ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Programmation Web - Django

Compte Rendu du contrôle

TIRGANI Badreddine

4IIR-G3

On souhaite développer une application e-shoping basée sur Django.

Les entités avec ses attributs :

L'entité Produit est définie par son id (clè primaire), produitRef, nomProduit, dateProduction et prix. Categorie est défini par son id, nomCategorie. Entité Commande est défini par id, referenceCmd et dateCmd. Et finalement, Personne défini par son id, nom, prenom, email

Les relations :

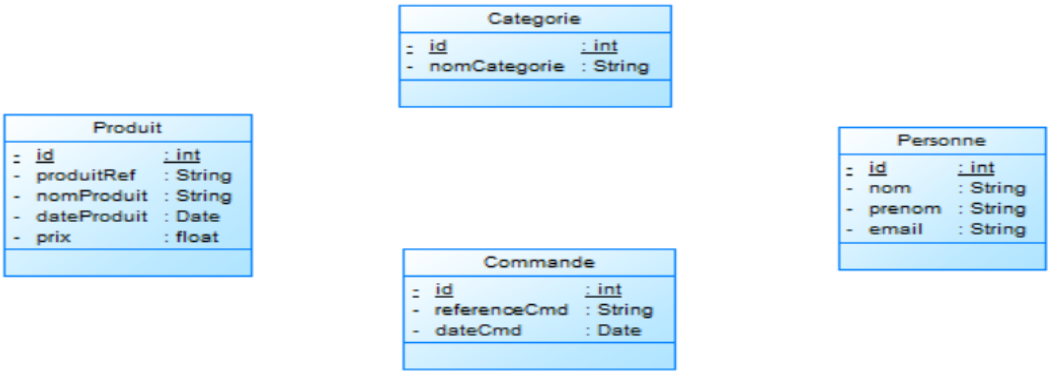
Chaque produit est appartient à une categorie. Ainsi, chaque categorie contient un ou plusieurs produits. Chaque commande contient au moins un produit et doit être passer par une seule personne.

NB : En ce qui concerne la base de données, il est préférable de garder la configuration par défaut et de travailler avec SQLite.

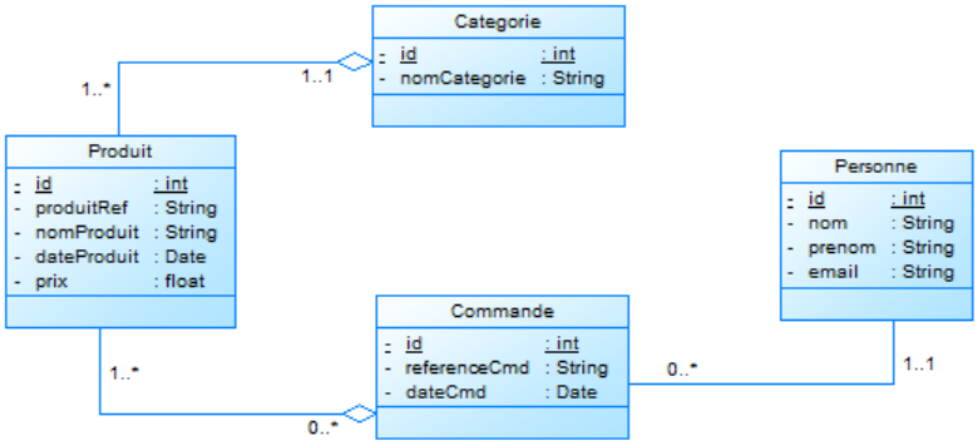
LA REALISATION :

Conception et architecture

1. Etablir un diagramme de classes qui montre les entités de l'application. On ne représentera que les attributs.



2. Etablir les relations entre les différentes entités de l'application



Implémentation

1. Créer un projet django nommé « contrôle », dedans ce projet créer une application avec votre nom de cette forme « nom_prenom »

```
PS C:\Users\atlas pro electro\PycharmProjects\controle> python manage.py startapp TIRGANI_Badreddine
```

2. La partie ORM :

a. Réaliser les modèles de l'application avec les différents champs de chaque modèle.

