

# **SERVICIOS WEB USANDO DJANGO**

Presentado por:

Luis Poot

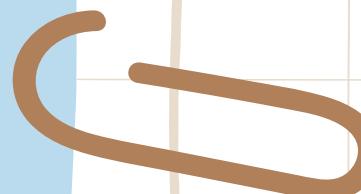
Christopher Rajon

Ricardo Ayala

# HISTORIA

Django fue inventado por Lawrence Journal-World en 2003, para cumplir con el corto plazos en el periódico y al mismo tiempo satisfacer las demandas de Desarrolladores web experimentados.

La liberación inicial al público fue en julio de 2005.



# ¿CÓMO FUNCIONA?

Django sigue el patrón de diseño MVT (Model View Template).

- Modelo: los datos que desea presentar, generalmente datos de una base de datos.
- Ver: un controlador de solicitudes que devuelve la plantilla y el contenido relevantes, según la solicitud del usuario.
- Plantilla - Un archivo de texto (como un archivo HTML) que contiene el diseño de la página web, con lógica sobre cómo mostrar los datos.

# MODELO

El modelo proporciona datos de la base de datos.

Los datos se entregan como un mapeo relacional de objetos (ORM), que es una técnica diseñada para facilitar el trabajo con bases de datos.

La forma más común de extraer datos de una base de datos es SQL. Un problema con SQL es que tienes que tener una comprensión bastante buena de la estructura de la base de datos para poder trabajar con él.

Django, con ORM, facilita la comunicación con la base de datos, sin tener que escribir sentencias SQL complejas.

Los modelos suelen estar ubicados en un archivo llamado **models.py**

# VISTA

Una vista es una función o método que toma solicitudes http como argumentos, importa los modelos relevantes y descubre qué datos enviar a la plantilla, y devuelve el resultado final.

Las vistas suelen ubicarse en un archivo denominado `.views.py`

# PLANTILLA

Una plantilla es un archivo en el que se describe cómo se debe representar el resultado.

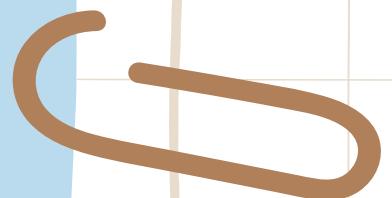
Las plantillas son a menudo archivos .html, con código HTML que describe el diseño de una página web. Pero también puede estar en otros formatos de archivo para presentar otros resultados, pero nos concentraremos en .html archivos.

Django usa HTML estándar para describir el diseño, pero usa etiquetas Django para agregar lógica:

```
<h1>My Homepage</h1>
```

```
<p>My name is {{ firstname }}.</p>
```

Las plantillas de una aplicación se encuentran en una carpeta denominada **templates**



# DIRECCIONES URL

Django también proporciona una forma de navegar por las diferentes páginas de un sitio web.

Cuando un usuario solicita una URL, Django decide a qué vista la enviará.

Esto se hace en un archivo llamado `.urls.py`

# PLANTILLA

Cuando haya instalado Django y creado su primera aplicación web Django, y el navegador solicita la URL, esto es básicamente lo que sucede:

1. Django recibe la URL, comprueba el archivo y llama a la vista que coincide con la URL **urls.py**
2. La vista, ubicada en , comprueba los modelos relevantes.**views.py**
3. Los modelos se importan desde el archivo **models.py**
4. A continuación, la vista envía los datos a una plantilla especificada en la carpeta **template**
5. La plantilla contiene etiquetas HTML y Django, y con los datos que devuelve contenido HTML terminado de vuelta al navegador.

# **REALIZAR UN SERVICIO CON PYTHON USANDO DJANGO CON LA ARQUITECTURA REST**

**<https://github.com/LEPP2001/app-django-rest.git>**

# INTRODUCCIÓN

Django Rest Framework es una aplicación Django que permite construir proyectos software bajo la arquitectura REST, incluye gran cantidad de código para reutilizar (Views, Resources, etc.) y una interfaz administrativa desde la cual es posible realizar pruebas sobre las operaciones HTTP como lo son: POST y GET.

