

Rapport Projet Django



Réalisé par : IRGUI Ilyas

Encadré par : Mr. HABIB AYAD

Filière : IA&GI

Année Universitaire : 2020/2021

Table de matières

I. Introduction

- 1.1. « Django » c'est quoi ?
- 1.2. Comment ça marche le Back-end « DJANGO » ?
- 1.3. L'architecture M.V.T

II. Création du projet

- 2.1. Configuration de l'environnement
- 2.2. Création des Applications Django
- 2.3. Le modèle MCD et la connexion à MYSQL
- 2.4. Création du Super utilisateur
- 2.5. Implémentation des modèles

III. Configuration du Rest_API

- 3.1. La sérialisation des modèles
- 3.2. L'authentification par Token

IV. Les Manipulations CRUD

- 4.1. Rechercher
- 4.2. Ajouter
- 4.3. Modifier
- 4.4. Supprimer

V. Conclusion

I. Introduction

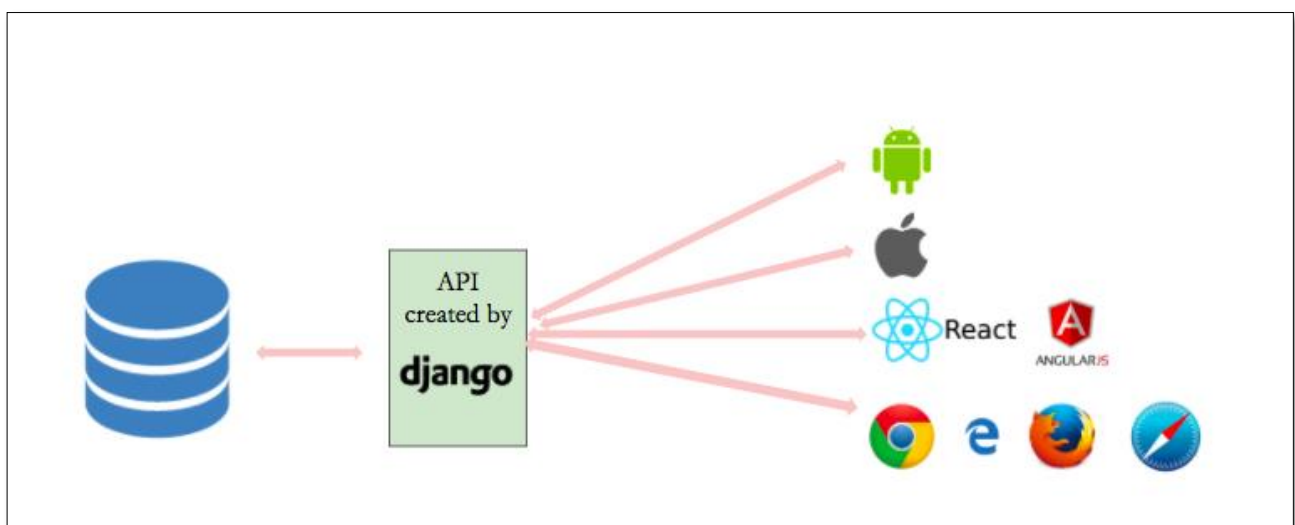
1.1. Django c'est quoi ?

Django est un Framework python open-source consacré au développement web 2.0. Les concepteurs de Django lui ont attribué le slogan suivant : " Le Framework web pour les perfectionnistes sous pression ". Il est donc clairement orienté pour les développeurs ayant comme besoin de produire un projet solide rapidement et sans surprise ... c'est à dire à tous les développeurs !

Comme il est toujours compliqué de partir de rien, Django vous propose une base de projet solide. Django est donc une belle boîte à outils qui aide et oriente le développeur dans la construction de ses projets.

Pour la petite histoire Django a vu le jour en 2003 et a été publié sous licence BSD en juillet 2005.

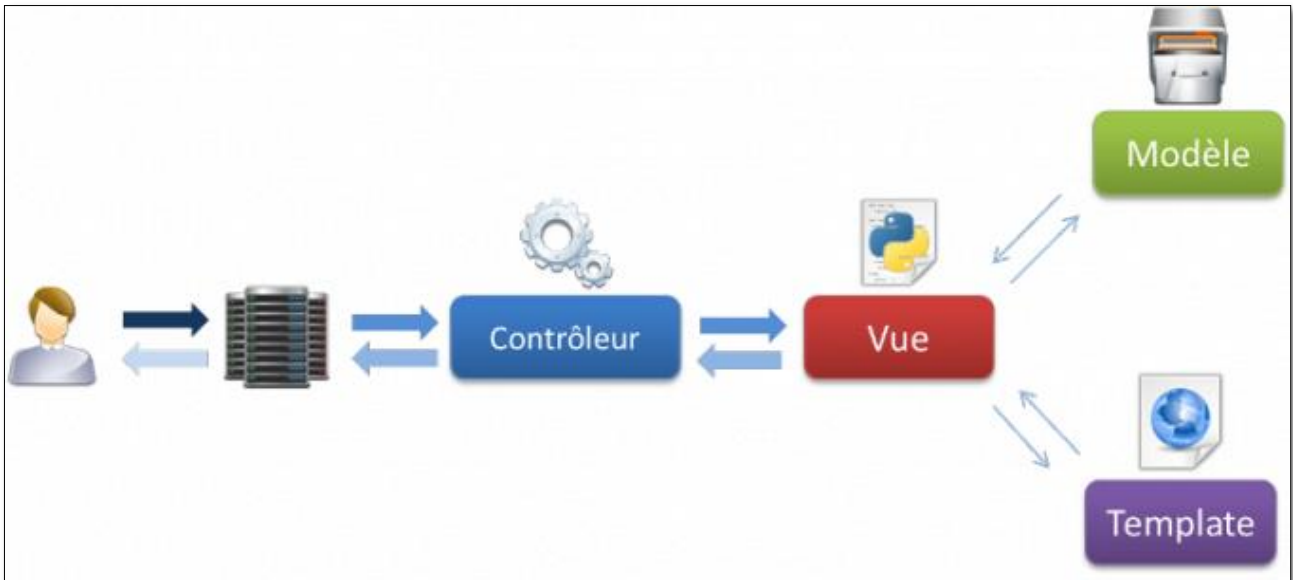
1.2. Le Back-end 'DJANGO' :



