PROYECTO: MOTOS - API

OBJETIVO: Crear una aplicación Backend que provea datos por medio de API Web.

1. Ingresamos a la carpeta donde hemos creado el ambiente virtual de Python

```
D:\>cd proyectos_python

D:\Proyectos_Python>dir
 El volumen de la unidad D es Nuevo vol
 El número de serie del volumen es: 30B7-184D

 Directorio de D:\Proyectos_Python

01/11/2022  03:22 p. m.    <DIR>          .
27/10/2022  02:50 p. m.    <DIR>          venv-py3110
               0 archivos              0 bytes
               2 dirs  102.936.891.392 bytes libres

D:\Proyectos_Python>cd venv-py3110

D:\Proyectos_Python\venv-py3110>scripts\activate

(venv-py3110) D:\Proyectos_Python\venv-py3110>
```

2. Verificamos la versión de Django que tenemos instalada

```
(venv-py3110) D:\Proyectos_Python\venv-py3110>python -m django --version
4.1.3

(venv-py3110) D:\Proyectos_Python\venv-py3110>
```

De ser necesario, instalamos Django con el comando

```
(venv-py3110) D:\Proyectos_Python\venv-py3110>python -m pip install django
Collecting django
  Downloading Django-4.1.3-py3-none-any.whl (8.1 MB)
     ---------------------------------------- 8.1/8.1 MB 8.0 MB/s eta 0:00:00
Collecting asgiref<4,>=3.5.2
  Using cached asgiref-3.5.2-py3-none-any.whl (22 kB)
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.3-py3-none-any.whl (42 kB)
Collecting tzdata
  Downloading tzdata-2022.6-py2.py3-none-any.whl (338 kB)
     ---------------------------------------- 338.8/338.8 kB 7.0 MB/s eta 0:00:00
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.5.2 django-4.1.3 sqlparse-0.4.3 tzdata-2022.6
```

3. En Django, debemos hacer la diferenciación entre proyecto y aplicación: un proyecto es una colección de archivos de configuración y aplicaciones, mientras que la aplicación es un desarrollo web escrito para realizar la lógica empresarial.

   Por lo anterior, primero creamos un proyecto y después una aplicación.

   a. Creamos un proyecto Django llamado app_motos

```
(venv-py3110) D:\Proyectos_Python\venv-py3110>django-admin startproject app_motos

(venv-py3110) D:\Proyectos_Python\venv-py3110>dir
 El volumen de la unidad D es Nuevo vol
 El número de serie del volumen es: 30B7-184D

 Directorio de D:\Proyectos_Python\venv-py3110

23/11/2022  09:06 a. m.    <DIR>          .
01/11/2022  03:22 p. m.    <DIR>          ..
09/11/2022  03:29 p. m.    <DIR>          app_blog
27/10/2022  03:12 p. m.    <DIR>          app_holamundo
23/11/2022  09:06 a. m.    <DIR>          app_motos
17/11/2022  08:57 p. m.    <DIR>          app_newspaper
29/10/2022  08:52 p. m.             1.996 comandos.rtf
27/10/2022  02:41 p. m.    <DIR>          Include
27/10/2022  02:41 p. m.    <DIR>          Lib
01/11/2022  06:10 p. m.               683 manage.py
22/11/2022  07:23 p. m.    <DIR>          motos
01/11/2022  07:47 p. m.    <DIR>          post
27/10/2022  02:41 p. m.               225 pyvenv.cfg
27/10/2022  02:46 p. m.    <DIR>          Scripts
               3 archivos          2.904 bytes
              11 dirs  98.031.259.648 bytes libres

(venv-py3110) D:\Proyectos_Python\venv-py3110>
```

b. Ingresamos a la carpeta del proyecto recién creado

```
(venv-py3110) D:\Proyectos_Python\venv-py3110>cd app_motos

(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>dir
 El volumen de la unidad D es Nuevo vol
 El número de serie del volumen es: 30B7-184D

 Directorio de D:\Proyectos_Python\venv-py3110\app_motos

23/11/2022  09:06 a. m.    <DIR>          .
23/11/2022  09:06 a. m.    <DIR>          ..
23/11/2022  09:06 a. m.    <DIR>          app_motos
23/11/2022  09:06 a. m.               687 manage.py
               1 archivos            687 bytes
               3 dirs  98.031.259.648 bytes libres

(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>
```

c. Creamos una aplicación llamada motos_api

```
(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>python manage.py startapp motos_api

(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>dir
 El volumen de la unidad D es Nuevo vol
 El número de serie del volumen es: 30B7-184D

 Directorio de D:\Proyectos_Python\venv-py3110\app_motos

23/11/2022  09:10 a. m.    <DIR>          .
23/11/2022  09:06 a. m.    <DIR>          ..
23/11/2022  09:10 a. m.    <DIR>          app_motos
23/11/2022  09:06 a. m.               687 manage.py
23/11/2022  09:10 a. m.    <DIR>          motos_api
               1 archivos            687 bytes
               4 dirs  98.031.251.456 bytes libres

(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>
```

