		<b>Escuela Politécnica Superior</b> <b>Ingeniería Informática</b> <b>Prácticas de Sistemas Informáticos 2</b>			
<b>Grupo</b>	<b>2311</b>	<b>Práctica</b>	1B	<b>Fecha</b>	08/03/2025
<b>Alumno/a</b>		Gómez, Hernández, Marco			
<b>Alumno/a</b>		Haya, de la Vega, Juan			

## Práctica 1B: Cliente y servidor con RPC's y Django

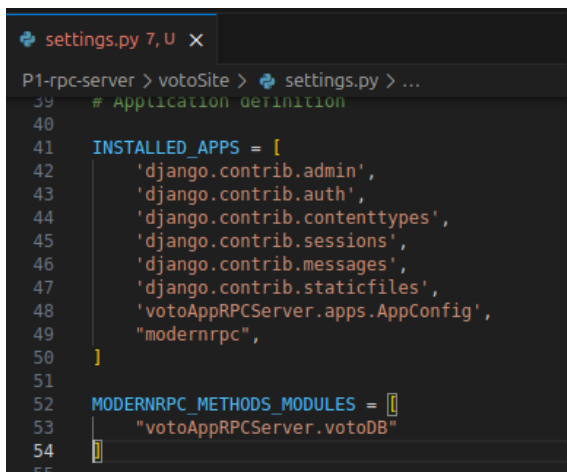
### Ejercicio número 1:

Ejecute los pasos descritos anteriormente, uno tras otro, para crear el nuevo proyecto. Incluya en la memoria evidencias (capturas de pantalla) de haber realizado estos pasos. Indica por qué no será necesario hacer uso de los formularios Django y las plantillas en la aplicación servidor RPC.

Ejecutamos los comandos que se nos piden y todo funciona correctamente (se ha omitido el /\* en el primer comando para que nos lo copie todo bien)

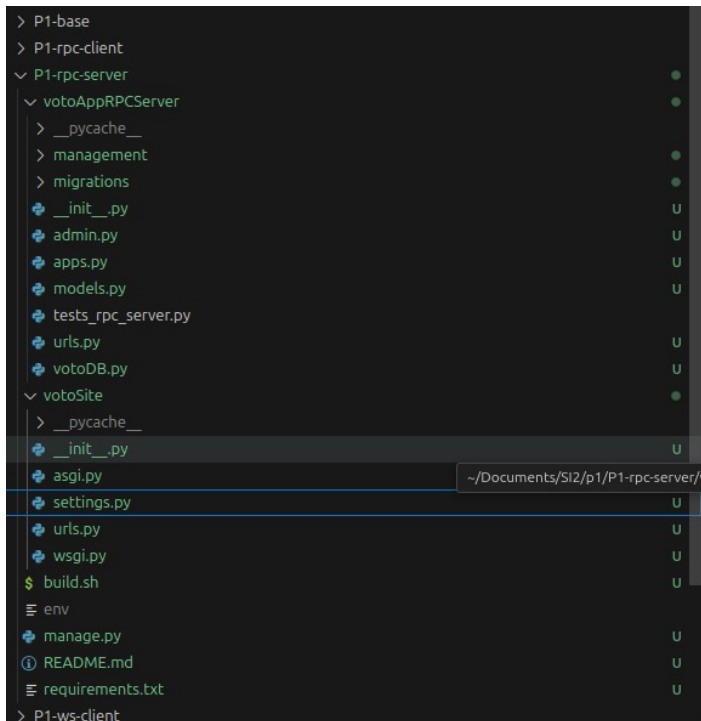
```
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1$ cp -r P1-base/* P1-rpc-server
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1$ cd P1-rpc-server
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ mv votoApp votoAppRPCServer
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ find ./ -type f -exec sed -i "s/votoApp/votoAppRPCServer/g" {} \;
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ rm votoAppRPCServer/forms.py
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ rm votoAppRPCServer/views.py
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ rm -rf votoAppRPCServer/templates
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ rm votoAppRPCServer/tests_views.py
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ rm votoAppRPCServer/tests_models.py
```

Modificamos el settings.py como se nos dice:



```
settings.py 7, U x
P1-rpc-server > votoSite > settings.py > ...
39 # Application definition
40
41 INSTALLED_APPS = [
42     'django.contrib.admin',
43     'django.contrib.auth',
44     'django.contrib.contenttypes',
45     'django.contrib.sessions',
46     'django.contrib.messages',
47     'django.contrib.staticfiles',
48     'votoAppRPCServer.apps.AppConfig',
49     "modernrpc",
50 ]
51
52 MODERNRPC_METHODS_MODULES = [
53     "votoAppRPCServer.votoDB"
54 ]
55
```

Y vemos que la estructura de ficheros que nos queda es la deseada:



No son necesarios los ficheros forms.py, views.py y los ficheros HTML de la carpeta templates (plantillas y formularios django) debido a que el servidor RPC no necesitará generar interfaces HTML porque de esto se encargará el cliente; cliente que llamará al servidor para que este acceda a la base de datos (el usuario no se comunicará directamente con el servidor, sino a través del cliente y su interfaz HTML) mediante RPC's (el cliente llamará a procedimientos del servidor que accederán a la base de datos). Es decir, el servidor sólo tendrá la utilidad de que se utilicen sus procedimientos locales para acceder a la base de datos de manera remota, devolviendo los resultados de los procedimientos en formato XML-RPC (también le llegarán los datos del procedimiento a llamar en este formato), y ya el que haya llamado al procedimiento (cliente), hará lo que deba con esos datos (en este caso generar interfaces HTML con ellos).

## ***Ejercicio número 2:***

Ejecute los pasos descritos anteriormente, uno tras otro, para exportar la funcionalidad de acceso a la BD como procedimientos remotos. Incluya en la memoria evidencias (capturas de pantalla) de haber realizado estos pasos. Indica razonadamente por qué es necesario hacer uso del método `model_to_dict`.