Comme le montre le schéma ci-dessus, le Framework DJANGO est exécuté uniquement coté serveur ou comme on préfère l'appeler BACK-END. (Tout comme le PHP avec LARAVEL ou JAVA avec JEE/SPRING). C'est le BACK-END qui communique avec la SGBD (SQL/NOSQL) pour l'interprétation des données. En ce qui concerne le FRONT-END, qui est l'équivalent des Vues dans le modèle MVC, son rôle est constitué de faciliter le développement du GUI.

1.3. L'architecture M.V.T :

Le MVT représente une architecture orientée autour de trois pôles : le modèle, la vue et le Template . Elle s'inspire de l'architecture MVC, très répandue dans les Frameworks web. Son objectif est de séparer les responsabilités de chaque pôle afin que chacun se concentre sur ses tâches.



Modèle

Le modèle interagit avec la base de données. Sa mission est de chercher dans une base de données les items correspondant à une requête et de renvoyer une réponse facilement exploitable par le programme.

Template

Un Template est un fichier HTML qui peut recevoir des objets Python et qui est lié à une vue (nous y reviendrons). Il est placé dans le dossier templates.

Vue

La vue joue un rôle central dans un projet structuré en MVT : sa responsabilité est de recevoir une requête HTTP et d'y répondre de manière intelligible par le navigateur.

La vue réalise également toutes les actions nécessaires pour répondre à la requête :

- Si une interaction avec la base de données est requise, la vue appelle un modèle et récupère les objets renvoyés par ce dernier.
- Si un gabarit est nécessaire, la vue l'appelle.

II. Création du projet

2.1. Configuration de l'environnement :

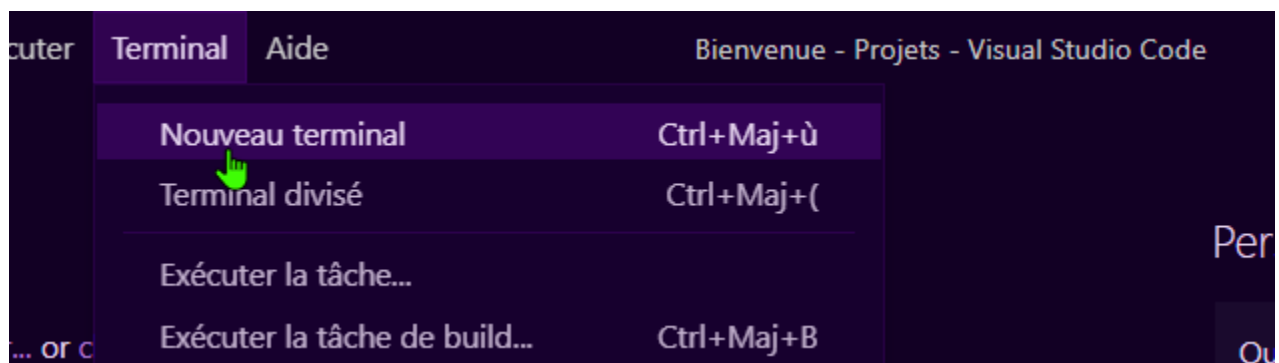
Premièrement on crée un nouveau env :

```
conda create --name myenv
```

Ensuite on lance Anaconda-prompt et taper la commande suivante :

```
conda install -c anaconda django
```

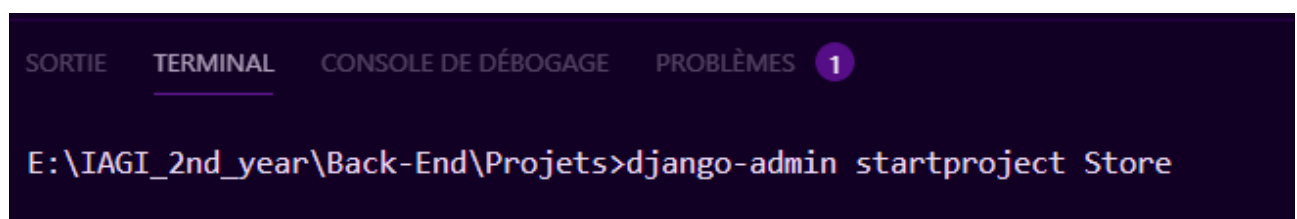
On lance maintenant notre IDE VSC (Visual Studio Code) et on ajoute un nouveau terminal :