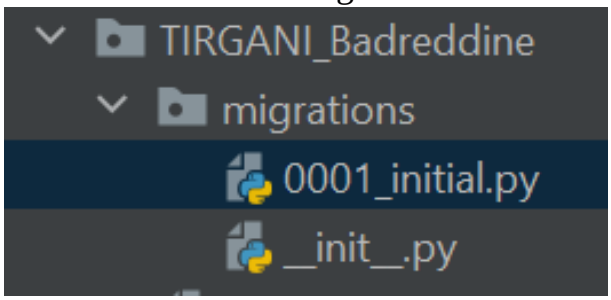
```python
class Personne(models.Model):
    id = models.IntegerField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')
    nom = models.CharField(max_length=45)
    prenom = models.CharField(max_length=45)
    email = models.CharField(max_length=45)

class Categorie(models.Model):
    id = models.IntegerField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')
    nomCategorie = models.CharField(max_length=45)
    personne = models.ForeignKey(Personne, on_delete=models.CASCADE, related_name='Personne', default=None)

class Commande(models.Model):
    id = models.IntegerField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')
    referenceCmd = models.CharField(max_length=45)

class Produit(models.Model):
    produitRef = models.CharField(max_length=45)
    nomProduit = models.CharField(max_length=45)
    dateProduit = models.DateField(max_length=45)
    prix = models.FloatField()
    categorie = models.ForeignKey(Categorie, on_delete=models.CASCADE, related_name='Categorie', default=None)
    commande = models.ForeignKey(Commande, on_delete=models.CASCADE, related_name='Commande', default=None)
```

b. Effectuer des migrations afin de créer les tables dans la base de données.



c. A l'aide de l'utilitaire shell ajouter 2 enregistrements dans la table categorie et 5 enregistrements dans la table produits.

3. Ecrire des views permettant de répondre aux spécifications fonctionnelles suivantes :

a. Renvoie la liste des produits 10 éléments par page.

```
#PRODUITS####################
# la liste des clients
def produits(request):
    produits_list = Produit.objects.all()
    page = request.GET.get('page', 1)
    paginator = Paginator(produits_list, 10)
    try:
        produits = paginator.page(page)
    except PageNotAnInteger:
        produits = paginator.page(1)
    except EmptyPage:
        produits = paginator.page(paginator.num_pages)
    context= {'produits': produits}
    return render(request, 'produit/produits.html', context)
```

b. Consulter les détails d'un produit

```
# pour afficher les détail d'un patient
def detailsProduit(request, id):
    produits = Produit.objects.get(id=id)
    context = {'produits': produits}
    return render(request, 'produit/detailsProduit.html', context)
```

c. Consulter les produits dont le nom contient un mot clé

```
def searchProduit(request):
    searched = request.POST['searched']
    clients_searched = Produit.objects.filter(produitRef__contains=searched)
    page = request.GET.get('page', 1)
    paginator = Paginator(clients_searched, 10)
    try:
        produits = paginator.page(page)
    except PageNotAnInteger:
        produits = paginator.page(1)
    except EmptyPage:
        produits = paginator.page(paginator.num_pages)
    context = {'produits': produits}
    return render(request, 'produit/produits.html', context)
```

d. Consulter les commandes d'une Personne

```python
def commandesPersonne(request, id):
    commandes_list = "none"
    commandes_list = Commande.objects.filter(personne_id=id)
    taille = len(commandes_list)
    if taille == 0:
        operations_list = "none"
        context = {'commandes': commandes_list}
        return render(request, 'commande/commandes.html')
    else:
        page = request.GET.get('page', 1)
        paginator = Paginator(commandes_list, 10)
        try:
            commandes = paginator.page(page)
        except PageNotAnInteger:
            commandes = paginator.page(1)
        except EmptyPage:
            commandes = paginator.page(paginator.num_pages)
        context= {'commandes': commandes}
        return render(request, 'commande/commandes.html', context)
```