# PREPARACIÓN

## Abrir el cmd y situarse en la ruta del proyecto

```
1.cd C:\Users\Luis Poot\Desktop\app-django-rest
```

## Crear un entorno virtual

```
1.python -m venv virtual  
2.virtual\Scripts\activate
```

## Instalar los paquetes necesarios

```
1.pip install django  
2.pip install djangorestframework
```

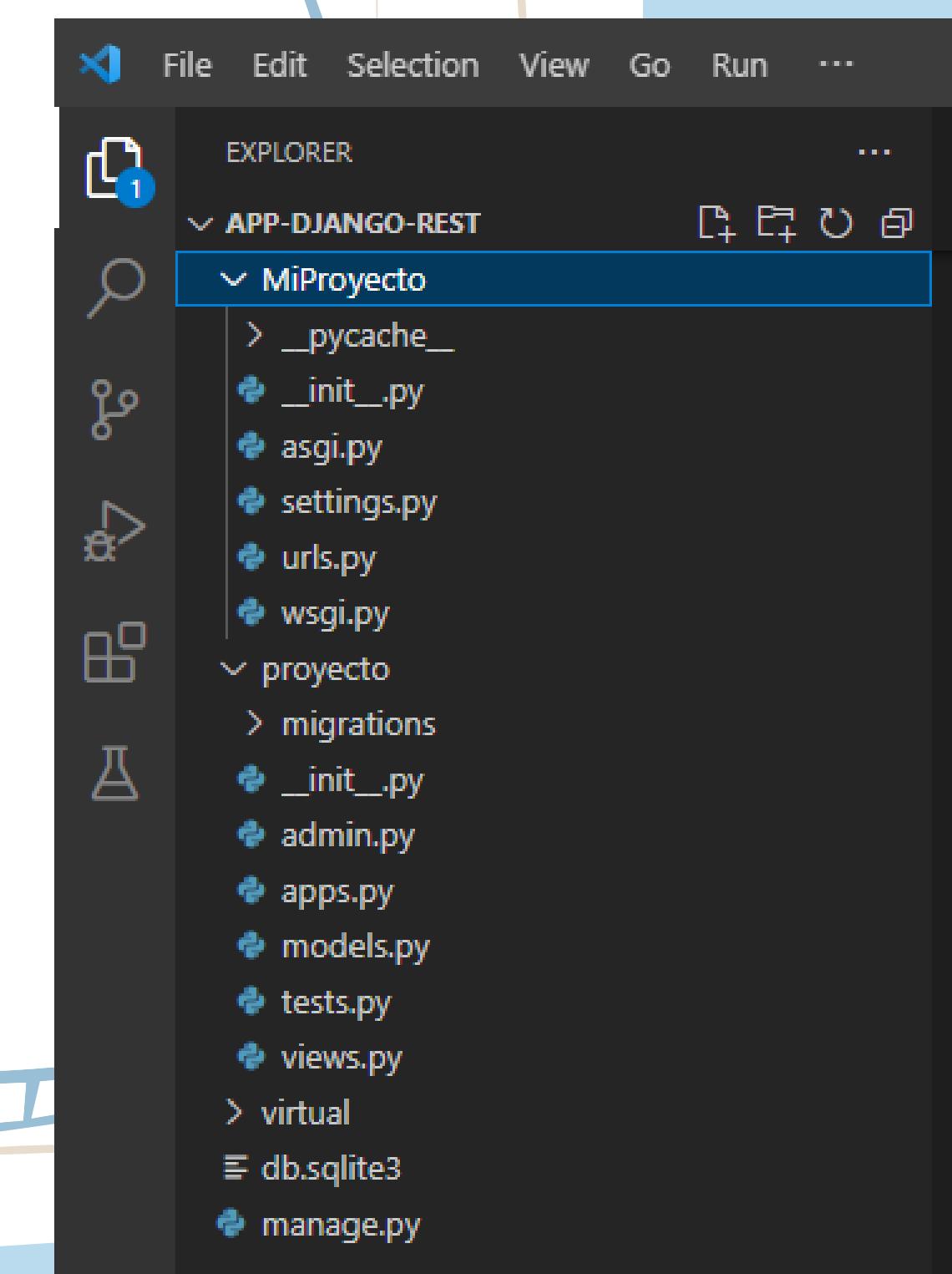
# CREACIÓN

## Crear un proyecto

1. django-admin startproject MiProyecto .

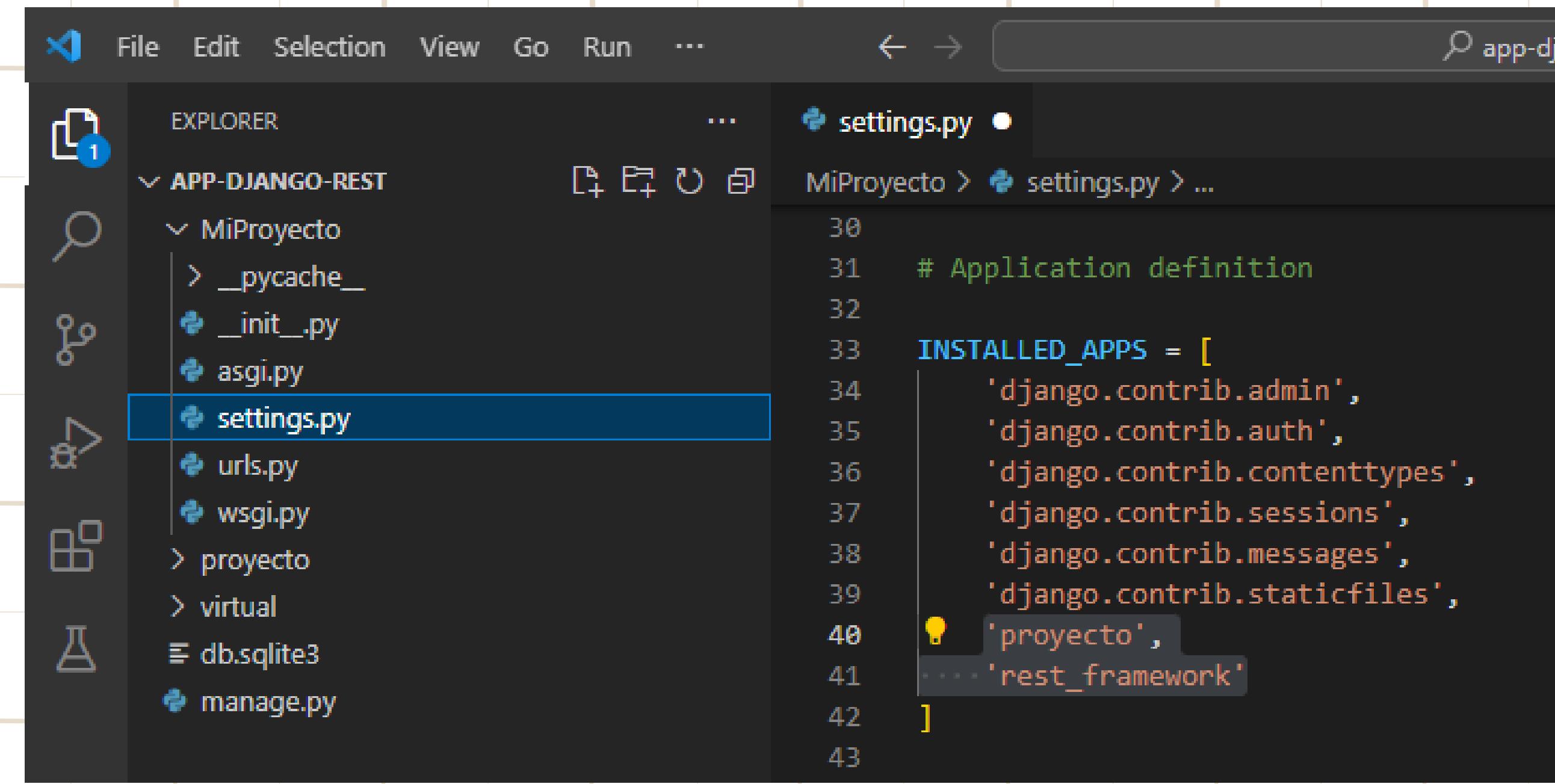
## Crear una aplicación

1. python manage.py startapp proyecto



# FUNCIONAMIENTO

Buscar y agregar componentes al proyecto



```
File Edit Selection View Go Run ... ← → 🔍 app-dja

EXPLORER ...
APP-DJANGO-REST ...
MiProyecto ...
__pycache__
__init__.py
asgi.py
settings.py
urls.py
wsgi.py
proyecto ...
virtual ...
db.sqlite3
manage.py

MiProyecto > settings.py > ...
30
31     # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'proyecto',
41     'rest_framework'
42 ]
43
```