Capturas solicitadas:

```

urls.py x
P1-rpc-server > votoAppRPCServer > urls.py
1 """
2 URL configuration for VotingProj project.
3
4 The 'urlpatterns' list routes URLs to views. For more information please see:
5 https://docs.djangoproject.com/en/4.2/topics/http/urls/
6 Examples:
7 Function views
8     1. Add an import: from my_app import views
9     2. Add a URL to urlpatterns: path('', views.home, name='home')
10 Class-based views
11     1. Add an import: from other_app.views import Home
12     2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
13 Including another URLconf
14     1. Import the include() function: from django.urls import include, path
15     2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
16 """
17 from django.urls import path
18 from modernrpc.views import RPCEntryPoint
19
20 urlpatterns = [path("rpc/", RPCEntryPoint.as_view(), name="rpc")]
21

```

```

votoDB.py 2, M x
P1-rpc-server > votoAppRPCServer > votoDB.py > eliminar_voto
8 from votoAppRPCServer.models import Censo, Voto
9 from modernrpc.core import rpc_method
10 from django.forms.models import model_to_dict
11
12
13 @rpc_method
14 def verificar_censo(censo_data):
15     """ Check if the voter is registered in the Censo
16     :param censo_dict: dictionary with the voter data
17                       (as provided by CensoForm of the client)
18     :return True or False if censo_data is not valid
19     """
20     if bool(censo_data) is False or not\
21         Censo.objects.filter(**censo_data).exists():
22         return False
23     return True
24
25
26 @rpc_method
27 def registrar_voto(voto_dict):
28     """ Register a vote in the database
29     :param voto_dict: dictionary with the vote data (as provided by VotoForm
30                     of the client)
31     plus de censo_id (numeroDNI) of the voter
32     :return new voto info if succesful, None otherwise
33     """
34     try:
35         voto = Voto.objects.create(**voto_dict)
36         # get default values from voto
37         voto = Voto.objects.get(pk=voto.pk)
38         voto_a_devolver = model_to_dict(voto)
39         voto_a_devolver['marcaTiempo'] = str(voto.marcaTiempo)
40     except Exception as e:
41         print("Error: Registrando voto: ", e)
42         return None
43     return voto_a_devolver
44

```

```

votoDB.py 2, M x
P1-rpc-server > votoAppRPCServer > votoDB.py > eliminar_voto
45
46 @rpc_method
47 def eliminar_voto(idVoto):
48     """ Delete a vote in the database
49     :param idVoto: id of the vote to be deleted
50     :return True if succesful,
51     False otherwise
52     """
53     try:
54         voto = Voto.objects.get(id=idVoto)
55     except Voto.DoesNotExist:
56         return False
57     voto.delete()
58     return True
59
60
61 @rpc_method
62 def get_votos_from_db(idProcesoElectoral):
63     """ Gets votes in the database correspondint to some electoral processs
64     :param idProcesoElectoral: id of the vote to be deleted
65     :return list of dicts with the info of each vote found
66     """
67     votos = Voto.objects.filter(idProcesoElectoral=idProcesoElectoral)
68
69     votos_a_devolver = []
70     for voto in votos:
71         voto_a_devolver = model_to_dict(voto)
72         voto_a_devolver['marcaTiempo'] = str(voto.marcaTiempo)
73         votos_a_devolver.append(voto_a_devolver)
74
75     return votos_a_devolver

```

El uso del método `model_to_dict` en este contexto es necesario porque convierte instancias de modelos de Django en diccionarios, lo que facilita la serialización y transmisión de datos en las respuestas de los métodos RPC. Dado que Django maneja modelos como objetos de Python con atributos que pueden incluir relaciones complejas o valores no directamente serializables, `model_to_dict` permite extraer los datos en un formato estructurado y manejable, asegurando que la información del voto registrado o recuperado pueda ser enviada fácilmente como un diccionario (tipo aceptado para este tipo de comunicación, no como los modelos de Django).

### Ejercicio número 3:

Ejecuta los pasos descritos más arriba e incluye evidencias en la memoria de haberlos llevado a cabo. Ejecuta los test proporcionados con el proyecto y comprueba que no devuelven errores. Adjunta en la memoria una captura de pantalla en la que se muestre el resultado de ejecutar los test (`python manage.py test votoAppRPCServer.tests_rpc_server`). Los test proporcionados deben tomarse como requisitos extras del sistema.

Capturas sobre los pasos realizados:

En el fichero `env`, ponemos la base de datos de la VM1, y encendemos la misma:

```
DATABASE_SERVER_URL='postgres://alumnodb:alumnodb@localhost:15432/voto'
```

```
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ python manage.py makemigrations
No changes detected
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, votoAppRPCServer
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
  Applying votoAppRPCServer.0001_initial... OK
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ python manage.py populate
Database cleaned successfully
Censo objects created successfully
❖ (si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 25, 2025 - 19:26:19
Django version 4.2.13, using settings 'votoSite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

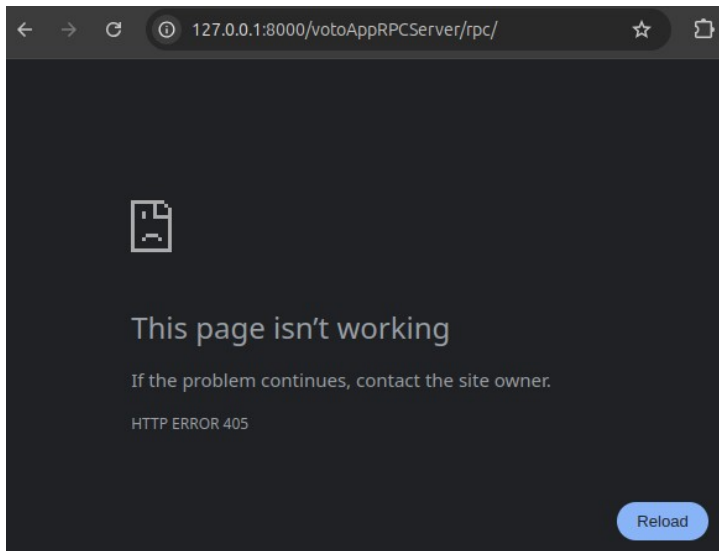
Prueba de que la base de datos se ha creado y poblado:

The screenshot shows the Database Navigator interface. On the left, the tree view shows the database structure: 'voto' (localhost:15432) contains a 'public' schema with several tables, including 'censo' which is highlighted. The 'censo' table has 224 rows. On the right, the 'Data' tab is active, displaying a grid of 200 rows (rows 1-18 are visible). The columns are: 'numeroDNI', 'nombre', 'fechaNacimiento', 'anioCenso', and 'codigoAutorizacion'. The data shows a list of people with their DNI, names, birth dates, and census years.

	numeroDNI	nombre	fechaNacimiento	anioCenso	codigoAutorizacion
1	39739740E	Jose Moreno Locke	09/04/66	2025	729
2	83583583L	Restituta Sparrow Martinez	31/08/73	2025	535
3	67867868T	Randall Martinez Mojamuto	04/06/96	2025	056
4	12812814Q	Gabriel Ribas Pomares	30/05/54	2025	605
5	92992992L	Benjamin Reyes Lopez	06/05/90	2025	095
6	50850851I	Emiliano Reyes Gracia	16/11/81	2025	428
7	47047047F	Luis Mas Moreno	27/08/54	2025	620
8	29829830H	Emiliano Locke Gracia	08/05/60	2025	796
9	26926927J	Camilo Marques Gibson	28/05/94	2025	716
10	97797797K	Eva Ribera Pelaez	24/04/51	2025	697
11	18618619K	Armando Linus Ribera	28/04/89	2025	424
12	30630631P	Luisa Linus Coll	13/02/00	2025	090
13	59659659J	Alberto Martinez Poza	28/09/93	2025	466
14	27627628Z	Jack Garau Morales	17/05/89	2025	947
15	87387387L	Kate Moss Sparrow	28/12/92	2025	853
16	94994994D	Sofia Poza Gracia	06/07/76	2025	104
17	49349349T	Jack Punset Gonzalez	11/01/59	2025	794
18	32932933O	Petra Marques Pelaez	09/06/73	2025	599

Prueba de que aparece el error HTTP 405:





```
marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 25, 2025 - 19:17:25
Django version 4.2.13, using settings 'votoSite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

Not Found: /
[25/Feb/2025 19:21:08] "GET / HTTP/1.1" 404 2178
Not Found: /favicon.ico
[25/Feb/2025 19:21:08] "GET /favicon.ico HTTP/1.1" 404 2229
Method Not Allowed (GET): /votoAppRPCServer/rpc/
Method Not Allowed: /votoAppRPCServer/rpc/
[25/Feb/2025 19:21:41] "GET /votoAppRPCServer/rpc/ HTTP/1.1" 405 0
```

Ejecutando los tests se puede ver que sale lo siguiente, que aunque se pasan todos los tests, salta un warning que nos dice algo sobre la carpeta staticfiles:

```
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ python manage.py test votoAppRPCServer.tests_rpc_server
Found 4 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
/home/marco/Documents/SI2/si2_venv/lib/python3.12/site-packages/django/core/handlers/base.py:61: UserWarning: No directory at: /home/marco/Documents/SI2/p1/P1-rpc-server/staticfiles/
  mw_instance = middleware(adapted_handler)
....
Ran 4 tests in 0.083s

OK
Destroying test database for alias 'default'...
```

Este warning, lo podemos solucionar haciendo un collectstatic (para crear el directorio staticfiles solicitado; aunque para la entrega lo eliminaremos ya que ocupa mucho y se puede crear de está sencilla manera) antes:

```
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ python manage.py collectstatic --noinput
125 static files copied to '/home/marco/Documents/SI2/p1/P1-rpc-server/staticfiles', 375 post-processed.
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ python manage.py test votoAppRPCServer.tests_rpc_server
Found 4 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
....
Ran 4 tests in 0.262s

OK
Destroying test database for alias 'default'...
```

## Ejercicio número 4:

Ejecuta los pasos descritos más arriba. Prueba a acceder desde el navegador del PC del laboratorio a la URL `localhost:28000/votoAppRPCServer/rpc` para comprobar que el despliegue de la aplicación es correcto. Incluye evidencias en la memoria en forma de capturas de pantalla mostrando que el acceso a dicha URL es correcto.

Antes que nada, cabe mencionar que tenemos el repositorio en la VM2 y VM3 configurado ya de la P1A, con lo que no se incluye aquí como configurar todo, sino los cambios que hay que hacer en los ficheros que ya teníamos en la P1A. Además, se destaca que estamos usando el mismo directorio de trabajo `p1base` (en las VM) para guardar todos los proyectos de esta práctica por comodidad. Por esto las rutas pueden no ser lo que cabría esperar.

Aunque no lo pone en ningún lado, modificaremos el fichero `~/.bashrc` en la VM2 para que la variable de entorno `TARGET` guarde la ruta al proyecto ya que se utiliza para que al hacerle push al repo se haga un migrate y un collectstatic. Nos queda así:

```
sudo loadkeys es
export PATH=/home/si2/venv/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin
export TARGET="/home/si2/repo/p1base/p1/P1-rpc-server"
```

La variable de entorno `target` se usa en `/home/si2/repo/p1base.git/hooks/post-receive`:

```
#!/bin/bash
GIT_WORK_TREE=/home/si2/repo/p1base git checkout -f

# Reiniciar Gunicorn
sudo systemctl restart gunicorn

python $TARGET/manage.py migrate
python $TARGET/manage.py collectstatic --noinput
```

A su vez, tenemos que cambiar el fichero `/etc/systemd/system/gunicorn.service` para indicarle a Gunicorn dónde está nuestro proyecto:

```
[Unit]
Description=Gunicorn WSGI Application Server
After=network.target

[Service]
User=si2
Group=si2
WorkingDirectory=/home/si2/repo/p1base/p1/P1-rpc-server
Environment="PATH=/home/si2/venv/bin"
ExecStart=/home/si2/venv/bin/gunicorn --workers 1 --bind 0.0.0.0:8000 votoSite.wsgi:application
ExecReload=/bin/kill -s HUP $MAINPID
KillMode=mixed
TimeoutStopSec=5
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

Hacemos un push desde el host:

```

(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-server$ git push vm2 main
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 4 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (10/10), 997.37 KiB | 12.16 MiB/s, done.
Total 10 (delta 5), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Operations to perform:
remote:   Apply all migrations: admin, auth, contenttypes, sessions, votoAppRPCServer
remote: Running migrations:
remote:   No migrations to apply.
remote:
remote: 0 static files copied to '/home/si2/repo/plbase/p1/P1-rpc-server/staticfiles', 125 unmodified, 349 post-processed.
To ssh://localhost:22022/~/.repo/plbase.git
   ac34207..ad4ed15  main -> main