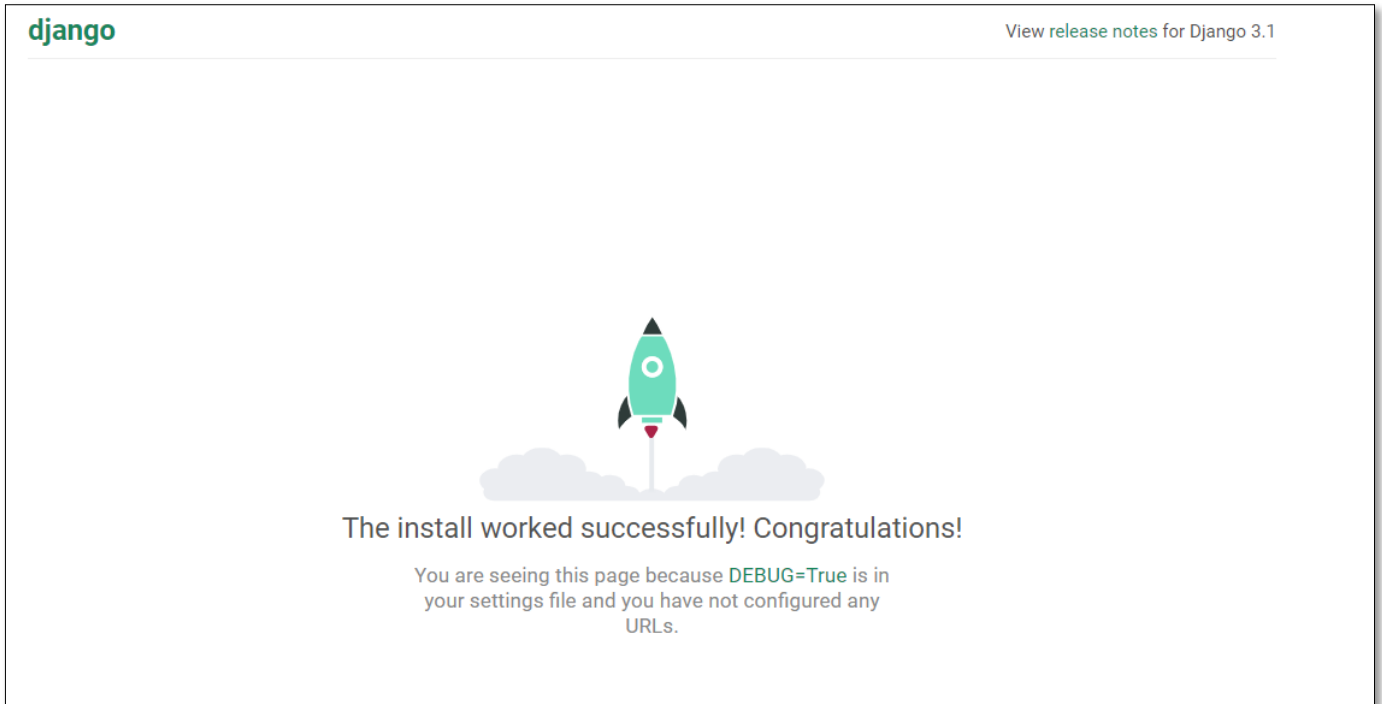
Après, on active l'environnement :

```
(base) E:\IAGI_2nd_year\Back-End\Projets>conda activate base  
(base) E:\IAGI_2nd_year\Back-End\Projets>
```

Maintenant, on crée notre projet :



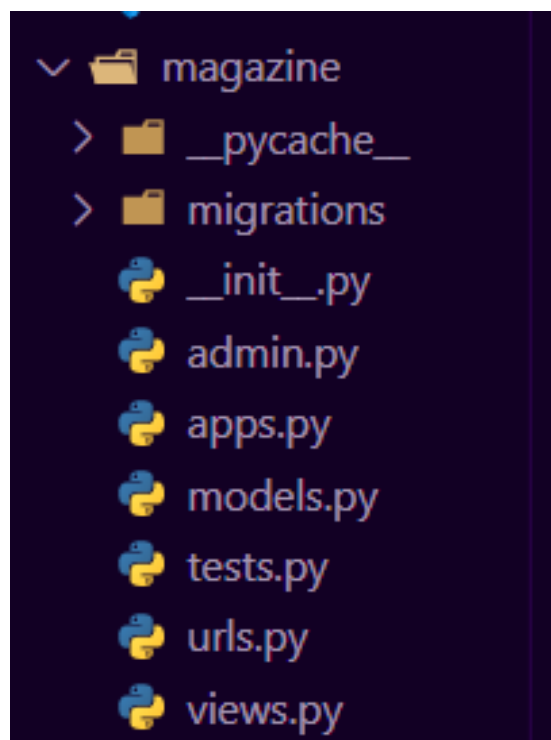
On lance le serveur avec la commande « `python manage.py runserver` » et on peut voir que tout va bien. Nous pouvons commencer le développement de notre application.