4. En nuestro proyecto vamos a requerir instalar dos librerías:
   a. ***Django REST Framework (DRF)***: es un conjunto de herramientas potente y flexible para crear ***API's web***; uno de sus beneficios es que facilita la serialización (proceso que permite una transferencia de datos sin problemas). DRF adopta implementaciones como vistas basadas en clases, formularios, validador de modelos, QuerySet y más.

      python -m pip install djangorestframework

```
(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>python -m pip install djangorestframework
Requirement already satisfied: djangorestframework in c:\program files\python3110\lib\site-packages (3.1
4.0)
Requirement already satisfied: django>=3.0 in c:\program files\python3110\lib\site-packages (from django
restframework) (4.1.2)
Requirement already satisfied: pytz in c:\program files\python3110\lib\site-packages (from djangorestfra
mework) (2022.5)
Requirement already satisfied: asgiref<4,>=3.5.2 in c:\program files\python3110\lib\site-packages (from
django>=3.0->djangorestframework) (3.5.2)
Requirement already satisfied: sqlparse>=0.2.2 in c:\program files\python3110\lib\site-packages (from dj
ango>=3.0->djangorestframework) (0.4.3)
Requirement already satisfied: tzdata in c:\program files\python3110\lib\site-packages (from django>=3.0
->djangorestframework) (2022.5)

(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>
```

   b. ***Django-cors-headers***: Las aplicaciones desarrolladas en Django pueden necesitar interactuar con otras aplicaciones alojadas en diferentes dominios (o incluso en diferentes puertos); Para que estas solicitudes tengan éxito y no se presenten problemas de seguridad, deberá utilizar el uso compartido de recursos de origen cruzado (***CORS - Cross-Origin Resource Sharing***) en su servidor.

      python -m pip install django-cors-headers

```
(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>python -m pip install django-cors-headers
Requirement already satisfied: django-cors-headers in c:\program files\python3110\lib\site-packages (3.13.0)
Requirement already satisfied: Django>=3.2 in c:\program files\python3110\lib\site-packages (from django-cors-headers) (
4.1.2)
Requirement already satisfied: asgiref<4,>=3.5.2 in c:\program files\python3110\lib\site-packages (from Django>=3.2->dja
ngo-cors-headers) (3.5.2)
Requirement already satisfied: sqlparse>=0.2.2 in c:\program files\python3110\lib\site-packages (from Django>=3.2->djang
o-cors-headers) (0.4.3)
Requirement already satisfied: tzdata in c:\program files\python3110\lib\site-packages (from Django>=3.2->django-cors-he
aders) (2022.5)

(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>
```

5. Abrimos el proyecto con VS Code

```
(venv-py3110) D:\Proyectos_Python\venv-py3110\app_blog>
(venv-py3110) D:\Proyectos_Python\venv-py3110\app_blog>code .

(venv-py3110) D:\Proyectos_Python\venv-py3110\app_blog>
```

6. Modificamos el archivo ***app_motos/settings.py*** para "alertar" a Django de la nueva ***APP motos_api*** y configurar las librerías instaladas ***rest_framework*** y ***corsheader***

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'corsheaders',
    'motos_api.apps.MotosApiConfig',
]
```

```
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

REST_FRAMEWORK = {
    "DEFAULT_PERMISSION_CLASSES" : [
        "rest_framework.permissions.AllowAny" ,
    ],
}

CORS_ALLOWED_ORIGINS = (
 "http://localhost:3004" , # Dominio del componente Front-End
 "http://localhost:8000" , # Dominio del Componente Back-End
)

CSRF_TRUSTED_ORIGINS = ["http://localhost:3004"] #Dominio del componente Front-End

APPEND_SLASH = False
```

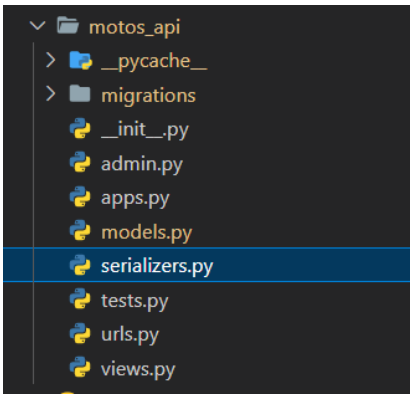
7. Para almacenar los productos, en nuestro caso, motos, modificamos el **archivo motos_api/models.py** para que cree una **tabla** llamada **Moto** con los campos **id(llave primaria), reference, trademark, model, Price, image y supplier.**