# EJECUTAR

## Correr el servidor

### 1. python manage.py runserver

```
(virtual) C:\Users\Luis Poot\Desktop\app-django-rest>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

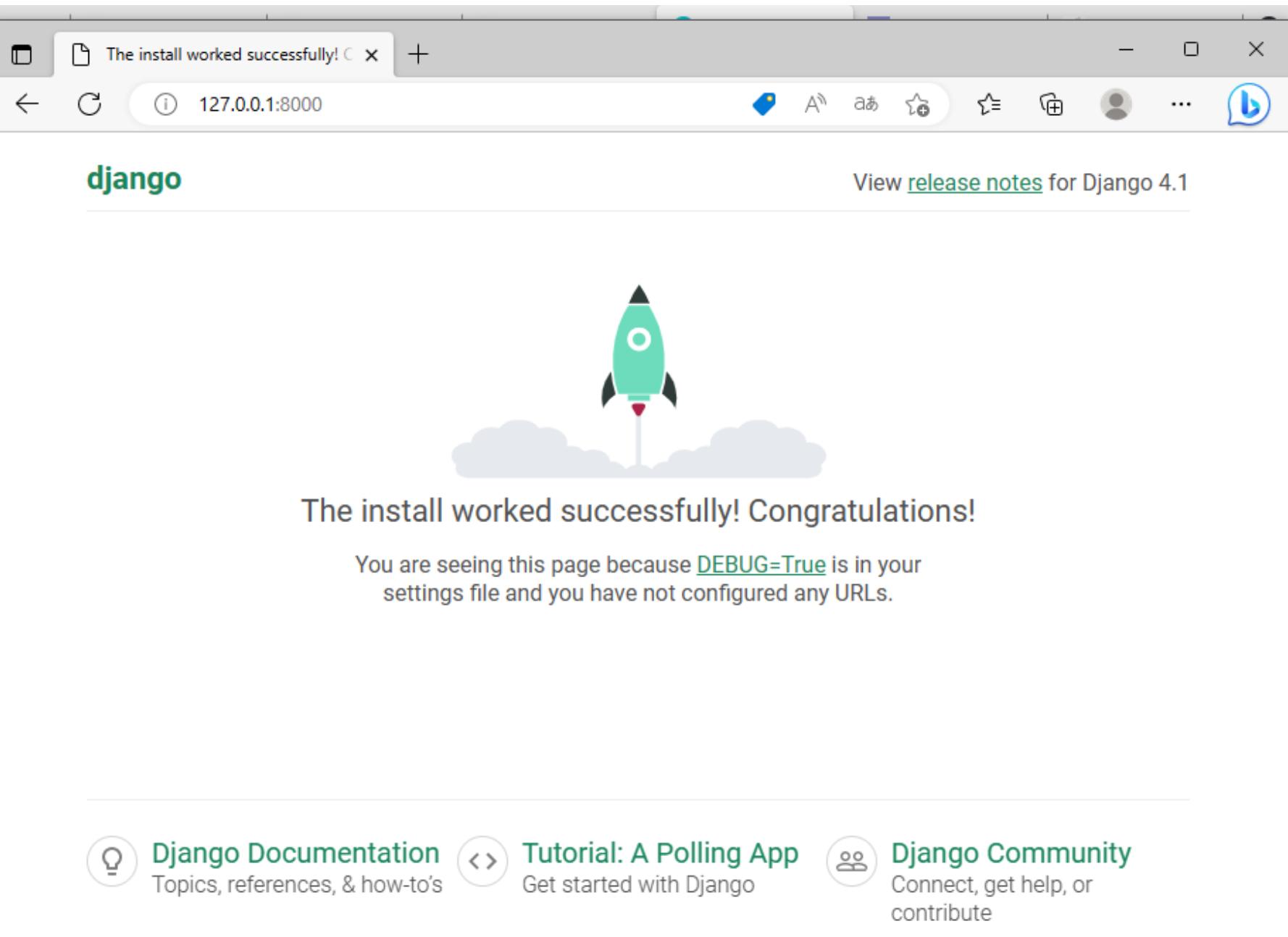
System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
March 21, 2023 - 18:35:16
Django version 4.1.7, using settings 'MiProyecto.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[21/Mar/2023 18:35:56] "GET / HTTP/1.1" 200 10681
[21/Mar/2023 18:35:56] "GET /static/admin/css/fonts.css HTTP/1.1" 200 423
[21/Mar/2023 18:35:57] "GET /static/admin/fonts/Roboto-Light-webfont.woff HTTP/1.1" 200 85692
[21/Mar/2023 18:35:57] "GET /static/admin/fonts/Roboto-Bold-webfont.woff HTTP/1.1" 200 86184
[21/Mar/2023 18:35:58] "GET /static/admin/fonts/Roboto-Regular-webfont.woff HTTP/1.1" 200 85876
Not Found: /favicon.ico
[21/Mar/2023 18:35:58] "GET /favicon.ico HTTP/1.1" 404 2114
[21/Mar/2023 18:35:58] "GET / HTTP/1.1" 200 10681
[21/Mar/2023 18:36:38] "GET / HTTP/1.1" 200 10681
```

# EJECUTAR

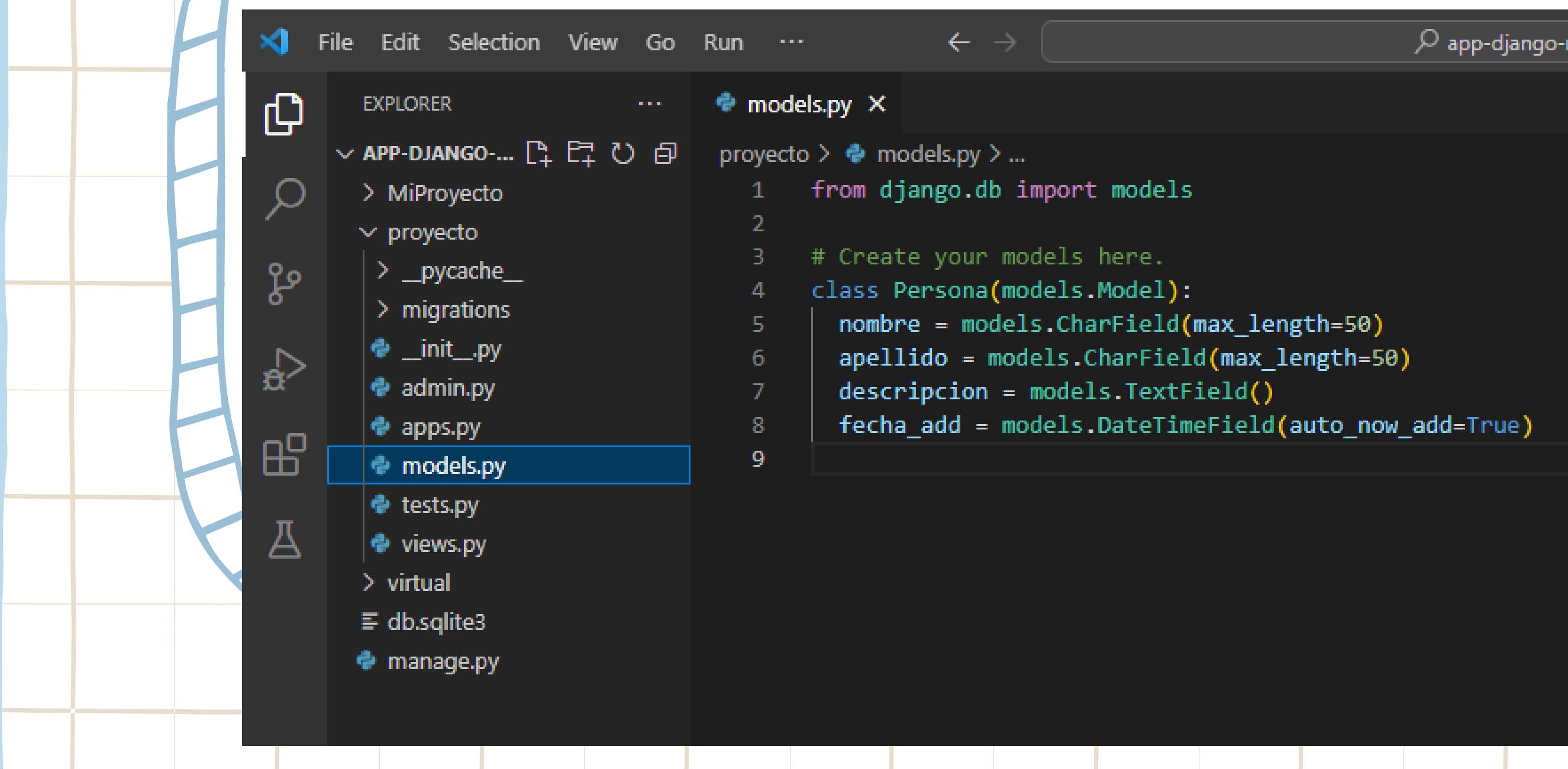
## Resultado

- Ir al navegador a `http://127.0.0.1:8000/`



# AGREGAR MODELO

Agregar el código siguiente en la ubicación  
**visible**



The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists the project structure:

- APP-DJANGO-...
- MiProyecto
- proyecto
  - \_\_pycache\_\_
  - migrations
  - \_\_init\_\_.py
  - admin.py
  - apps.py
  - models.py** (highlighted)
  - tests.py
  - views.py
- virtual
- db.sqlite3
- manage.py

The main editor area displays the contents of the selected `models.py` file:

```
from django.db import models

# Create your models here.
class Persona(models.Model):
    nombre = models.CharField(max_length=50)
    apellido = models.CharField(max_length=50)
    descripcion = models.TextField()
    fecha_add = models.DateTimeField(auto_now_add=True)
```

# AGREGAR MODELO

## Hacer las migraciones

- `python manage.py makemigrations`

```
Migrations for 'proyecto':  
  proyecto\migrations\0001_initial.py  
    - Create model Persona  
  
(virtual) C:\Users\Luis Poot\Desktop\app-django-rest>
```

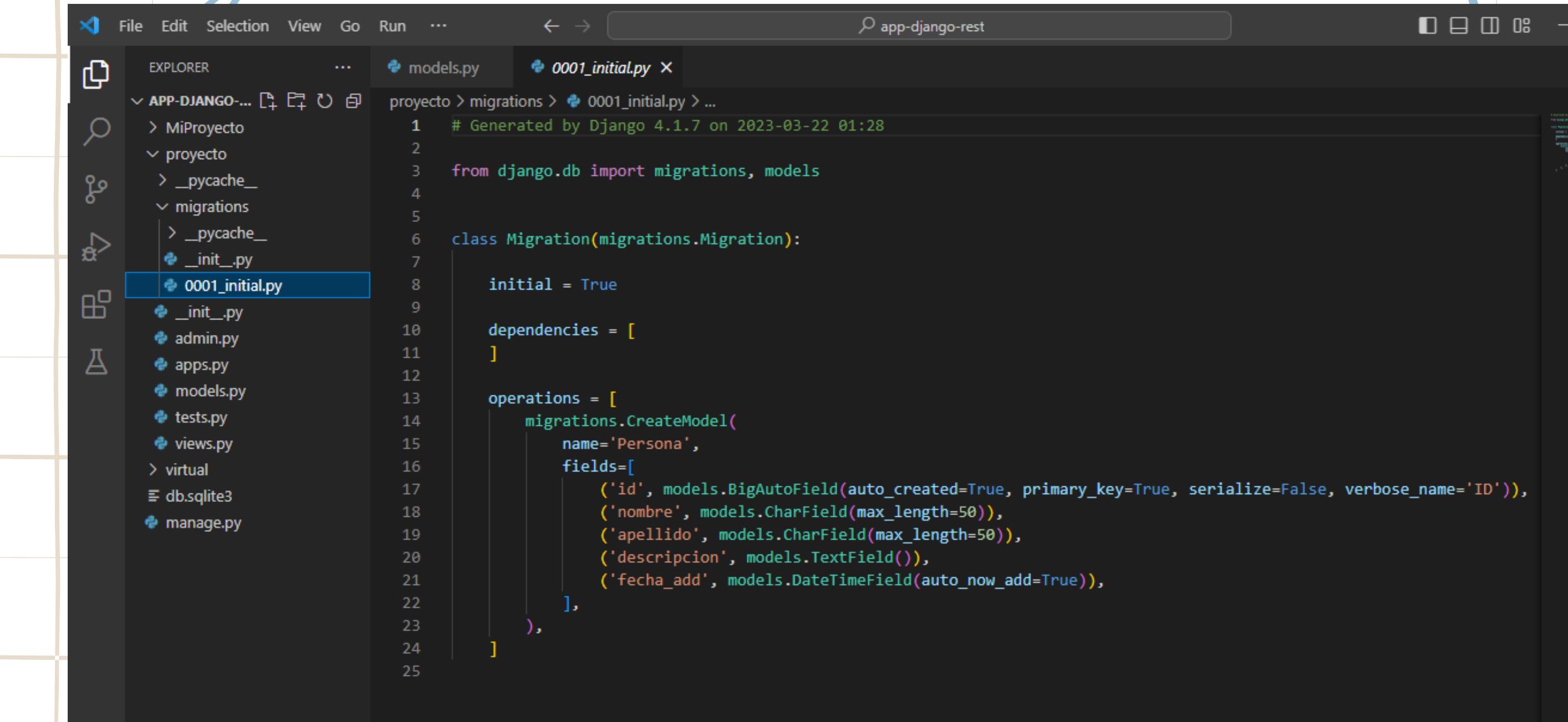
## Usar el modelo creado

- `python manage.py migrate`

```
Operations to perform:  
  Apply all migrations: admin, auth, contenttypes, proyecto, 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 proyecto.0001_initial... OK  
  Applying sessions.0001_initial... OK
```

# AGREGAR MODELO

¿Qué se obtiene de la migración?



The screenshot shows a code editor window with a dark theme. The title bar says "app-django-rest". The left sidebar is an "EXPLORER" view showing the project structure:

- APP-DJANGO-... (selected)
- MiProyecto
- proyecto
  - \_\_pycache\_\_
  - migrations
    - \_\_pycache\_\_
    - \_\_init\_\_.py
    - 0001\_initial.py (selected)
  - \_init\_.py
  - admin.py
  - apps.py
  - models.py
  - tests.py
  - views.py
- virtual
- db.sqlite3
- manage.py