4. Mapping des urls

a. Faire le routage des urls

```python
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index, name='index'),
    path('personnes/', views.personnes),

    path('addPersonne/', views.addPersonne),
    path('personne/createPer', views.createPer),
    path('personne/editPer/<int:id>', views.editPer),
    path('personne/deletePersonne/<int:id>', views.deletePersonne),
    path('personne/detailsPersonne/<int:id>', views.detailsPersonne),
    path('personne/editPersonne/<int:id>', views.editPersonne),
    path('personne/commandesPersonne/<int:id>', views.commandesPersonne),
    path('personne/searchPersonne', views.searchPersonne, name='search-personne'),
    path('personne/formulaire', views.FormPersonne),

]
```

```python
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index, name='index'),
    path('produits/', views.produits),

    path('addProduit/', views.addProduit),
    path('produit/createProd', views.createProd),
    path('produit/editProd/<int:id>', views.editProd),
    path('produit/deleteProduit/<int:id>', views.deleteProduit),
    path('produit/detailsProduit/<int:id>', views.detailsProduit),
    path('produit/editProduit/<int:id>', views.editProduit),
    path('produit/searchProduit', views.searchProduit, name='search-produit'),
    path('produit/formulaire', views.FormProduit),


]
```

```python
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index, name='index'),
    path('commandes/', views.commandes),

    path('addCommande/', views.addCommande),
    path('commande/createCom', views.createCom),
    path('commande/editCom/<int:id>', views.editCom),
    path('commande/deleteCommande/<int:id>', views.deleteCommande),
    path('commande/detailsCommande/<int:id>', views.detailsCommande),
    path('commande/editCommande/<int:id>', views.editCommande),
    path('commande/searchCommande', views.searchCommande, name='search-commande'),
    path('commande/formulaire', views.FormCommande),


]
```

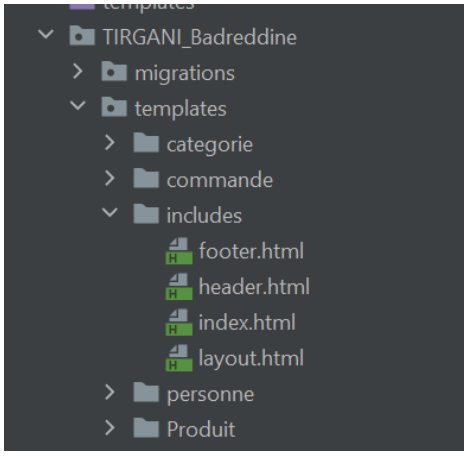
```python
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index, name='index'),
    path('categories/', views.categories),

    path('addCategorie/', views.addCategorie),
    path('categorie/createCat', views.createCat),
    path('categorie/editCat/<int:id>', views.editCat),
    path('categorie/deleteCategorie/<int:id>', views.deleteCategorie),
    path('categorie/detailsCategorie/<int:id>', views.detailsCategorie),
    path('categorie/editCategorie/<int:id>', views.editCategorie),
    path('categorie/searchCategorie', views.searchCategorie, name='search-categorie'),
    path('categorie/formulaire', views.FormCategorie),

]
```

5. Templates : à l'aide du langage HTML et du langage du template

a. Définir un fichier de template de base comme layout de votre application, dont vous inclue header et footer ainsi vous définit un block content



b. Réaliser une interface de présentation avec pagination

Voir execution

c. Intégrer une fonctionnalité de recherche qui permet de chercher les produit dont le nom contient un mot clé saisi avec pagination.