```

Ahora, creamos un env en el proyecto del servidor RPC en la VM2 y le ponemos esto (la ip que pone es la de la interfaz wifi del host y el puerto es el de postgres de la VM1 visto desde el host):

```

# IMPORTANT: this file should not be in a repository
# To remove a file named env from a Git repository
# but keep it in the source (local system), follow these steps:
# Remove the file from Git tracking but keep it locally
## git rm --cached env
# Add 'env' to .gitignore (so it's not tracked again)
## echo "env" >> .gitignore
# Commit the changes
## git commit -m "Removed env from Git tracking and added to .gitignore"
# Push the changes to the remote repository
## git push
# use sqlite 3
## DATABASE_SERVER_URL=sqlite:///db.sqlite3
# use postgres

DATABASE_SERVER_URL='postgres://alumnodb:alumnodb@192.168.1.149:15432/voto'
# Cambiar la ip por la ip de la interfaz wifi de nuestro host si lo ejecutamos en la máquina virtual
# o dejarlo en localhost si lo ejecutamos en el host;
# y el puerto 15432 si la BD está en la VM1 y 5432 si la BD está en el host

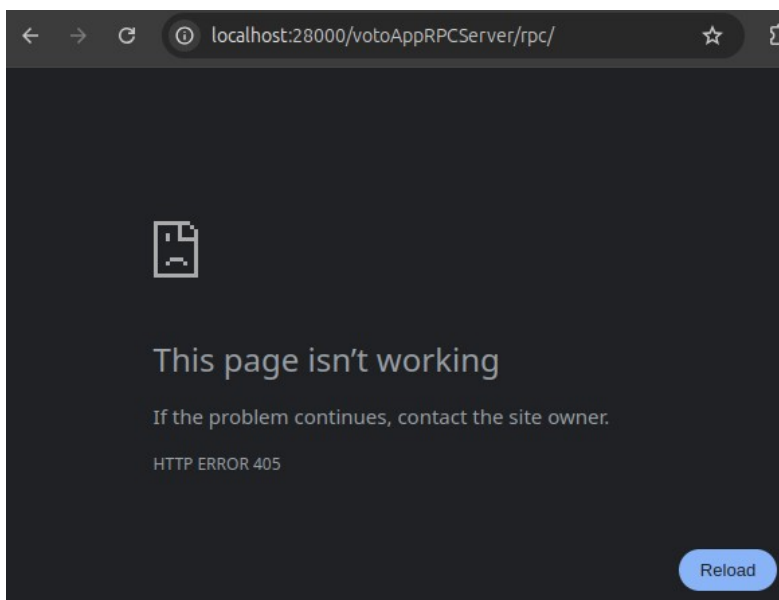
# The client does not need to store data in any database
# so let us define a sqlite in orden to avoid warning messages

# Poner a True si estamos en un entorno de desarrollo
DEBUG=True

SECRET_KEY = 'django-insecure-alczftn)j1#v%xmk@5j(n*px43c8kxgi_ua4%khc+t7g_)s9d'

```

Y ahora ya probaremos a abrir la url que se nos dice en el navegador para ver si devuelve lo que debería (error 405, como se vio antes):



Aunque no se pide, ejecutamos los tests y todo sale perfecto:



```

si2@si2-ubuntu-vm-3:~/repo/plbase/pl/P1-rpc-server$ python manage.py test votoAppRPCServer.tests_rpc_server
Found 4 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
....
-----
Ran 4 tests in 0.368s

OK
Destroying test database for alias 'default'...

```

## Ejercicio número 5:

Ejecute los pasos descritos anteriormente, uno tras otro, para crear el nuevo proyecto. Incluya en la memoria evidencias (capturas de pantalla) de haber realizado estos pasos. Indica por qué no será necesario hacer uso de los modelos de datos Django en la aplicación cliente RPC.

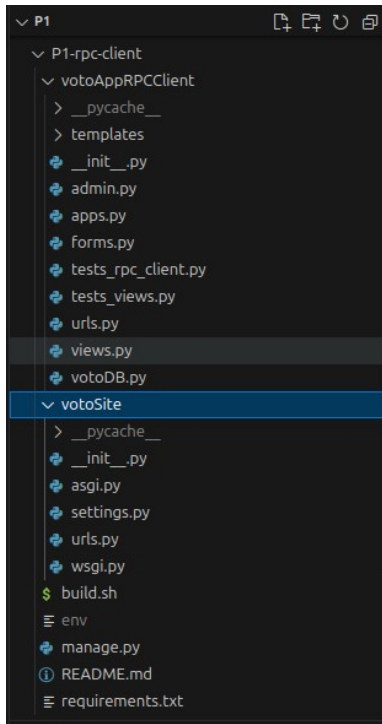
Ejecutamos los comandos que se nos piden y todo funciona correctamente (se ha omitido el /\* en el primer comando para que nos lo copie todo bien)

```

• (si2_venv) marco@marco-laptop:~/Documents/SI2/pl$ cp -r P1-base/* P1-rpc-client
• (si2_venv) marco@marco-laptop:~/Documents/SI2/pl$ cd P1-rpc-client
• (si2_venv) marco@marco-laptop:~/Documents/SI2/pl/P1-rpc-client$ mv votoApp votoAppRPCClient
• (si2_venv) marco@marco-laptop:~/Documents/SI2/pl/P1-rpc-client$ find ./ -type f -exec sed -i "s/votoApp/votoAppRPCClient/g" {} \;
• (si2_venv) marco@marco-laptop:~/Documents/SI2/pl/P1-rpc-client$ rm -rf votoAppRPCClient/models.py
• (si2_venv) marco@marco-laptop:~/Documents/SI2/pl/P1-rpc-client$ rm -rf votoAppRPCClient/migrations
• (si2_venv) marco@marco-laptop:~/Documents/SI2/pl/P1-rpc-client$ rm -rf votoAppRPCClient/management
• (si2_venv) marco@marco-laptop:~/Documents/SI2/pl/P1-rpc-client$ rm -rf votoAppRPCClient/tests_models.py
• (si2_venv) marco@marco-laptop:~/Documents/SI2/pl/P1-rpc-client$

```

Y vemos que la estructura de ficheros que nos queda es la deseada:



Modificamos lo que se nos pide en los ficheros que se nos pide:

```
settings.py 7 X
P1-rpc-client > votoSite > settings.py > ...
147   RPCAPIBASEURL = os.environ.get("RPCAPIBASEURL")
148

$ env
P1-rpc-client > $ env
30
31   RPCAPIBASEURL='http://localhost:28000/votoAppRPCServer/rpc/'
32   # Cambiar la ip por la ip de la interfaz wifi de nuestro host si lo ejecutamos en la máquina virtual
33   # o dejarlo en localhost si lo ejecutamos en el host;
34   # y el puerto poner el en el que se esté ejecutando el servidor RPC en el host si lo tenemos en el host,
35   # o el puerto 28000 si el servidor RPC está en la VM2
```

Los modelos no son necesarios en la aplicación cliente RPC porque este no va a interactuar con la base de datos mediante modelos propios que se traduzcan a esta, sino que esto lo hará el servidor, y el cliente usará los procedimientos de este de manera remota, siendo el servidor el que accede a sus modelos (y estos a la base de datos) para devolver los datos obtenidos al cliente.