The main editor area shows the content of the selected file, "0001\_initial.py":

```
# Generated by Django 4.1.7 on 2023-03-22 01:28
from django.db import migrations, models

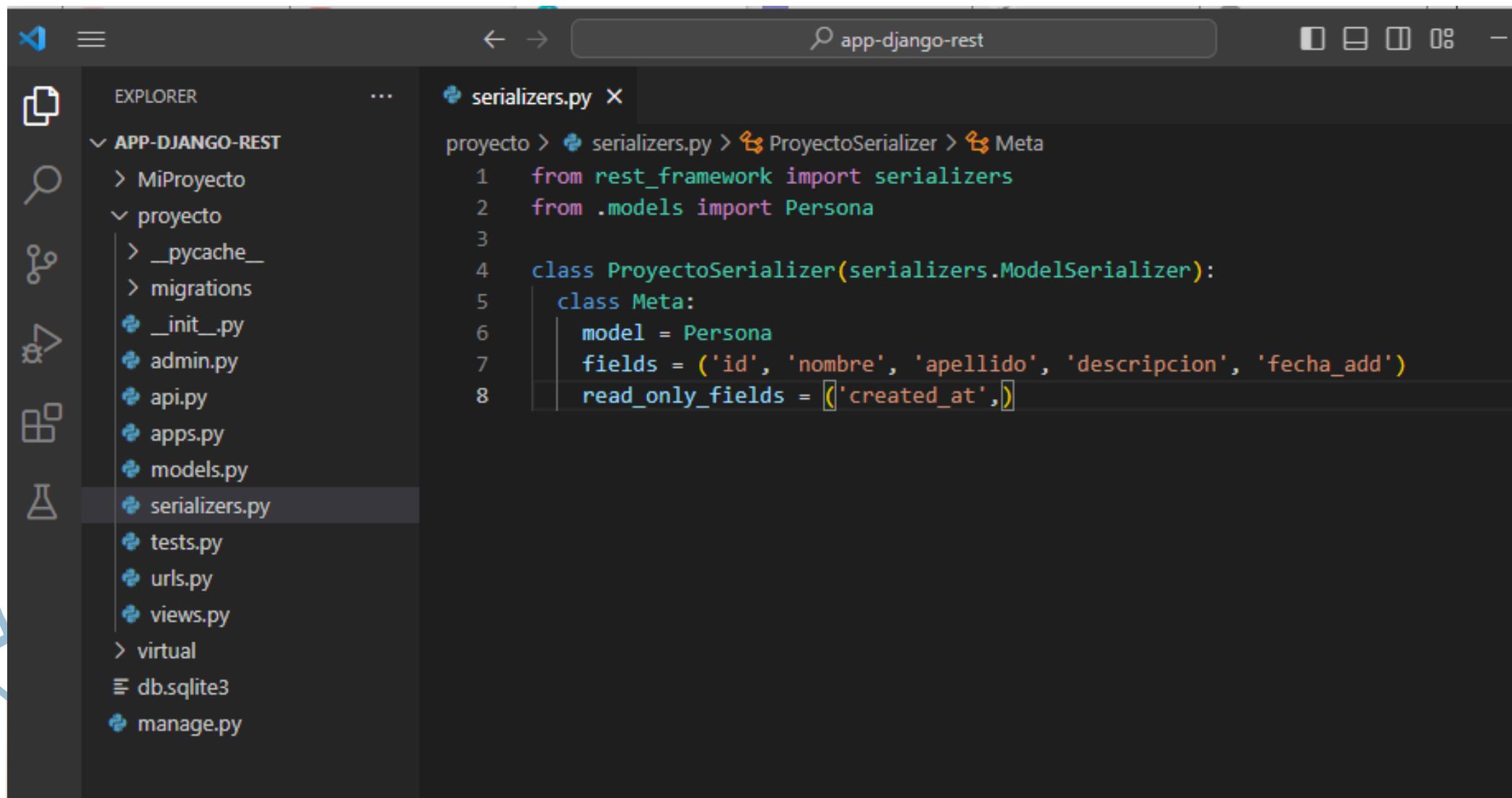
class Migration(migrations.Migration):
    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Persona',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('nombre', models.CharField(max_length=50)),
                ('apellido', models.CharField(max_length=50)),
                ('descripcion', models.TextField()),
                ('fecha_add', models.DateTimeField(auto_now_add=True)),
            ],
        ),
    ],
]
```

# AÑADIR LOS CAMPOS

Crear un archivo serializers.py dentro de la carpeta proyecto y agregar las siguientes líneas

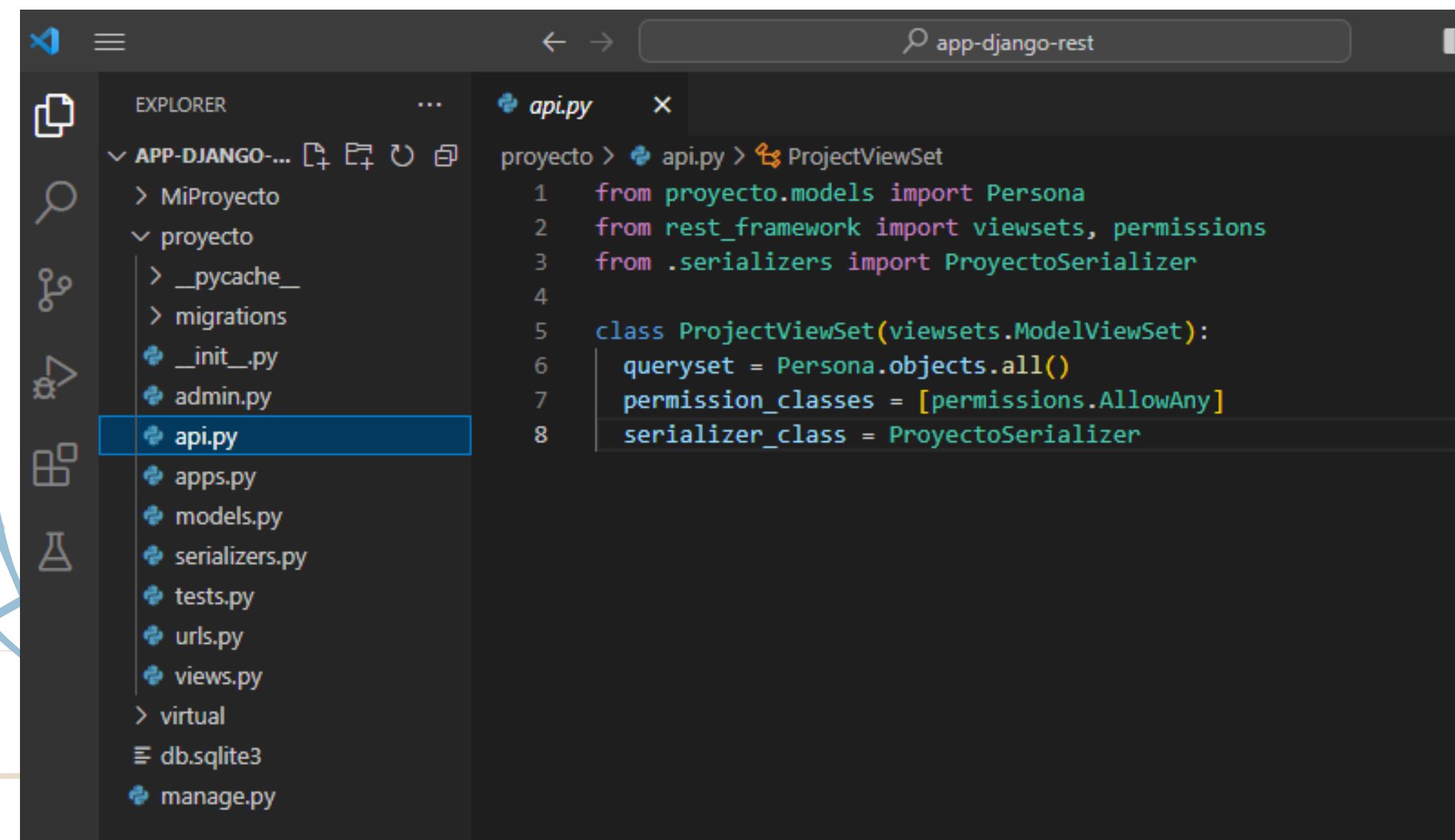


```
from rest_framework import serializers
from .models import Persona

class ProyectoSerializer(serializers.ModelSerializer):
    class Meta:
        model = Persona
        fields = ('id', 'nombre', 'apellido', 'descripcion', 'fecha_add')
        read_only_fields = ['created_at']
```

# AÑADIR CONSULTAS

Crear un archivo api.py dentro de la carpeta proyecto y agregar las siguientes líneas