```
motos_api > models.py > ...
1   from django.db import models
2
3   # Create your models here.
4   class Moto(models.Model):
5       id = models.AutoField(primary_key = True)
6       reference = models.CharField(max_length = 30, blank = False, null = False)
7       trademark = models.CharField(max_length = 30, blank = False, null = False)
8       model = models.CharField(max_length = 4, blank = False, null = False)
9       price = models.IntegerField()
10      image = models.CharField(max_length = 150, blank = False, null = False)
11      supplier = models.CharField(max_length = 60, blank = False, null = False)
12
13      def __str__(self):
14          return self.reference
```

8. Necesitamos crear una clase ***ModelAdmin*** para representar el modelo en la interfaz de administración; por lo anterior, modificamos el archivo ***motos_api/admin.py***.

```python
# motos_api/admin.py
from django.contrib import admin
from .models import Moto

# Register your models here.
class MotoAdmin(admin.ModelAdmin):
    list_display = (
        "reference",
        "trademark",
        "model",
        "price",
        "image",
        "supplier",
    )

admin.site.register(Moto, MotoAdmin)
```

9. En la carpeta ***app_motos/motos_api***, creamos los archivos ***serializers.py y urls.py***



10. Creamos un archivo de migraciones con el comando ***makemigrations***

```
(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>python manage.py makemigrations motos_api
Migrations for 'motos_api':
  motos_api\migrations\0001_initial.py
    - Create model Moto

(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>
```

11. Construimos la base de datos real con el comando ***migrate*** que ejecuta las instrucciones del archivo de migraciones

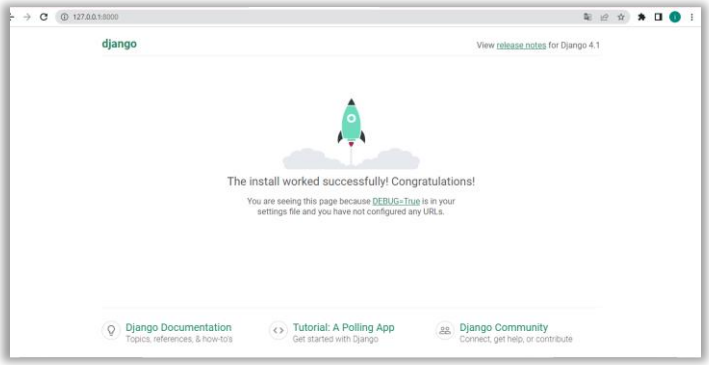
```
(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, motos_api, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... 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 motos_api.0001_initial... OK
  Applying sessions.0001_initial... OK

(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>
```

12. Creamos un superusuario que pueda iniciar sesión en la base de datos SQL Lite . En la consola de la línea de comandos, escribimos ***python manage.py createsuperuser*** y responder a las solicitudes de nombre de usuario, correo electrónico y contraseña.

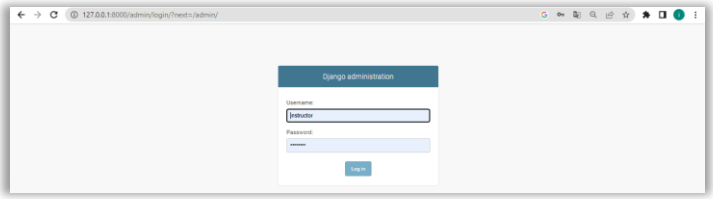
```
(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>python manage.py createsuperuser
Username (leave blank to use 'jocastron'): instructor
Email address: instructor.sena.cba@gmail.com
Password:
Password (again):
Superuser created successfully.