## Ejercicio número 6:

Ejecute los pasos descritos anteriormente, uno tras otro, para crear el nuevo fichero `votoDB.py`, que invoque la funcionalidad de la aplicación como llamadas a procedimiento remoto. Comenta cada función describiendo tanto los argumentos de entrada como los valores devueltos. Incluya en la memoria evidencias (capturas de pantalla) de haber realizado estos pasos. Indica qué tipo de binding, de los tipos vistos en las transparencias de teoría sobre RPC, realiza el cliente RPC codificado.

Captura del fichero modificado completamente:

```
votoDB.py 1, M X
P1-rpc-client > votoAppRPCClient > votoDB.py > verificar_censo
8   from django.conf import settings
9   from xmlrpc.client import ServerProxy
10
11
12   def verificar_censo(censo_data):
13       """ Check if the voter is registered in the Censo
14       :param censo_dict: dictionary with the voter data
15       |                 (as provided by CensoForm)
16       :return True or False if censo_data is not valid
17       """
18       with ServerProxy(settings.RPCAPIBASEURL) as proxy:
19           return proxy.verificar_censo(censo_data)
20
21
22   def registrar_voto(voto_dict):
23       """ Register a vote in the database
24       :param voto_dict: dictionary with the vote data (as provided by VotoForm)
25       | plus de censo_id (numeroDNI) of the voter
26       :return new voto info if succesful, None otherwise
27       """
28       with ServerProxy(settings.RPCAPIBASEURL) as proxy:
29           return proxy.registrar_voto(voto_dict)
30
31
32   def eliminar_voto(idVoto):
33       """ Delete a vote in the database
34       :param idVoto: id of the vote to be deleted
35       :return True if succesful,
36       | False otherwise
37       """
38       with ServerProxy(settings.RPCAPIBASEURL) as proxy:
39           return proxy.eliminar_voto(idVoto)
40
41
42   def get_votos_from_db(idProcesoElectoral):
43       """ Gets votes in the database correspondint to some electoral processs
44       :param idProcesoElectoral: id of the vote to be deleted
45       :return list of dicts with each vote found info
46       """
47       with ServerProxy(settings.RPCAPIBASEURL) as proxy:
48           return proxy.get_votos_from_db(idProcesoElectoral)
```

Aunque ya se comenta en el propio código, aquí lo describimos rápido todo:

`verificar_censo(censo_data)`: Recibe un diccionario `censo_data` con la información del votante, proporcionada por `CensoForm`. Devuelve `True` si el votante está registrado en el censo, de lo contrario, `False`.

`registrar_voto(voto_dict)`: Recibe un diccionario `voto_dict` con los datos del voto, incluyendo el `censo_id` (DNI del votante), proporcionados por `VotoForm`. Retorna la información del nuevo voto registrado en forma de diccionario (por ser RPC, no se devuelve como un objeto de un modelo ya que sería un tipo no soportado por XML-RPC) si el proceso es exitoso, o `None` en caso contrario.

`eliminar_voto(idVoto)`: Recibe el identificador `idVoto` del voto a eliminar. Retorna `True` si la eliminación fue exitosa o `False` en caso contrario.

`get_votos_from_db(idProcesoElectoral)`: Recibe el identificador `idProcesoElectoral` del proceso electoral del cual se quieren obtener los votos. Devuelve una lista de diccionarios (por ser RPC, no se devuelve como una lista de objetos de un modelo ya que sería un tipo no soportado por XML-RPC) con la información de cada voto encontrado (cada diccionario).

En cuanto al binding, el código utiliza XML-RPC y realiza un binding dinámico (en específico, tardío, aunque esto no se menciona en las diapositivas de teoría), ya que `ServerProxy` permite que el cliente descubra y llame a los métodos del servidor en tiempo de ejecución sin necesidad de conocerlos previamente. En este caso, los métodos remotos se invocan dinámicamente sobre el proxy, lo que indica que el cliente no está ligado estáticamente a una interfaz fija, sino que resuelve las llamadas a procedimientos remotos en ejecución.

## Ejercicio número 7:

**Ejecuta los pasos descritos más arriba e incluye evidencias en la memoria de haberlos llevado a cabo. Ejecuta los test proporcionados con el proyecto sobre las vistas y comprueba que no devuelven errores. Adjunta en la memoria una captura de pantalla en la que se muestre el resultado de ejecutar los test (python manage.py test).**

El fichero `env` que utilizaremos será este (ver `RPCAPIBASEURL` configurado como se nos pide y `DATABASE_SERVER_URL` en la que ponemos la url de la base de datos de la VM1 para que los tests funcionen, aunque para el cliente no debería ser necesario):

```
$ env
P1-rpc-client > $ env
1  # IMPORTANT: this file should not be in a repository
2  # To remove a file named env from a Git repository
3  # but keep it in the source (local system), follow these steps:
4  # Remove the file from Git tracking but keep it locally
5  ## git rm --cached env
6  # Add 'env' to .gitignore (so it's not tracked again)
7  ## echo "env" >> .gitignore
8  # Commit the changes
9  ## git commit -m "Removed env from Git tracking and added to .gitignore"
10 # Push the changes to the remote repository
11 ## git push
12 # use sqlite 3
13 ## DATABASE_SERVER_URL=sqlite:///db.sqlite3
14 # use postgres
15
16 DATABASE_SERVER_URL='postgres://alumnodb:alumnodb@localhost:15432/voto'
17 # Cambiar la ip por la ip de la interfaz wifi de nuestro host si lo ejecutamos en la máquina virtual
18 # o dejarlo en localhost si lo ejecutamos en el host;
19 # y el puerto 15432 si la BD está en la VM1 y 5432 si la BD está en el host.
20 # Si no vamos a ejecutar los tests, podemos comentar esto y descomentar lo de
21 # configurar la base de datos de sqlite 3, porque aquí no se usa la base de datos
22
23 # The client does not need to store data in any database
24 # so let us define a sqlite in order to avoid warning messages
25
26 # Poner a True si estamos en un entorno de desarrollo
27 DEBUG=True
28
29 SECRET_KEY = 'django-insecure-alczftnjl1$y%xmK@5j(n*px43c8kxgi_ua4%khc+t7g)_s9d'
30
31 RPCAPIBASEURL='http://localhost:28000/votoAppRPCServer/rpc/'
32 # Cambiar la ip por la ip de la interfaz wifi de nuestro host si lo ejecutamos en la máquina virtual
33 # o dejarlo en localhost si lo ejecutamos en el host;
34 # y el puerto poner el en el que se esté ejecutando el servidor RPC en el host si lo tenemos en el host,
35 # o el puerto 28000 si el servidor RPC está en la VM2
36
```