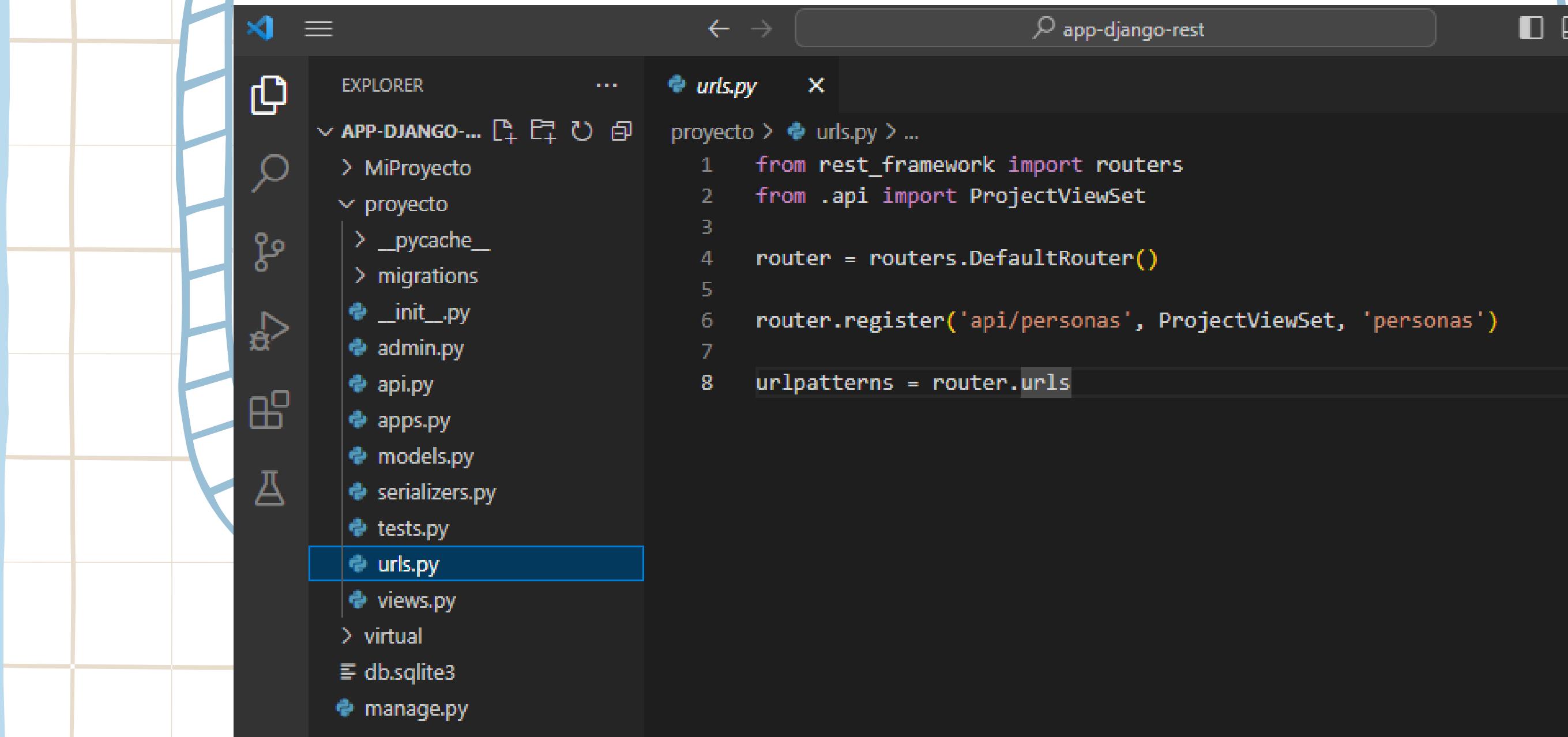


```
from proyecto.models import Persona
from rest_framework import viewsets, permissions
from .serializers import ProyectoSerializer

class ProjectViewSet(viewsets.ModelViewSet):
    queryset = Persona.objects.all()
    permission_classes = [permissions.AllowAny]
    serializer_class = ProyectoSerializer
```

# REGISTRAR URL

Dentro del archivo urls.py dentro de la carpeta proyecto, agregar las siguientes líneas



The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists the project structure:

- APP-DJANGO-...
- MiProyecto
- proyecto
  - \_pycache\_
  - migrations
  - \_\_init\_\_.py
  - admin.py
  - api.py
  - apps.py
  - models.py
  - serializers.py
  - tests.py
  - urls.py
- views.py
- virtual
- db.sqlite3
- manage.py

The 'urls.py' file is selected in the Explorer sidebar and is open in the main editor area. The code content is as follows:

```
from rest_framework import routers
from .api import ProjectViewSet

router = routers.DefaultRouter()
router.register('api/personas', ProjectViewSet, 'personas')

urlpatterns = router.urls
```

# AÑADIR URLs A LA APLICACION PRINCIPAL

Editar el archivo urls.py dentro de la carpeta MiProyecto y agregar las siguientes lineas

```
14     2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15     """
16     from django.contrib import admin
17     from django.urls import path, include
18
19     urlpatterns = [
20         path('admin/', admin.site.urls),
21         path('', include('proyecto.urls')),
22     ]
23
```

# EJECUTAR

## Correr el servidor

### 1. python manage.py runserver

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
March 21, 2023 - 21:39:03
Django version 4.1.7, using settings 'MiProyecto.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[21/Mar/2023 21:39:11] "GET / HTTP/1.1" 200 10681
[21/Mar/2023 21:39:11] "GET /static/admin/css/fonts.css HTTP/1.1" 304 0
[21/Mar/2023 21:39:11] "GET /static/admin/fonts/Roboto-Bold-webfont.woff HTTP/1.1" 304 0
[21/Mar/2023 21:39:11] "GET /static/admin/fonts/Roboto-Regular-webfont.woff HTTP/1.1" 304 0
[21/Mar/2023 21:39:11] "GET /static/admin/fonts/Roboto-Light-webfont.woff HTTP/1.1" 304 0
C:\Users\Luis Poot\Desktop\app-django-rest\MiProyecto"urls.py changed, reloading.
Watching for file changes with StatReloader
Performing system checks...