2.2. Création des Applications Django :

On crée notre application nommé « magazine » à l'aide de la commande suivante :

```
python manage.py startapp magazine
```

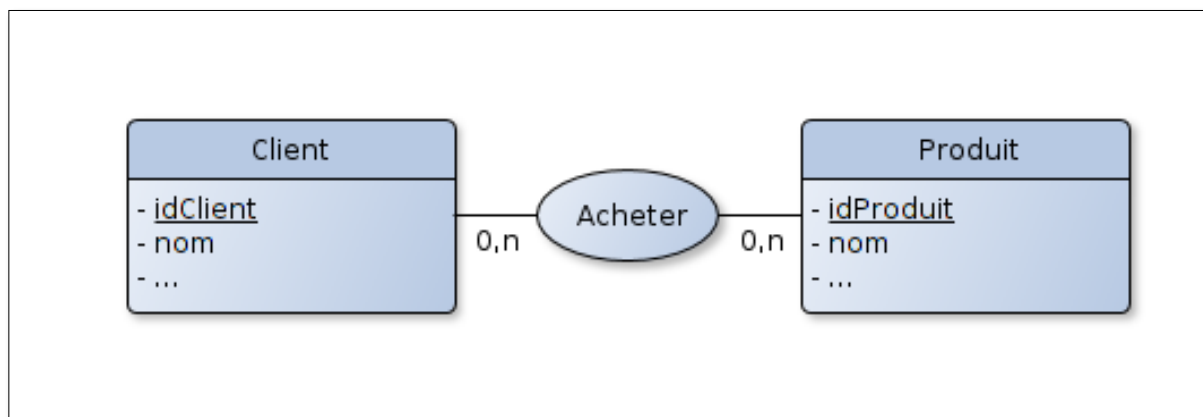


Ensuite, on l'ajoute dans la liste des 'Installed_apps' :

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'magazine',  
]
```

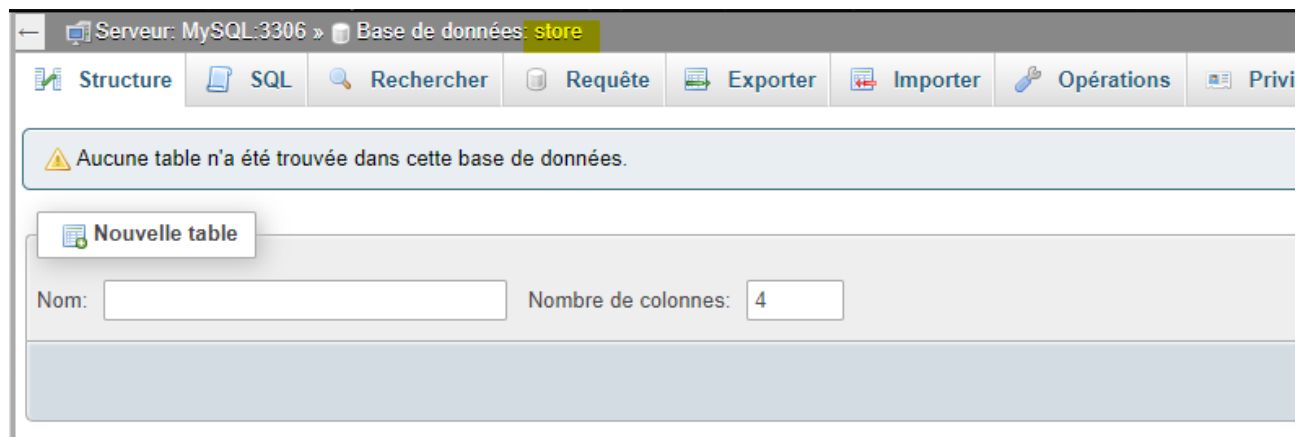
2.3. Présentation des tables (MCD) et connecter MYSQL :

Un Client a acheté zéro ou plusieurs Produits.
Un Produit a été acheté par zéro ou plusieurs Clients.



Alors, on a une relation "Many to many" entre les deux tables clients et produits.

Après avoir installé le une relation client MySQL dans notre environnement, on crée une base de données vide appelé « Store » comme suite :



On configure le fichier settings.py afin de prendre en charge cette base de données qu'on vient de créer :

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'store',
        'USER': 'root',
        'PASSWORD': '',
        'HOST': 'localhost',
        'PORT': '3306',
        'OPTIONS': {
            'init_command' : "SET sql_mode = 'STRICT_TRANS_TABLES'",
        },
    }
}
```

On Lance la migration par la commande migrate :


```
(base) E:\IAGI_2nd_year\Back-End\Projets\Store>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK

(base) E:\IAGI_2nd_year\Back-End\Projets\Store>
```

Et si on vérifie notre base de données, on trouve :

| Table | Action | Lignes | Type | Interclassement | Taille | Perte |
|---|--|--------|--------|--------------------|---------|-------|
| <input type="checkbox"/> auth_group | ★ Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | MyISAM | utf8mb4_0900_ai_ci | 4,0 kio | - |
| <input type="checkbox"/> auth_group_permissions | ★ Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | MyISAM | utf8mb4_0900_ai_ci | 1,0 kio | - |
| <input type="checkbox"/> auth_permission | ★ Parcourir Structure Rechercher Insérer Vider Supprimer | 24 | MyISAM | utf8mb4_0900_ai_ci | 7,1 kio | - |
| <input type="checkbox"/> auth_user | ★ Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | MyISAM | utf8mb4_0900_ai_ci | 4,0 kio | - |
| <input type="checkbox"/> auth_user_groups | ★ Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | MyISAM | utf8mb4_0900_ai_ci | 1,0 kio | - |
| <input type="checkbox"/> auth_user_user_permissions | ★ Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | MyISAM | utf8mb4_0900_ai_ci | 1,0 kio | - |
| <input type="checkbox"/> django_admin_log | ★ Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | MyISAM | utf8mb4_0900_ai_ci | 1,0 kio | - |
| <input type="checkbox"/> django_content_type | ★ Parcourir Structure Rechercher Insérer Vider Supprimer | 6 | MyISAM | utf8mb4_0900_ai_ci | 9,1 kio | - |
| <input type="checkbox"/> django_migrations | ★ Parcourir Structure Rechercher Insérer Vider Supprimer | 18 | MyISAM | utf8mb4_0900_ai_ci | 2,9 kio | - |
| <input type="checkbox"/> django_session | ★ Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | MyISAM | utf8mb4_0900_ai_ci | 1,0 kio | - |