Y ahora, como se dice en los pasos, probaremos que todo funciona en la url que se nos dice (no hacemos pruebas manuales de que funciona todo porque ya se hacen con los tests que ejecutaremos, y en apartados posteriores se verá que todo funciona en la VM):

```
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-client$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 25, 2025 - 21:36:14
Django version 4.2.13, using settings 'votoSite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

[25/Feb/2025 21:36:17] "GET /votoAppRPCClient/ HTTP/1.1" 200 991
[]
```

← → ↺ 127.0.0.1:8000/votoAppRPCClient/ ☆ 📁 👤

## Introduzca la Informacion Censo de la Persona que Vota (votoSite)

Número de DNI:

Nombre:

Fecha de Nacimiento:

Código de Autorización:

En cuanto a los tests de las vistas, se ejecutarán junto a los otros que haya con el siguiente comando (en el que se aprecia que todo funciona perfecto, salvo el warning de la carpeta staticfiles):

```
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-client$ python manage.py test
Found 17 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
/home/marco/Documents/SI2/si2_venv/lib/python3.12/site-packages/django/core/handlers/base.py:61: UserWarning: No directory at: /home/marco/Documents/SI2/p1/P1-rpc-client/staticfiles/
  mw_instance = middleware(adapted_handler)
..voto {'id': 12, 'idCircunscripcion': 'CIRC123', 'idMesaElectoral': 'MESA123', 'idProcesoElectoral': 'ELEC123', 'nombreCandidatoVotado': 'Candidate A', 'censo': '23', 'codigoRespuesta': '000', 'marcaTiempo': '2025-02-25 20:47:27.723237+00:00'}
.....num rows: 1
voto_id: 19
.....
-----
Ran 17 tests in 0.963s

OK
Destroying test database for alias 'default'...
```

El warning de la carpeta staticfiles se soluciona de manera sencilla igual que hicimos en uno de los ejercicios anteriores, que es haciendo collectstatic antes:

```
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-client$ python manage.py collectstatic --noinput
125 static files copied to '/home/marco/Documents/SI2/p1/P1-rpc-client/staticfiles', 375 post-processed.
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-client$ python manage.py test
Found 17 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..voto {'id': 23, 'idCircunscripcion': 'CIRC123', 'idMesaElectoral': 'MESA123', 'idProcesoElectoral': 'ELEC123', 'nombreCandidatoVotado': 'Candidate A', 'censo': '23', 'codigoRespuesta': '000', 'marcaTiempo': '2025-02-25 20:50:29.976154+00:00'}
.....num rows: 1
voto_id: 30
.....
-----
Ran 17 tests in 1.714s

OK
Destroying test database for alias 'default'...
```

## Ejercicio número 8:

Ejecuta los pasos descritos más arriba. Prueba a acceder desde el navegador del PC del laboratorio a la URL localhost:38000/votoAppRPCClient/ para comprobar que el despliegue de la aplicación es

correcto. Registra un voto, lístalo (tanto desde la aplicación usando testbd como desde un cliente SQL como DBeaver), y bórralo. Registra el voto tanto desde localhost:38000/votoAppRPCClient/ como desde localhost:38000/votoAppRPCClient/testbd. Incluye evidencias en la memoria en forma de capturas de pantalla.

Antes que nada, cabe mencionar que tenemos el repositorio en la VM2 y VM3 configurado ya de la P1A, con lo que no se incluye aquí como configurar todo, sino los cambios que hay que hacer en los ficheros que ya teníamos en la P1A. Además, se destaca que estamos usando el mismo directorio de trabajo p1base (en las VM) para guardar todos los proyectos de esta práctica por comodidad. Por esto las rutas pueden no ser lo que cabría esperar.

Aunque no lo pone en ningún lado, modificaremos el fichero ~/.bashrc en la VM3 para que la variable de entorno TARGET guarde la ruta al proyecto ya que se utiliza para que al hacerle push al repo se haga un migrate y un collectstatic. Nos queda así:

```
sudo loadkeys es
export PATH=/home/si2/venv/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin
export TARGET="/home/si2/repo/p1base/p1/P1-rpc-client"
```

La variable de entorno target se usa en /home/si2/repo/p1base.git/hooks/post-receive:

```
#!/bin/bash
GIT_WORK_TREE=/home/si2/repo/p1base git checkout -f

# Reiniciar Gunicorn
sudo systemctl restart gunicorn

python $TARGET/manage.py migrate
python $TARGET/manage.py collectstatic --noinput
```

Asu vez, tenemos que cambiar el fichero /etc/systemd/system/gunicorn.service para indicarle a Gunicorn dónde está nuestro proyecto:

```
[Unit]
Description=Gunicorn WSGI Application Server
After=network.target

[Service]
User=si2
Group=si2
WorkingDirectory=/home/si2/repo/p1base/p1/P1-rpc-client
Environment="PATH=/home/si2/venv/bin"
ExecStart=/home/si2/venv/bin/gunicorn --workers 1 --bind 0.0.0.0:8000 votoSite.wsgi:application
ExecReload=/bin/kill -s HUP $MAINPID
KillMode=mixed
TimeoutStopSec=5
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

Hacemos un push desde el host:

```
(si2 venv) marco@marco-laptop:~/Documents/SI2/p1/P1-rpc-client$ git push vm3 main
Enumerating objects: 37, done.
Counting objects: 100% (37/37), done.
Delta compression using up to 4 threads
Compressing objects: 100% (27/27), done.
Writing objects: 100% (27/27), 1.84 MiB | 8.92 MiB/s, done.
Total 27 (delta 19), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Operations to perform:
remote:   Apply all migrations: admin, auth, contenttypes, sessions
remote: Running migrations:
remote:   No migrations to apply.
remote:
remote: 0 static files copied to '/home/si2/repo/p1base/p1/P1-rpc-client/staticfiles', 125 unmodified, 349 post-processed.
To ssh://localhost:32022/~repo/p1base.git
   cbf38a0..dc1d0b0  main -> main
```

Ahora, creamos un env en el proyecto del cliente RPC en la VM3 y le ponemos esto (las ip que pone son la



de la interfaz wifi del host y el puerto es el de postgres de la VM1 visto desde el host para la base de datos (que lo ponemos para que no fallen los tests aunque no debería hacer falta desde el cliente); y el de Unicorn de la VM2 visto desde el host también para RPCAPIBASEURL):

```
## IMPORTANT: this file should not be in a repository
# To remove a file named env from a Git repository
# but keep it in the source (local system), follow these steps:
# Remove the file from Git tracking but keep it locally
## git rm --cached env
# Add 'env' to .gitignore (so it's not tracked again)
## echo "env" >> .gitignore
# Commit the changes
## git commit -m "Removed env from Git tracking and added to .gitignore"
# Push the changes to the remote repository
## git push
# use sqlite 3
## DATABASE_SERVER_URL=sqlite:///db.sqlite3
# use postgres

DATABASE_SERVER_URL='postgres://alumnodb:alumnodb@192.168.1.149:15432/voto'
# Cambiar la ip por la ip de la interfaz wifi de nuestro host si lo ejecutamos en la máquina virtual
# o dejarlo en localhost si lo ejecutamos en el host;
# y el puerto 15432 si la BD está en la VM1 y 5432 si la BD está en el host.
# Si no vamos a ejecutar los tests, podemos comentar esto y descomentar lo de
# configurar la base de datos de sqlite 3, porque aquí no se usa la base de datos

# The client does not need to store data in any database
# so let us define a sqlite in orden to avoid warning messages

# Poner a True si estamos en un entorno de desarrollo
DEBUG=True

SECRET_KEY = 'django-insecure-alczftnjl1#sv%xmkk@5j(n*px43c8kxgi_ua4%khc+t7g_)s9d'

RPCAPIBASEURL='http://192.168.1.149:28000/votoAppRPCServer/rpc/'
# Cambiar la ip por la ip de la interfaz wifi de nuestro host si lo ejecutamos en la máquina virtual
# o dejarlo en localhost si lo ejecutamos en el host;
# y el puerto poner el en el que se esté ejecutando el servidor RPC en el host si lo tenemos en el host,
# o el puerto 28000 si el servidor RPC está en la VM2
```

Aunque no se pide, ejecutaremos los tests para enseñar que va todo bien:

```
si2@si2-ubuntu-vm-3:~/repo/plbase/pl/P1-rpc-client$ python manage.py test
Found 17 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..voto {'id': 34, 'idCircunscripcion': 'CIRC123', 'idMesaElectoral': 'MESA123', 'idProcesoElectoral': 'ELEC123', 'nombreCandidatoVotado': 'Candidate A', 'censo': '23', 'codigoRespuesta': '000', 'marcaTiempo': '2025-02-25 21:07:42.713472+00:00'}
.....num rows: 1
voto_id: 41
.....
-----
Ran 17 tests in 2.525s