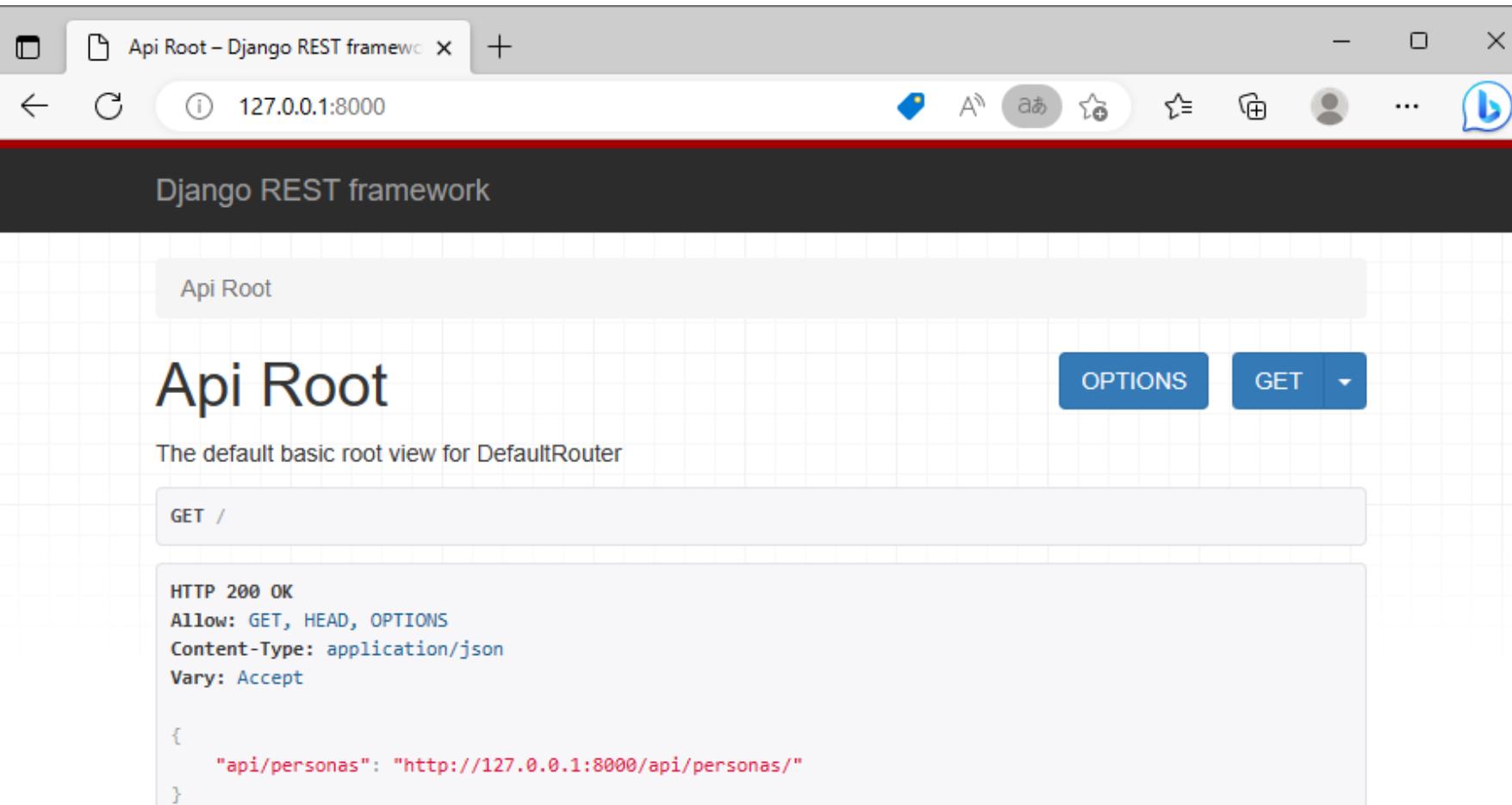
System check identified no issues (0 silenced).
March 21, 2023 - 22:12:37
Django version 4.1.7, using settings 'MiProyecto.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
C:\Users\Luis Poot\Desktop\app-django-rest\proyecto\serializers.py changed, reloading.
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
March 22, 2023 - 03:40:24
Django version 4.1.7, using settings 'MiProyecto.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[22/Mar/2023 03:40:41] "GET / HTTP/1.1" 200 5280
[22/Mar/2023 03:40:43] "GET /static/rest_framework/css/prettyify.css HTTP/1.1" 200 817
[22/Mar/2023 03:40:43] "GET /static/rest_framework/css/bootstrap-tweaks.css HTTP/1.1" 200 3385
[22/Mar/2023 03:40:43] "GET /static/rest_framework/css/default.css HTTP/1.1" 200 1152
[22/Mar/2023 03:40:43] "GET /static/rest_framework/css/bootstrap.min.css HTTP/1.1" 200 121457
[22/Mar/2023 03:40:44] "GET /static/rest_framework/js/default.js HTTP/1.1" 200 1268
[22/Mar/2023 03:40:44] "GET /static/rest_framework/js/csrf.js HTTP/1.1" 200 1719
[22/Mar/2023 03:40:44] "GET /static/rest_framework/js/bootstrap.min.js HTTP/1.1" 200 39680
[22/Mar/2023 03:40:44] "GET /static/rest_framework/js/ajax-form.js HTTP/1.1" 200 3597
[22/Mar/2023 03:40:44] "GET /static/rest_framework/js/prettyify-min.js HTTP/1.1" 200 13632
[22/Mar/2023 03:40:45] "GET /static/rest_framework/img/grid.png HTTP/1.1" 200 1458
```

# EJECUTAR

## Resultado

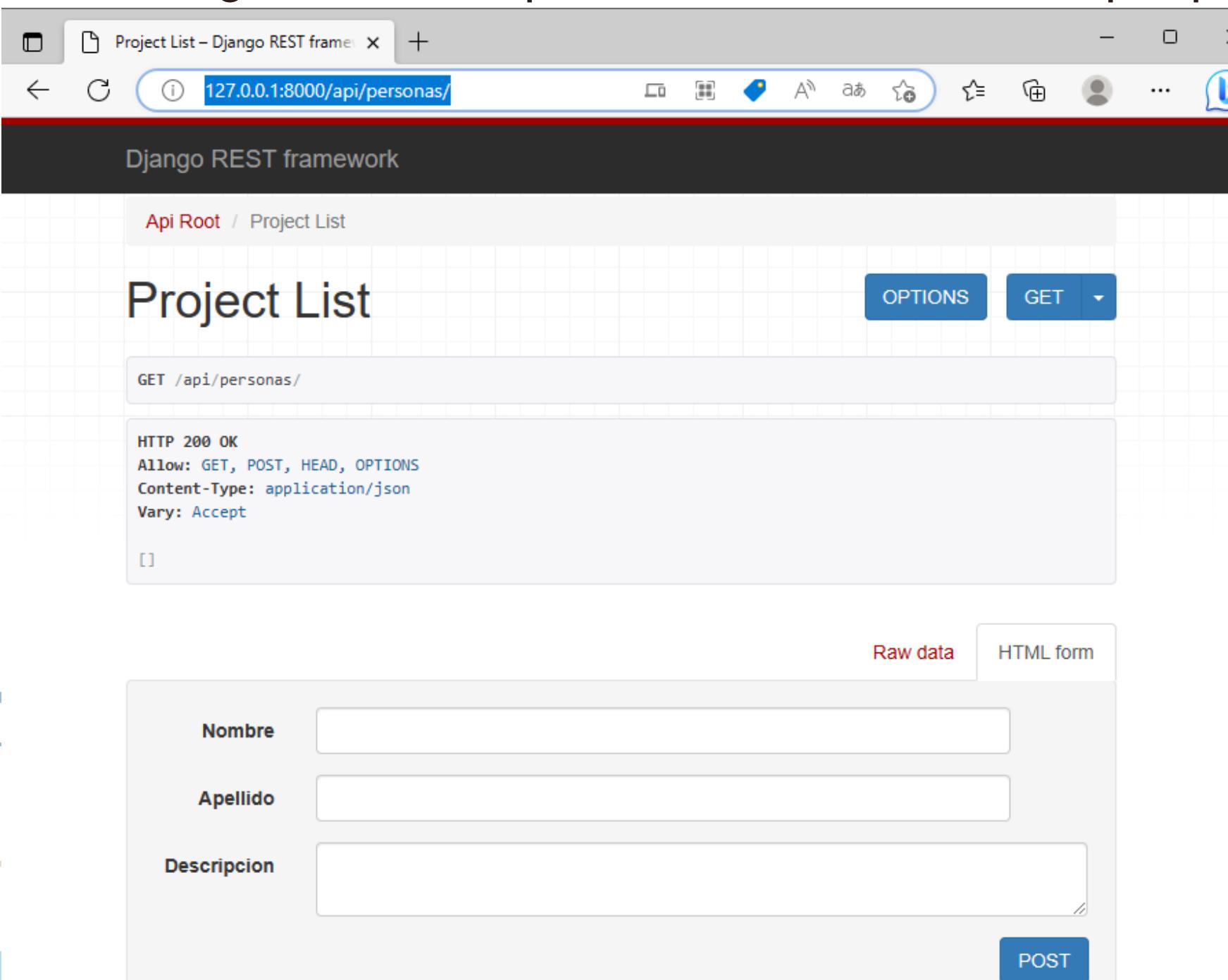
- Ir al navegador a <http://127.0.0.1:8000/>



# EJECUTAR

## Resultado

Ir al navegador a <http://127.0.0.1:8000/api/personas/>



# POST

## Agregar datos

Project List

OPTIONS GET

Raw data

Nombre: Luis

Apellido: Poot

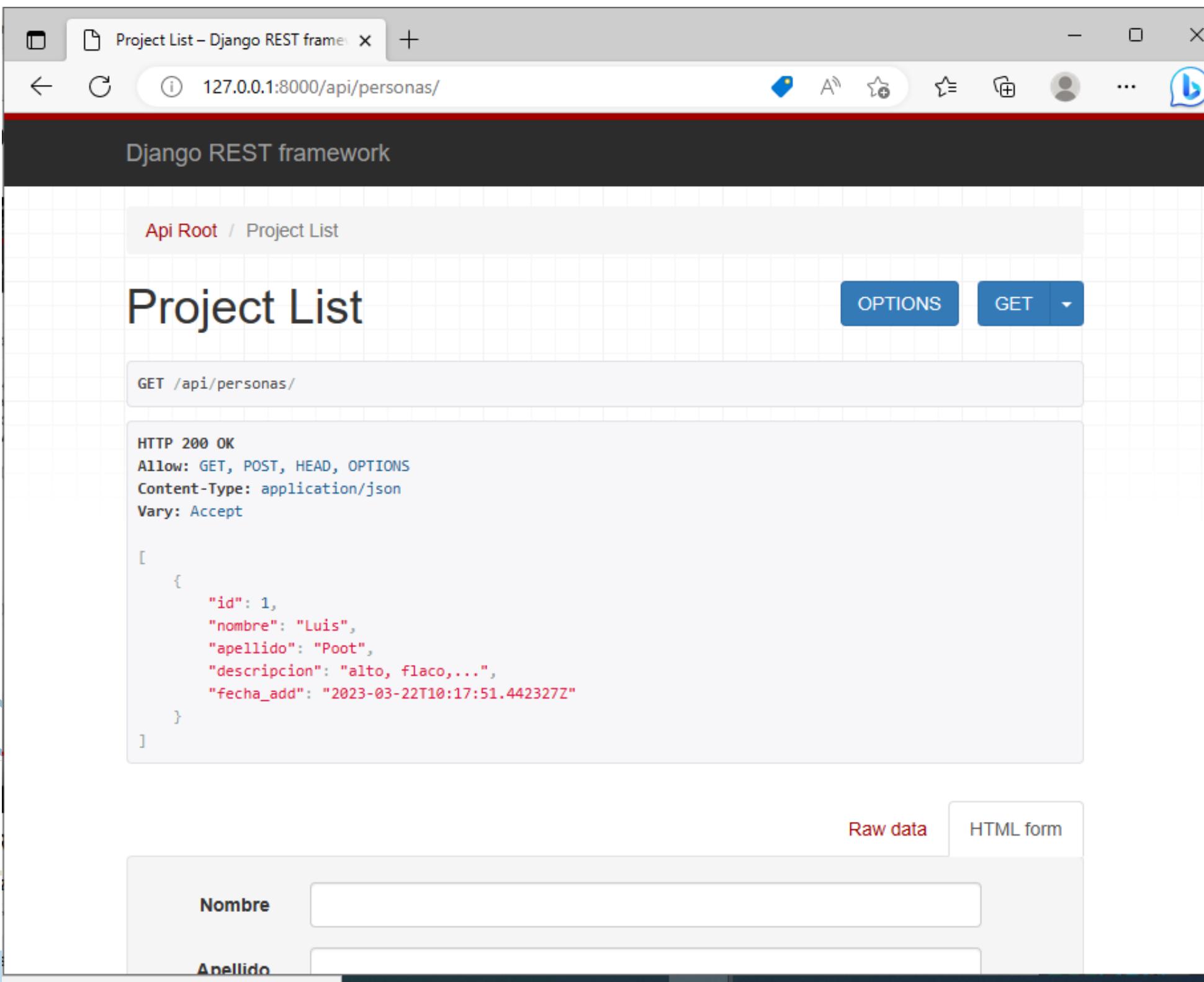
Descripcion: alto, flaco...

Make a POST request on the Project List resource

POST

# GET

Ir nuevamente a <http://127.0.0.1:8000/api/personas/>



**REALIZAR UN SERVICIO  
CON PYTHON USANDO  
DJANGO CON LA  
ARQUITECTURA SOAP**

# PASOS

Una de las opciones más conocidas es Django Suds, que proporciona una interfaz Python para interactuar con servicios web SOAP

Instalar Django Suds