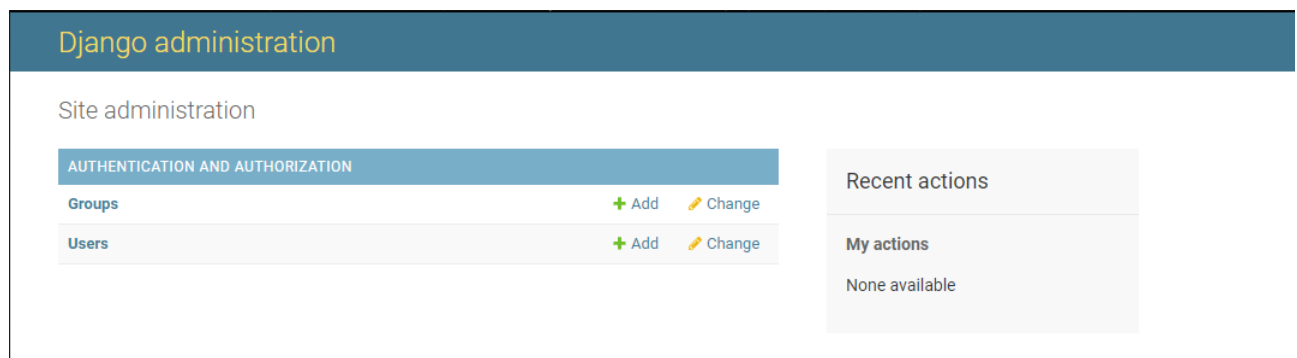
2.4. Création du Super utilisateur :

On va créer un super utilisateur (admin) qui sera chargé d'administrer les modèles qu'on va créer par la suite :

```
(base) E:\IAGI_2nd_year\Back-End\Projets\Store>python manage.py createsuperuser
Username (leave blank to use 'hp'): ilyas
Email address: ilyasirgui@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.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.

(base) E:\IAGI_2nd_year\Back-End\Projets\Store>
```

On peut accéder au panel d'administration à l'aide du chemin : localhost :8000/admin
(User Name : ilyas ----- Password : ilyas)



2.5. Implémentation des modèles :

On commence maintenant à créer les modèles nécessaires de notre application dans le fichier models.py

Les deux tables Products et Clients qui ont une relation many to many

Products

```
from django.db import models

# Create your models here.

class Products(models.Model):
    name = models.CharField(max_length=50)
    category = models.CharField(max_length=50)
    rate = models.IntegerField()
    owner = models.ForeignKey(
        'auth.User',
        related_name='Products',
        on_delete=models.CASCADE,
        null=True
    )
```

Clients

```
class Clients(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    email = models.CharField(max_length=50)
    age = models.IntegerField()
    #Many to many relationship
    Products = models.ManyToManyField(Products)
    owner = models.ForeignKey(
        'auth.User',
        related_name='Clients',
        on_delete=models.CASCADE,
        null=True
    )
```

Enregistrons maintenant les modèles dans le fichier `admin.py` de l'application 'Store

```
from django.contrib import admin
from .models import Products, Clients
# Register your models here.
admin.site.register(Clients)
admin.site.register(Products)
```

Lançons maintenant la migration :

```
(base) E:\IAGI_2nd_year\Back-End\Projets\Store>python manage.py makemigrations
Migrations for 'magazine':
  magazine\migrations\0001_initial.py
    - Create model Products
    - Create model Clients

(base) E:\IAGI_2nd_year\Back-End\Projets\Store>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, magazine, sessions
Running migrations:
  Applying magazine.0001_initial... OK

(base) E:\IAGI_2nd_year\Back-End\Projets\Store>
```

Revenons encore à notre base :

| Table | Structure | Rechercher | Insérer | Vider | Supprimer | Engine | Charset | Collation | Size |
|--|--------------|------------|---------|-------|-----------|------------------|---------------------------|-----------|-----------------|
| <input type="checkbox"/> magazine_clients | | | | | | MyISAM | utf8mb4_0900_ai_ci | | 1,0 kio |
| <input type="checkbox"/> magazine_clients_products | | | | | | MyISAM | utf8mb4_0900_ai_ci | | 1,0 kio |
| <input type="checkbox"/> magazine_products | | | | | | MyISAM | utf8mb4_0900_ai_ci | | 1,0 kio |
| 13 tables | Somme | | | | | 48 MyISAM | utf8mb4_0900_ai_ci | | 35,2 kio |

☐ Tout cocher
 Avec la sélection :

| MAGAZINE | |
|-----------|--|
| Clientss | + Add ✎ Change |
| Productss | + Add ✎ Change |

Depuis le panel d'administration on réalise quelques opérations :

Add products

Name:

Phone

Category:

Technology

Rate:

8

☐ PRODUCTS

☐ Phone

☐ Headphones

2 products

Add clients

First name:

Ali

Last name:

Alilo

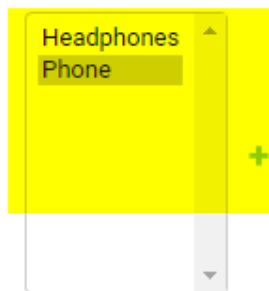
Email:

ali.ali@gmail.com

Age:

21

Products:



Hold down "Control", or "Command" on a Mac, to select more than one.