(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>
```

13. Iniciamos el servidor Django con el comando ***python manage.py runserver***.

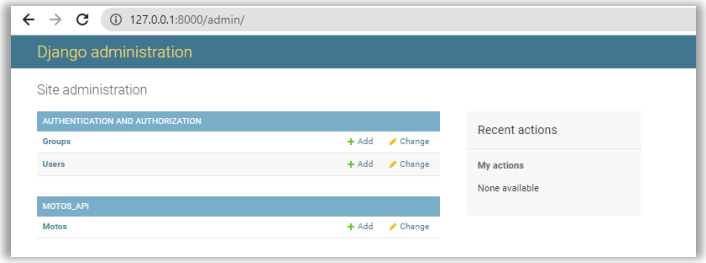
    a. En el navegador web ir a http://127.0.0.1:8000. Se verá la página de bienvenida

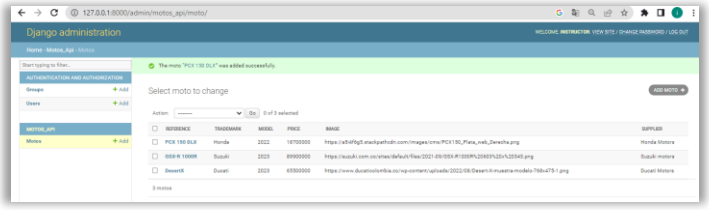


    b. En el navegador web ir a http://127.0.0.1:8000/admin/. Se verá la pantalla de inicio de sesión del gestor SQL Lite; iniciamos sesión con el ***superuser*** creado



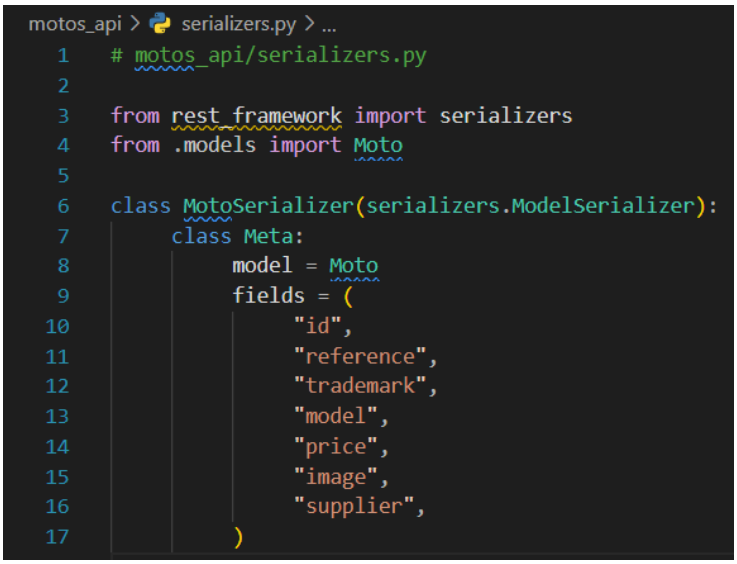
    c. En la tabla Motos, agregamos tres (3) registros





14. Django REST Framework utiliza la clase ***ModelSerializer*** para ***convertir*** cualquier ***objeto*** de tipo ***Model a*** un ***objeto JSON serializado***. Modificamos el archivo ***motos_api/serializers.py*** para serializar los datos del ***modelo MOTO***.

    ModelSerializer: La clase proporciona un acceso directo que le permite crear automáticamente una clase ***Serializer*** con campos que corresponden a los campos del Modelo. Las principales características de la clase ***ModelSerializer*** son: 1. Generará automáticamente un conjunto de campos para usted, según el modelo. 2. Generará automáticamente validadores para el serializador, como validadores únicos juntos. 3. Incluye implementaciones predeterminadas simples de ***.create()*** y ***.update()***. De forma predeterminada, todos los campos del modelo en la clase se asignarán a los campos del serializador correspondiente.



CREANDO API views en Django

Para poder responder a las ***peticiones HTTP*** vamos a crear ***dos views***:

- ***List View***: Esta clase se accesa a través del ***endpoint motos/api/*** y define dos métodos; el primer método responde a la ***petición GET*** enviando el listado de registros almacenados en la base de datos y el segundo método responde a la ***petición POST*** almacenando registros en la base de datos.
- ***Detail View***: Esta clase se accesa a través del ***endpoint motos/api/<int:moto_id>*** y define tres ***métodos HTTP***: ***GET*** (edita un registro), ***PUT*** (actualiza un registro), ***DELETE*** (borra un registro).

15. Creamos la API view Class MotoListApiView

```python
# motos_api/views.py

from django.shortcuts import render
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from rest_framework import permissions
from .models import Moto
from .serializers import MotoSerializer

# Create your views here.
# First endpoint view; HTTP methods: GET-All Records POST-New Record

class MotoListApiView(APIView):

    # 1. Lista todos los registros
    def get(self, request, *args, **kwargs):
        '''
        List all the moto items for given requested user
        '''
        motos = Moto.objects
        serializer = MotoSerializer(motos, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)

    # 2. Crea un nuevo registro
    def post(self, request, *args, **kwargs):
        '''
        Create the Moto with given moto data
        '''
        data = {
            'trademark': request.data.get('trademark'),
            'model': request.data.get('model'),
            'reference': request.data.get('reference'),
            'price': request.data.get('price'),
            'image': request.data.get('image'),
            'supplier': request.data.get('supplier'),
        }
        serializer = MotoSerializer(data=data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)