```
pip install django-suds
```

# PASOS

Agregar 'suds' a la lista de aplicaciones

```
INSTALLED_APPS = [  
    # ... otras aplicaciones ...  
    'suds',  
]
```

# PASOS

## Crear una vista en la aplicación

```
from django.http import HttpResponse
from django.views.decorators.csrf import csrf_exempt
from suds.client import Client

@csrf_exempt
def my_soap_view(request):
    # Crear una instancia del cliente SOAP
    client = Client('http://example.com/my/soap/service.wsdl')

    # Llamar a un método en el servicio web
    result = client.service.my_soap_method(param1, param2, ...)

    # Devolver una respuesta como XML
    response = HttpResponse(str(result))
    response['Content-Type'] = 'application/xml'
    return response
```

## CONCLUSIÓN

La vista 'my\_soap\_view' utiliza la biblioteca suds para crear una instancia del cliente SOAP y llamar a un método en el servicio web.

La respuesta se devuelve como un objeto HttpResponse con el contenido XML devuelto por el servicio.

# REFERENCIAS

- Introducción a Django (w3schools.com).  
[https://www.w3schools.com/django/django\\_intro.php](https://www.w3schools.com/django/django_intro.php)
- Create SOAP Web service with Django – Giā  
Dương Đức Minh (wordpress.com).  
<https://ducminhgd.wordpress.com/2017/09/17/create-soap-web-service-with-django/>
- Django documentation | Django documentation |  
Django (djangoproject.com).  
<https://docs.djangoproject.com/en/4.1/>