☐ CLIENTS

☐ Ali

1 clients

III. Configuration du REST_API

3.1. La sérialisation des modèles (CRUD) :

On commence par installer Django Rest Framework a l'aide de la commande suivante :

```
(base) E:\IAGI_2nd_year\Back-End\Projets\Store>pip install djangorestframework
```

Ensuite, on l'ajoute dans la liste des 'Installed_apps' :

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'magazine',  
    'rest_framework',  
]
```

On crée le fichier « serializer.py » où on va sérialiser nos modèles :

```
magazine > serializer.py > ClientsSerializer > Meta  
1  from rest_framework import serializers  
2  from .models import Products, Clients  
3  from django.contrib.auth.models import User  
4  
5  class ProductsSerializer(serializers.ModelSerializer):  
6      class Meta :  
7          model = Products  
8          fields = '__all__'  
9  
10 class ClientsSerializer(serializers.ModelSerializer):  
11     class Meta :  
12         model = Clients  
13         fields = '__all__'
```

3.2. L'authentification par Token :

Dans le fichier views.py on n'importe :

```
8 from rest_framework.authentication import TokenAuthentication
9
0 from rest_framework.permissions import IsAuthenticated
1
```

Puis on ajoute :

```
authentication_classes = [TokenAuthentication]

permission_classes = [IsAuthenticated]
```

Ensuite, on l'ajoute dans la liste des 'Installed_apps', après on lance la migration et on démarre le serveur, dans admin panel on crée notre Token.



IV. Les Manipulation CRUD

Importation :

```
from django.shortcuts import render
from rest_framework import generics, mixins
from .models import Products, Clients
from .serializer import ProductsSerializer, ClientsSerializer
from rest_framework.response import Response
from django.contrib.auth import get_user_model
from rest_framework.authentication import TokenAuthentication
from rest_framework.permissions import IsAuthenticated
```

Api Client :

```
class ClientsGenericApi(generics.GenericAPIView,
                        mixins.ListModelMixin,
                        mixins.RetrieveModelMixin,
                        mixins.CreateModelMixin,
                        mixins.UpdateModelMixin,
                        mixins.DestroyModelMixin):
    serializer_class = ClientsSerializer
    queryset=Clients.objects.all()
    lookup_field = 'id'
```


Api du Produit :

```
class ProductsGenericApi(generics.GenericAPIView,  
                        mixins.ListModelMixin,  
                        mixins.RetrieveModelMixin,  
                        mixins.CreateModelMixin,  
                        mixins.UpdateModelMixin,  
                        mixins.DestroyModelMixin):  
    serializer_class = ProductsSerializer  
    queryset=Products.objects.all()  
    lookup_field = 'id'
```

4.1. Rechercher :

On peut avoir la liste complète d'un client ou avoir un client par son id (même code pour produit)

```
1  # -----LES METHODES CRUD-----  
2  #-----#  
3      def get(self,request,id=None):  
4          if id:  
5              return self.retrieve(request)  
6          else:  
7              return self.list(request)  
8  #-----#
```

4.2. Ajouter :

L'ajout d'un client nécessite l'ajout des produits associés chose qui va être réaliser par une Loop for.

```
#-----#
def post(self,request):
    owner = request.data["owner"]
    superuser = User.objects.filter(is_superuser=True,username=owner).first()
    data = request.data
    Client = Clients.objects.create(first_name=data["first_name"],
                                   last_name = data["last_name"],
                                   email = data["email"],
                                   age = data["age"],
                                   owner=superuser)

    Client.save()

    for product in data["Products"]:
        product_obj = Products.objects.get(name=product["name"])
        Client.Products.add(product_obj)

    serializer = ClientsSerializer(Client)
    return Response(serializer.data)
#-----#
def put(self,request,id):
```

L'ajout d'un produit

```
#-----#
def post(self,request):
    owner = request.data["owner"]
    superuser = User.objects.filter(is_superuser=True,username=owner).first()
    data = request.data
    new_products =Products.objects.create(name = data["name"],
                                          category = data["category"],
                                          rate = data["rate"],
                                          owner = superuser)