        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

16. Modificamos el archivo ***motos_api/urls.py*** para incluir el ***endpoint*** de la ***view class MotoListApiView***

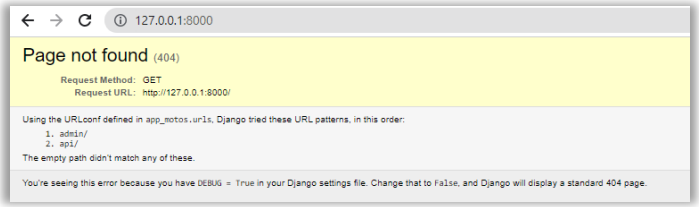
```python
# motos_api/urls.py

from django.urls import path
from .views import MotoListApiView

urlpatterns = [
    path('', MotoListApiView.as_view(), name="Moto_list"),
]
```

17. Modificamos el archivo ***app_motos/urls.py*** para incluir las rutas de la aplicación ***motos_api***

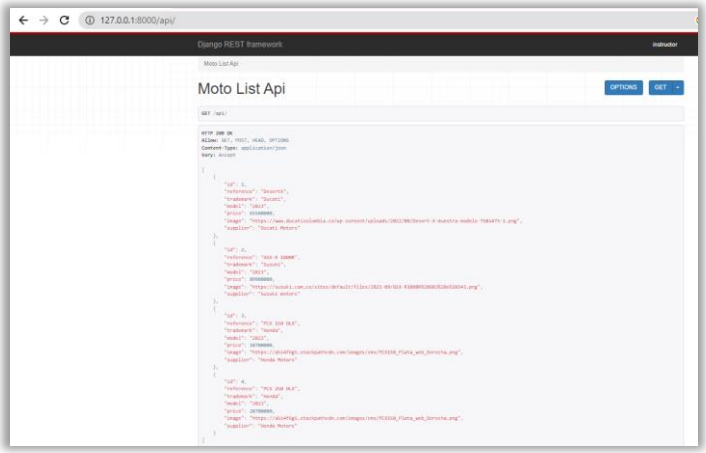
```python
from django.contrib import admin
from django.urls import path, include
from motos_api import urls as motos_urls

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include(motos_urls)),
]
```

18. Iniciamos el servidor Django con el comando ***python manage.py runserver***.

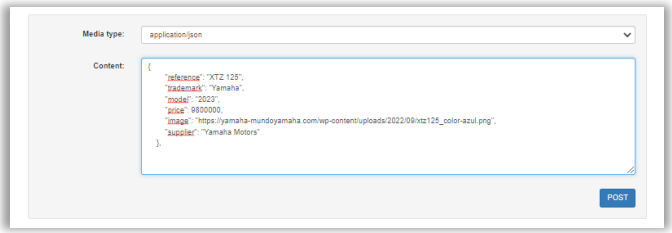
a. En el navegador web ir a ***http://127.0.0.1:8000***. No se podrá desplegar la pagina de Bienvenida debido a que ***URLConf*** no definió el ***endpoint*** correspondiente; en su lugar despliega las URLS configuradas

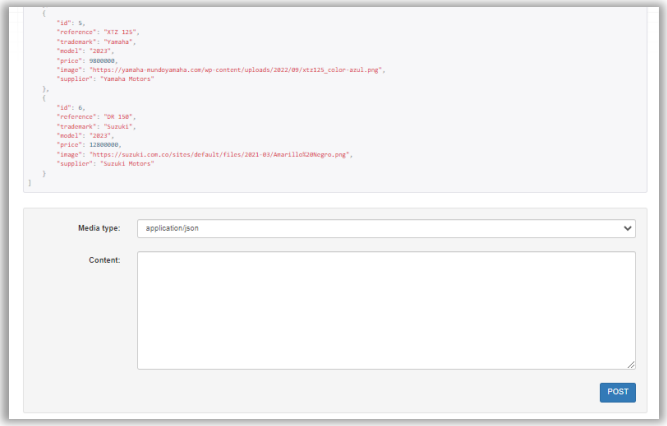


b. En el navegador web ir a ***http://127.0.0.1:8000/api/.*** Esta URL genera una petición HTTP GET que responde la clase MotoListApiView; por esa razón se despliega en formato JSON los datos contenidos en la base de datos.



c. A través de esta misma URL ***http://127.0.0.1:8000/api/*** se puede enviar una ***petición POST*** con un registro en ***formato JSON*** que será grabado por la ***clase MotoListApiView.***