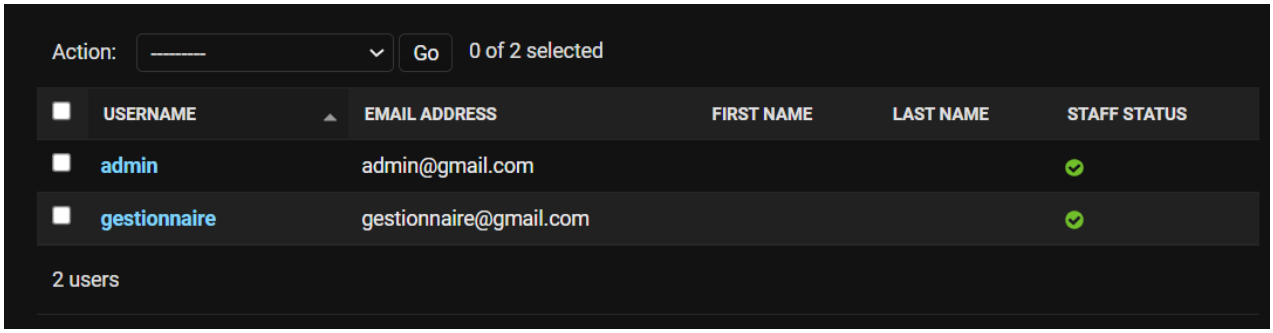
```
<form method="post" action="{% url 'search-produit' %}">
    {% csrf_token %}
    <div class="card-header">
        <div class="input-group">
          <div class="form-outline">
            <input type="search" class="form-control" placeholder="search" name="searched"/>
          </div>
          <button type="submit" class="btn btn-primary">
            <i class="fas fa-search"></i>
          </button>
        </div>
    </div>
</form>
```

6. Espace admin : affecter un administrateur à l'application

a. Créer un admin pour l'application

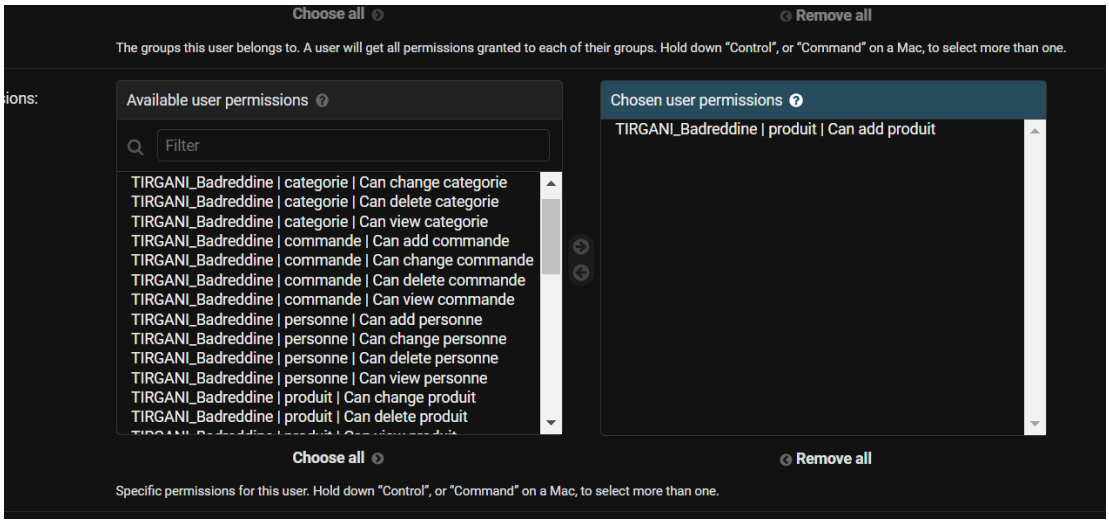
```
PS C:\Users\atlas pro electro\PycharmProjects\controle> python manage.py
Username (leave blank to use 'elharharimilouda'): admin
Email address: admin@gmail.com
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

b. Créer un utilisateur nommé « gestionnaire »



c. Enregistrer les modèles de l'application dans l'interface admin

d. Donner les privilèges d'ajouter un produit à l'utilisateur « gestionnaire »



7. Les formulaire : à l'aide de django.forms :

a. Réaliser un formulaire général pour la recherche

```
class GeneralForm(forms.Form):
    search = forms.CharField()
```

b. Générer les formulaires des modeles de l'application

```python
class PersonneForm(ModelForm):
    class Meta:
        model = Personne
        fields = ['id', 'nom', 'prenom', 'email']

class CommandeForm(ModelForm):
    class Meta:
        model = Commande
        fields = ['id', 'referenceCmd', 'dateCmd', 'personne']

class CategorieForm(ModelForm):
    class Meta:
        model = Categorie
        fields = ['id', 'nomCategorie']

class ProduitForm(ModelForm):
    class Meta:
        model = Produit
        fields = ['id', 'produitRef', 'nomProduit', 'dateProduit', 'prix', 'categorie', 'commande']
```