    new_products.save()
    serializer = ProductsSerializer(new_products)
    return Response(serializer.data)
#-----#
```

4.3. Modifier :

La modification d'un client (même code pour produit)

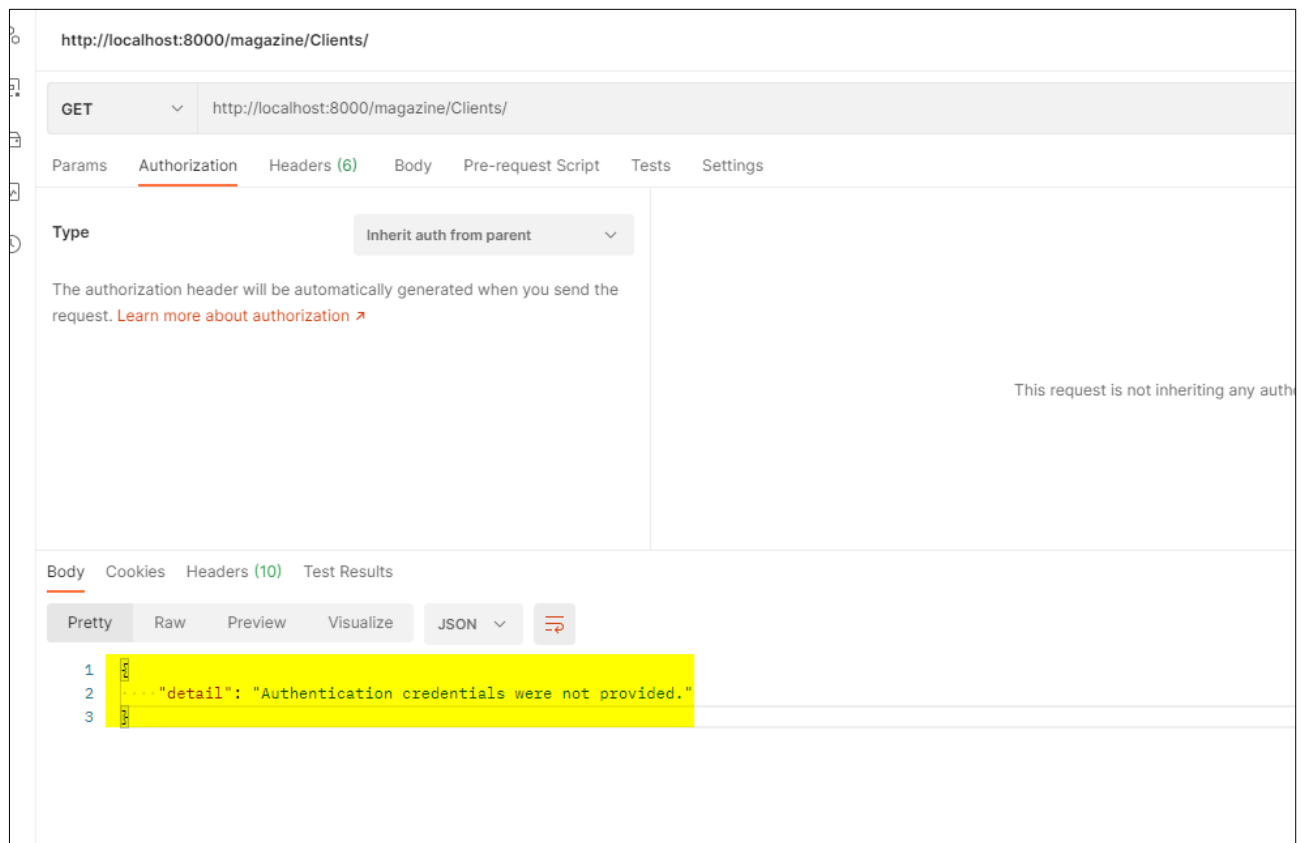
```
6 #-----#  
7     def put(self,request,id):  
8         return self.update(request,id)  
9 #-----#
```

4.4. Supprimer :

La Suppression d'un client (même code pour produit)

```
6 #-----#  
7     def put(self,request,id):  
8         return self.update(request,id)  
9 #-----#  
    return self.destroy(request,id)
```

Teste avec Postman :



Avant tous on doit fournir le Token qu'on a créé

GET
http://localhost:8000/magazine/Clients/

Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings

| | | |
|-------------------------------------|-------------------|--|
| <input checked="" type="checkbox"/> | Postman-Token ① | <calculated when request is sent> |
| <input checked="" type="checkbox"/> | Host ③ | <calculated when request is sent> |
| <input checked="" type="checkbox"/> | User-Agent ③ | PostmanRuntime/7.26.10 |
| <input checked="" type="checkbox"/> | Accept ③ | */* |
| <input checked="" type="checkbox"/> | Accept-Encoding ① | gzip, deflate, br |
| <input checked="" type="checkbox"/> | Connection ① | keep-alive |
| <input checked="" type="checkbox"/> | Authorization | token 48c5e7491d69176f4979a27195815b4be0b06691 |
| | Key | Value |

Body
Cookies
Headers (9)
Test Results

Pretty
Raw
Preview
Visualize
JSON