19. Creamos la API view Class MotoListApiView

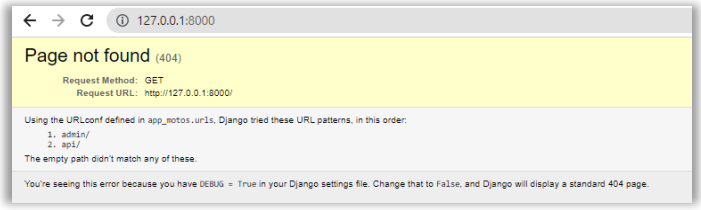
```python
# Second endpoint view. HTTP methods:   GET-Read Record
#                                       PUT-Update Record
#                                       DELETE-Delete Record
class MotoDetailApiView(APIView):

    # 1. Método auxiliar para obtener el objeto con moto_id dado
    def get_object(self, moto_id):
            try:
                return Moto.objects.get(id=moto_id)
            except Moto.DoesNotExist:
                return None

    # 2. Recupera el elemento Moto con moto_id dado
    def get(self, request, moto_id, *args, **kwargs):
        moto_instance = self.get_object(moto_id)
        if not moto_instance:
            return Response(
                {"res": "Object with moto id does not exists"},
                status=status.HTTP_400_BAD_REQUEST
            )

        serializer = MotoSerializer(moto_instance)
        return Response(serializer.data, status=status.HTTP_200_OK)

    # 3. Actualiza el elemento Moto con moto_id dado, si existe
    def put(self, request, moto_id, *args, **kwargs):
        moto_instance = self.get_object(moto_id)
        if not moto_instance:
            return Response(
                {"res": "Object with moto id does not exists"},
                status=status.HTTP_400_BAD_REQUEST
            )
        data = {
            'trademark': request.data.get('trademark'),
            'model': request.data.get('model'),
            'reference': request.data.get('reference'),
            'price': request.data.get('price'),
            'image': request.data.get('image'),
            'supplier': request.data.get('supplier'),
        }
        serializer = MotoSerializer(
                instance = moto_instance,
                data=data,
                partial = True
        )
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_200_OK)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    # 4. Elimina el elemento Moto con moto_id dado, si existe
    def delete(self, request, moto_id, *args, **kwargs):
        moto_instance = self.get_object(moto_id)
        if not moto_instance:
            return Response(
                {"res": "Object with moto id does not exists"},
                status=status.HTTP_400_BAD_REQUEST
            )
        moto_instance.delete()
        return Response(
            {"res": "Object deleted!"},
            status=status.HTTP_200_OK
        )
```

20. Modificamos el archivo ***motos_api/urls.py*** para incluir el ***endpoint*** de la ***view class MotoListApiView***

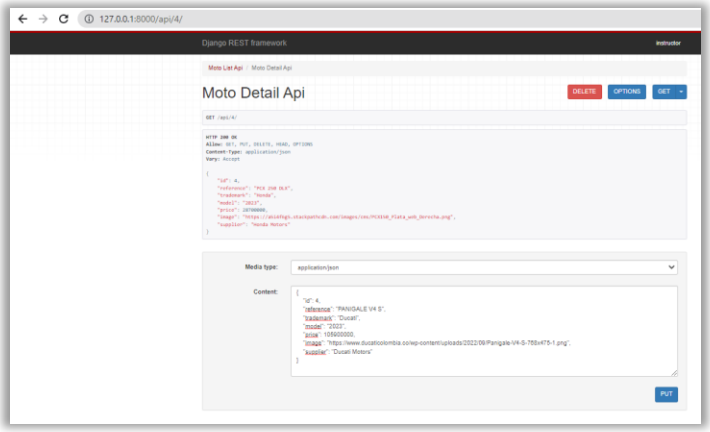
```python
motos_api > urls.py > ...
# motos_api/urls.py

from django.urls import path
from .views import MotoListApiView, MotoDetailApiView

urlpatterns = [
    path('', MotoListApiView.as_view(), name="Moto_list"),
    path('<int:moto_id>/', MotoDetailApiView.as_view(), name="Moto_detail"),
]
```

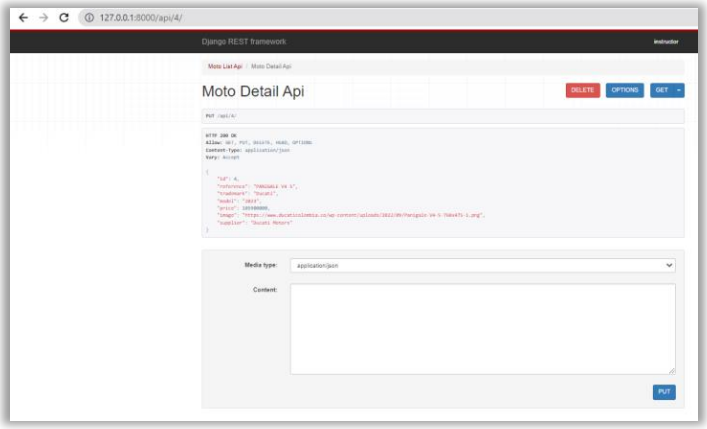
21. Iniciamos el servidor Django con el comando ***python manage.py runserver***.

