {

        'product': product

    })

* « userprofile » :

**Myaccount** : Cette vue affiche la page "Mon compte" lorsqu'un utilisateur est connecté. Elle rend simplement le modèle **'userprofile/myaccount.html'**.

def myaccount(request):

    return render(request, 'userprofile/myaccount.html')

**Myaccount\_after\_login** : Cette vue est utilisée pour rediriger l'utilisateur vers la page "Mon compte" après sa connexion.

def myaccount\_after\_login(request):

    return redirect('myaccount')

**Frontpage\_after\_logout** : Cette vue est utilisée pour rediriger l'utilisateur vers la page d'accueil après sa déconnexion.

def frontpage\_after\_logout(request):

    return redirect('frontpage')

**Signup** : Cette vue gère le processus d'inscription des utilisateurs. Si la méthode de la requête est POST, elle vérifie la validité du formulaire d'inscription (**UserCreationForm**). Si le formulaire est valide, elle crée un nouvel utilisateur, le connecte, crée un profil utilisateur (**Userprofile**), puis redirige l'utilisateur vers la page d'accueil. Si la méthode de la requête n'est pas POST, elle affiche simplement le formulaire d'inscription.

def signup(request):

    if request.method == 'POST':

        form = UserCreationForm(request.POST)

        print(form)

        if form.is\_valid():

            user = form.save()

            login(request, user)

            userprofile = Userprofile.objects.create(user=user)

            return redirect('frontpage')

    else :

        print('hello from else')

        form = UserCreationForm()

    return render(request, 'userprofile/signup.html', {

        'form': form

    })

Ensuite on mappe les vues dans le dossier store\urls.py.

urlpatterns = [

    path('search/', views.search, name='search'),

    path('add-to-cart/<int:product\_id>/', views.add\_to\_cart, name='add\_to\_cart'),

    path('change-quantity/<str:product\_id>/', views.change\_quantity, name='change\_quantity'),

    path('remove-from-cart/<int:product\_id>/', views.remove\_from\_cart, name='remove\_from\_cart'),

    # path('cart/', views.cart\_view, name='cart\_view'),

    # path('cart/checkout', views.checkout, name='checkout'),

    path('<slug:slug>/', views.category\_detail, name='category\_detail'),

    path('<slug:category\_slug>/<slug:slug>/', views.product\_detail, name='product\_detail'),

]

# Autres fonctionnalités et leurs mises en place

## Barre de recherche

Cette fonction récupère la requête de recherche à partir des paramètres GET, filtre les produits en fonction de la requête et renvoie les résultats.

def search(request):

    query = request.GET.get('query', '')

    products = Product.objects.filter(status=Product.ACTIVE).filter(Q(title\_\_icontains=query) | Q(description\_\_icontains=query))

    return render(request, 'store/search.html', {

        'query': query,

        'products': products,

    })

Grâce au modèle html « search.html » on définit le titre de la page comme "Search" et on affiche les produits correspondants à la requête de recherche spécifiée. Pour chaque produit trouvé, il affiche son titre, son prix et éventuellement une image, le tout présenté dans une mise en page responsive. En l'absence de produits correspondants, un message indiquant l'absence de résultats est affiché. Ce modèle offre une interface utilisateur claire et conviviale pour explorer les résultats de la recherche dans une application web de commerce électronique.

## Panier

class Cart(object):

    def \_\_init\_\_(self, request):

        self.session = request.session

        cart = self.session.get(settings.CART\_SESSION\_ID)

        if not cart:

            cart = self.session[settings.CART\_SESSION\_ID] = {}

        self.cart = cart

La classe **Cart** est instanciée dans les vues à partir de l'objet **request**, qui contient les informations de session de l'utilisateur. Tant qu'un utilisateur a une session active sur le site, il peut ajouter, modifier ou supprimer des articles dans son panier, même s'il n'est pas connecté.

Mais pour procéder au checkout il doit évidemment se connecter ou s’inscrire et ce grâce au décorateur **@login\_required** fourni par Django, qui redirigera automatiquement les utilisateurs non connectés vers la page de connexion avant de leur permettre d'accéder à la vue.

## Paiement via Stripe

On commence premièrement par créer un compte Stripe et générer les clés (publique et privée)

Ensuite on les intègre dans le fichier « settings.py »

STRIPE\_PUB\_KEY = 'pk\_test\_51OmyP7KlQcYuVFxManzKmMT6AJcZ4lI7Oqq7ZXOqRiDx9f7q6e9XCbX9G3Jx1Pyqp4t3nRiOghdWEro78\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

STRIPE\_SECRET\_KEY = 'sk\_test\_51OmyP7KlQcYuVFxMFA78wnZ15iplM8R2JNDMO1vtSJPPFe0aK06WylF88V4RO7UcpVN2vIomFrJYEKa\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

Par la suite on configure la clé d'API secrète de Stripe avec celle spécifiée dans les paramètres Django (**STRIPE\_SECRET\_KEY**) dans « views.py », et on crée la méthode « **stripe.checkout.Session.create()** » qui crée une nouvelle session de paiement Stripe. On récupère l'identifiant du paiement associé à la session de paiement Stripe «  **payment\_intent = session.payment\_intent**» et c’est tout pour le backend.

Dans « checkout.html » on a utilisé un script JavaScript qui facilite le processus de paiement côté client en utilisant l'API Stripe, en envoyant les informations de paiement à la vue Django et en redirigeant l'utilisateur vers la page de paiement Stripe pour finaliser la transaction.

## Implémentation de jeton CSRF

Le jeton CSRF vise à empêcher les attaques où un utilisateur authentifié involontairement déclenche une action malveillante sur le site. Dans le contexte des paiements, cela protège contre les tentatives de manipulation de transactions par des tiers non autorisés.

Son ajout est primordial dans la page du checkout, sign in et sign up.

Son implémentation est simple il suffit d’ajouter {% csrf\_token %} dans les blocs html « form » des pages concernées, pour le checkout le jeton est inclus dans le script JavaScript.

# Conclusion et perspectives

Dans le cadre de notre parcours de master Data science pour l’économie et la finance, nous sommes engagés dans la réalisation de divers projets et ateliers couvrant une gamme variée de modules. L'un de ces modules est celui de ‘’Langage de programmation’’, où nous avons entrepris la conception et le développement d'une application Web dédiée à la gestion des commandes sur la plateforme Django.

Au cours de ce projet, nous avons pu découvrir l’univers du développement Web et acquérir une maîtrise plus ou moins fluide des composants Django.

Cependant, notre ambition dépasse largement ces réalisations initiales. Notre objectif est de continuer à améliorer cette application en optimisant ses fonctionnalités existantes et en introduisant de nouvelles fonctionnalités interactives pour offrir une expérience utilisateur encore plus enrichissante.

Par exemple ajouter la fonctionnalité d’avoir plusieurs vendeurs (comme Jumia, Amazon ou Aliexpress), permettre aux utilisateurs d’écrire des avis et évaluations des produits, un support client en ligne, etc.