```

1  {
2    "id": 1,
3    "first_name": "Ali",
4    "last_name": "Aliilo",
5    "email": "ali.ali@gmail.com",
6    "age": 21,
7    "owner": 1,
8    "Products": [
9      2
10     ]
11  },
12  {
13    "id": 2,
14    "first_name": "khalid",
15    "last_name": "tosani",
16    "email": "to.khalid@hotmail.com",
17    "age": 50,
18    "owner": 1,
19    "Products": [
20      1
21     ]
22  }
23  ]

```

POST

http://localhost:8000/magazine/Clients/

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

5

6

7

8

9

10

11

12

.....{

....."first_name": "Imane",

....."last_name": "Amira",

....."email": "imane@gmail.com",

....."age": 11,

....."owner": 1,

....."Products": [

.....{

....."name": "Clavier"

.....}

.....]

.....}

Body

Cookies

Headers (9)

Test Results

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

8

9

10

11

.....{

....."id": 26,

....."first_name": "Imane",

....."last_name": "Amira",

....."email": "imane@gmail.com",

....."age": 11,

....."owner": null,

....."Products": [

.....5

.....]

.....}

PUT

http://localhost:8000/magazine/Clients/26/

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

....{

2

....."first_name": "Inass",

3

....."last_name": "Amira",

4

....."email": "Amira@gmail.com",

5

....."age": 11,

6

....."owner": null,

7

....."Products": []

8

.....5

9

.....]

10

....}

Body

Cookies

Headers (9)

Test Results

Pretty

Raw

Preview

Visualize

JSON

1

5

2

...."id": 26,

3

...."first_name": "Inass",

4

...."last_name": "Amira",

5

...."email": "Amira@gmail.com",

6

...."age": 11,

7

...."owner": null,

8

...."Products": [

9

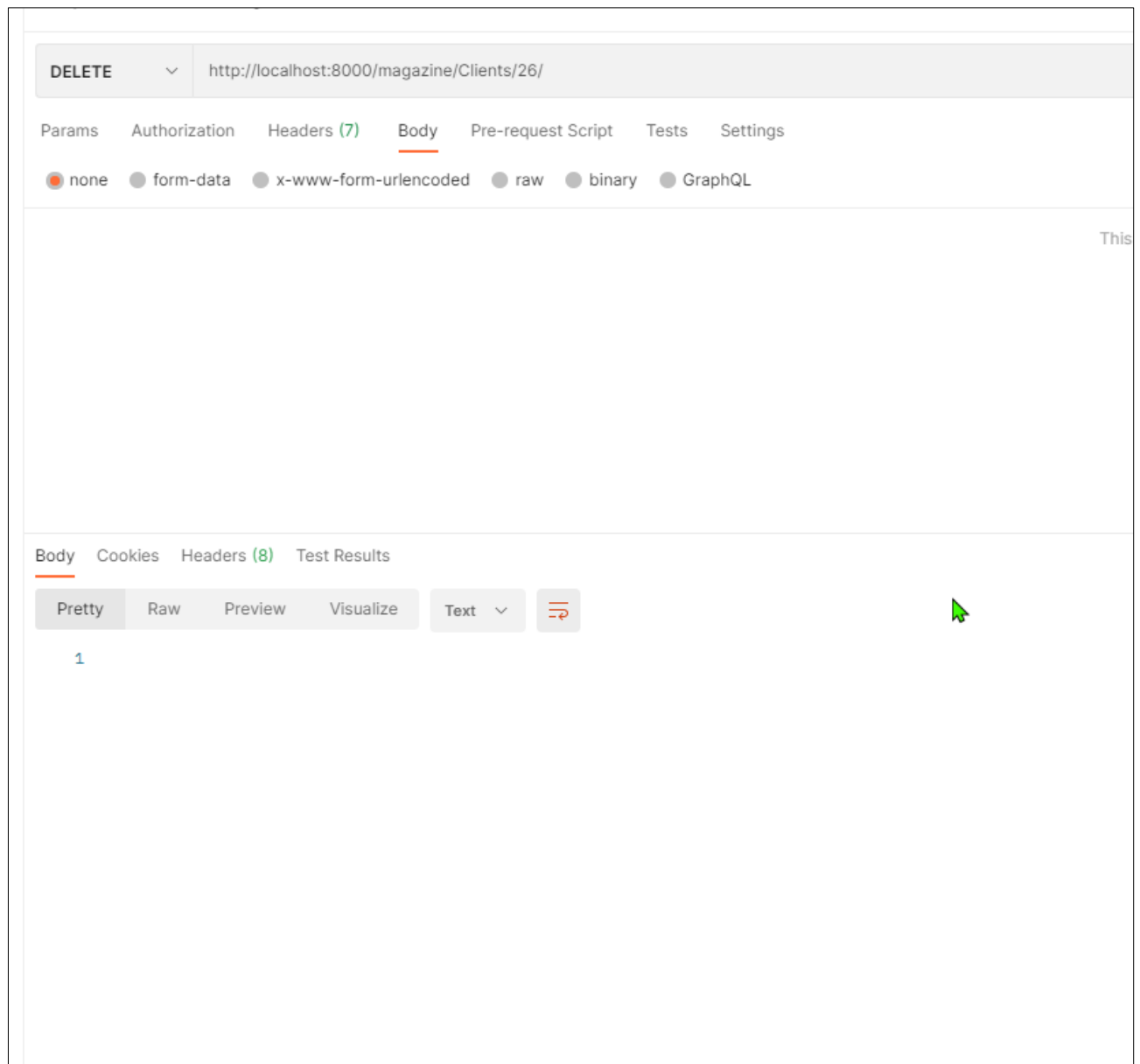
.....5

10

....]

11

5



V. Conclusion

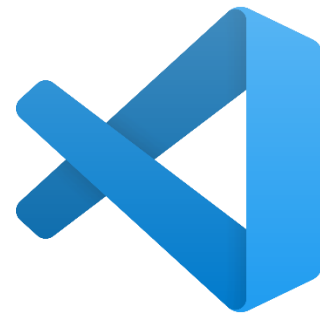
A l'aide de Django, on a réalisé une application web crud qui comporte deux modèles le modèle client et le modèle produits et puis on a utilisé post man afin de tester cette application

Pour le côté front end, vu qu'on n'a pas encore étudié une solution front comme Angular ou React on a utilisé l'outil post man, dès qu'on va entamer une solution, ça sera utile de le connecter avec notre application Django afin de créer une application web complète

Les outils utilisés dans ce projet :



Anaconda



Visual Studio Code



Wamp Server



PostMan