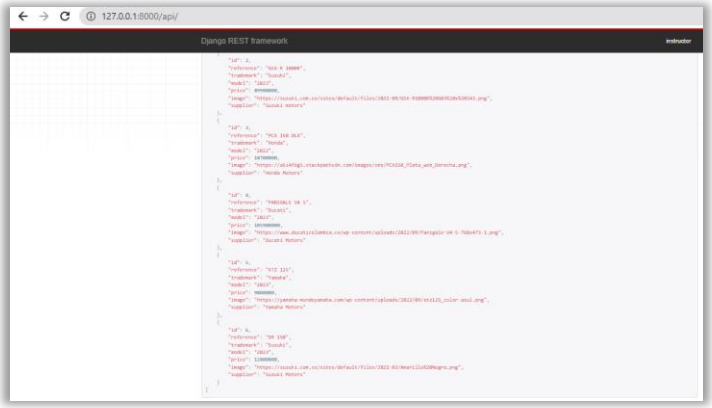
    a. En el navegador web ir a ***http://127.0.0.1:8000***.

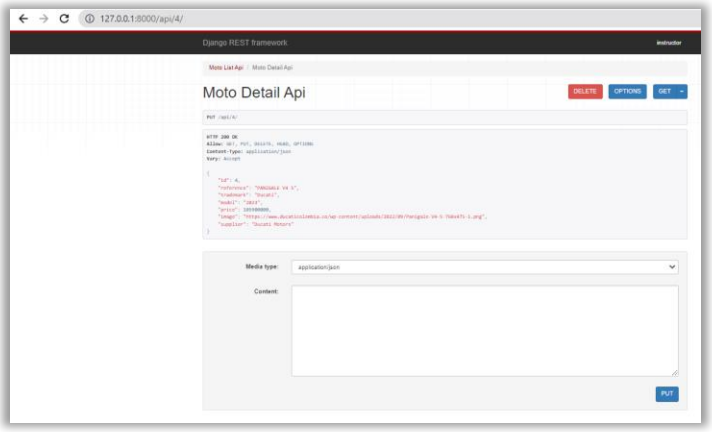


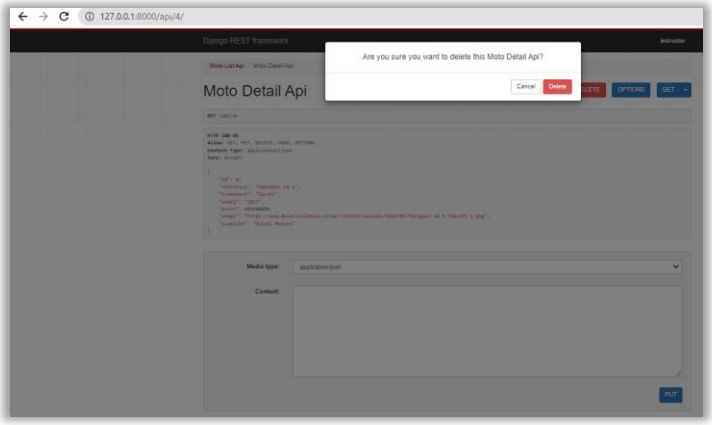
b. En el navegador web ir a ***http://127.0.0.1:8000/api/5.*** Esta URL genera una petición HTTP GET que responde la clase MotoListApiView; por esa razón se despliega en formato JSON los datos contenidos en la base de datos.

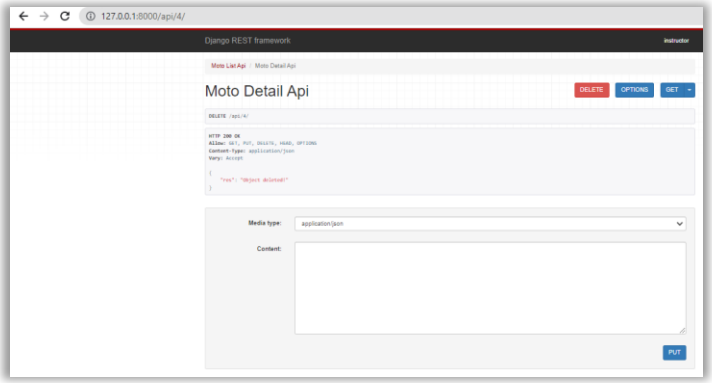


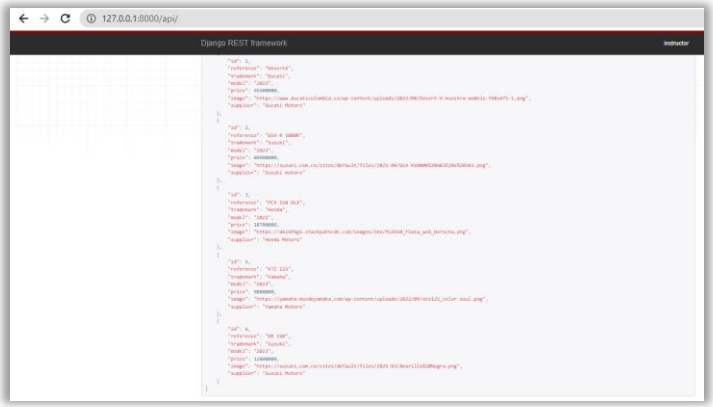




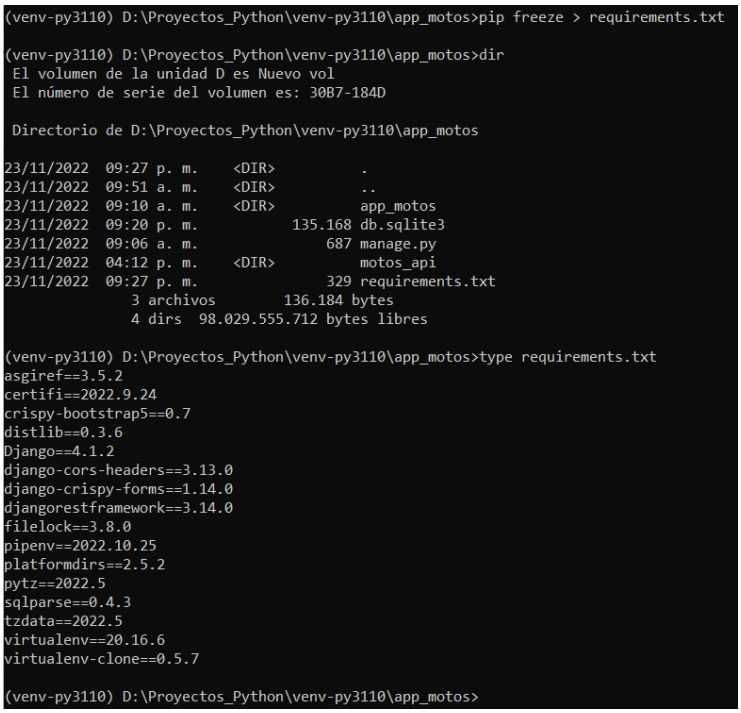








22. Para poder desplegar nuestro proyecto debemos crear un archivo texto ***requirements.txt*** en el directorio ***/app_motos***; ejecutamos el comando ***pip freeze > requeriments.txt***



```
(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>pip freeze > requirements.txt

(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>dir
 El volumen de la unidad D es Nuevo vol
 El número de serie del volumen es: 30B7-184D

 Directorio de D:\Proyectos_Python\venv-py3110\app_motos

23/11/2022  09:27 p. m.    <DIR>          .
23/11/2022  09:51 a. m.    <DIR>          ..
23/11/2022  09:10 a. m.    <DIR>          app_motos
23/11/2022  09:20 p. m.           135.168 db.sqlite3
23/11/2022  09:06 a. m.               687 manage.py
23/11/2022  04:12 p. m.    <DIR>          motos_api
23/11/2022  09:27 p. m.               329 requirements.txt
               3 archivos        136.184 bytes
               4 dirs  98.029.555.712 bytes libres

(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>type requirements.txt
asgiref==3.5.2
certifi==2022.9.24
crispy-bootstrap5==0.7
distlib==0.3.6
Django==4.1.2
django-cors-headers==3.13.0
django-crispy-forms==1.14.0
djangorestframework==3.14.0
filelock==3.8.0
pipenv==2022.10.25
platformdirs==2.5.2
pytz==2022.5
sqlparse==0.4.3
tzdata==2022.5
virtualenv==20.16.6
virtualenv-clone==0.5.7

(venv-py3110) D:\Proyectos_Python\venv-py3110\app_motos>
```

Nota: para instalar las dependencias ***a partir*** del archivo ***requirements.txt***, ejecutamos el comando ***pip install -r requirements.txt***