OK
Destroying test database for alias 'default'...
```

Ahora, para probar todo manualmente, buscamos un censo válido en Dbeaver:

	A-Z numeroDNI	A-Z nombre	A-Z fechaNacimiento	A-Z anioCenso	A-Z codigoAutorizacion
1	39739740E	Jose Moreno Locke	09/04/66	2025	729

Ahora probaremos a registrar un voto desde la url principal:



## Introduzca la Informacion Censo de la Persona que Vota (votoSite)

Número de DNI:

Nombre:

Fecha de Nacimiento:

Código de Autorización:

Se puede ver que el censo se verificó correctamente, con lo que introducimos los datos de nuestro voto:

## Registro de Voto (votoSite)

Introduzca los datos del nuevo voto a registrar:

ID Proceso Electoral:

ID Circunscripcion:

ID Mesa Electoral:

Nombre Candidato Votado:

Y al darle a enviar, nos sale que se registró correctamente con todos los datos bien:

## Voto Registrado con Éxito (votoSite)

Id: 45

Codigo Respuesta: 000

Marca Tiempo : 2025-02-25 21:13:58.776191+00:00

Id Circunscripcion : b

Id Mesa Electoral : c

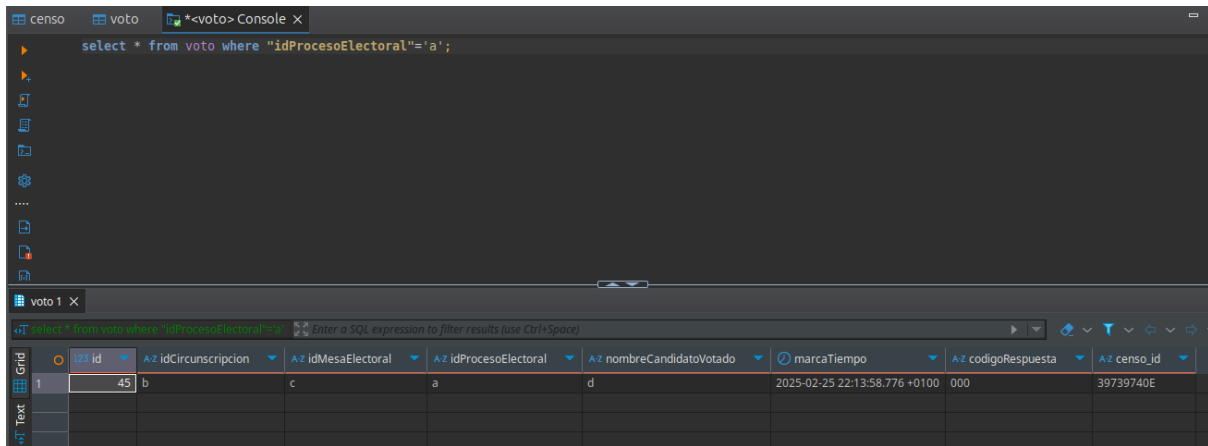
Id Proceso Electoral : a

Nombre Candidato Votado: d

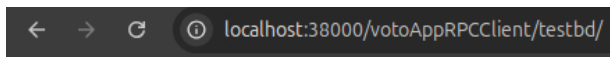
Si refrescamos la tabla voto en Dbeaver, vemos que se guardó en la BD correctamente:

	id	idCircunscripcion	idMesaElectoral	idProcesoElectoral	nombreCandidatoVotado	marcaTiempo	codigoRespuesta	censo_id
1	45	b	c	a	d	2025-02-25 22:13:58.776 +0100	000	39739740E

Podemos en Dbeaver listarlo mediante una consulta SQL también:



Al igual que desde testbd completando el campo de abajo con el valor a y pulsando el botón:



### Test Base de Datos: Registro de Voto (votoSite)

Introduzca los datos del nuevo voto a registrar:

ID Proceso Electoral:

ID Circunscripcion:

ID Mesa Electoral:

Nombre Candidato Votado:

Número de DNI:

Nombre:

Fecha de Nacimiento:

Código de Autorización:

### Test Base de Datos: Borrado de Voto

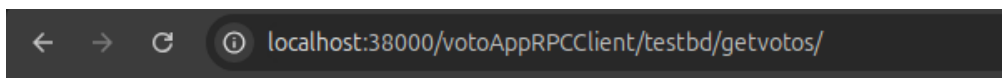
Introduzca el ID del voto a Borrar:

ID del Voto:

### Test Base de Datos: Listado de Votos

Introduzca el ID del proceso electoral:

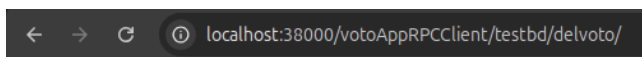
ID del Proceso Electoral:



### Votos Registrados (votoSite)

id	idCircunscripcion	idMesaElectoral	Candidato Votado	Marca Tiempo	Codigo Respuesta
45	b	c	d	2025-02-25 21:13:58.776191+00:00	000

Probamos ahora a eliminar el voto también desde testbd poniendo el id 45 (se ve en Dbeaver que es el id de nuestro voto):

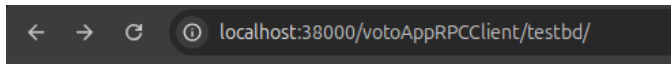


¡Voto eliminado correctamente!

En Dbeaver vemos ahora que se ha borrado correctamente tras refrescar:

id	idCircunscripcion	idMesaElectoral	idProcesoElectoral	nombreCandidatoVotado	marcaTiempo	codigoRespuesta	censo_id

Haremos ahora el mismo procedimiento pero registrando el voto desde testbd:



## Test Base de Datos: Registro de Voto (votoSite)

Introduzca los datos del nuevo voto a registrar:

ID Proceso Electoral:

ID Circunscripcion:

ID Mesa Electoral:

Nombre Candidato Votado:

Número de DNI:

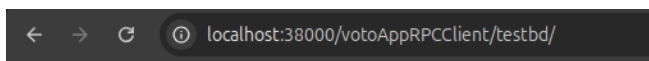
Nombre:

Fecha de Nacimiento:

Código de Autorización:

## Test Base de Datos: Borrado de Voto

Y al darle a enviar, nos sale que se registró correctamente con todos los datos bien:



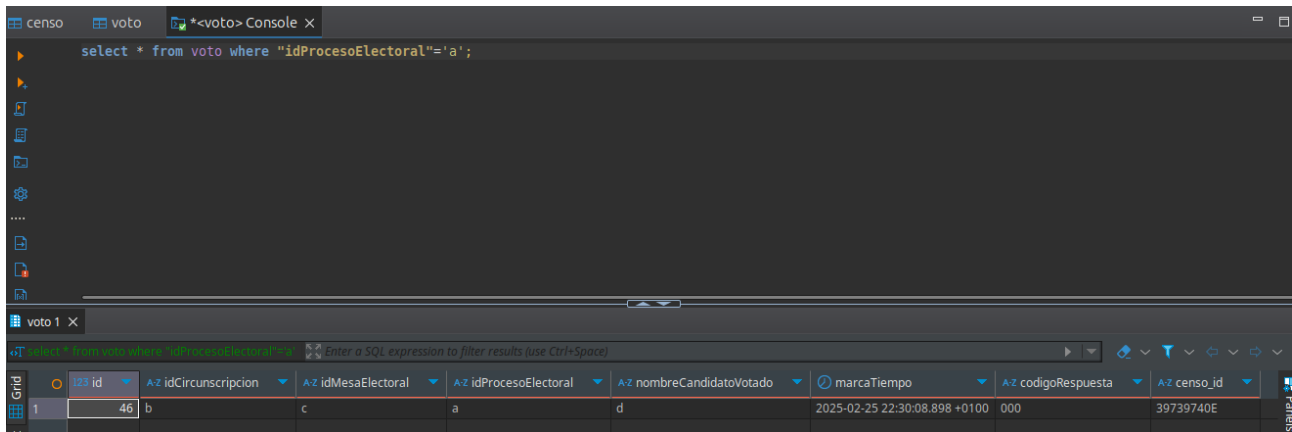
## Voto Registrado con Éxito (votoSite)

Id: 46  
Codigo Respuesta: 000  
Marca Tiempo : 2025-02-25 21:30:08.898425+00:00  
Id Circunscripcion : b  
Id Mesa Electoral : c  
Id Proceso Electoral : a  
Nombre Candidato Votado: d

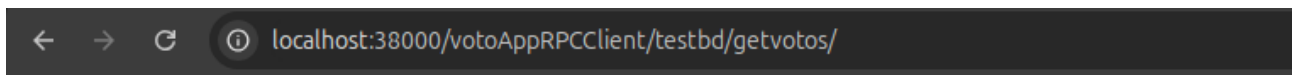
Si refrescamos la tabla voto en Dbeaver, vemos que se guardó en la BD correctamente:

	id	Az idCircunscripcion	Az idMesaElectoral	Az idProcesoElectoral	Az nombreCandidatoVotado	marcaTiempo	Az codigoRespuesta	Az censo_id
1	46	b	c	a	d	2025-02-25 22:30:08.898 +0100	000	39739740E

Podemos en Dbeaver listarlo mediante una consulta SQL también:



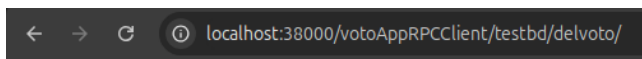
Al igual que desde testbd completando el campo de abajo con el valor a y pulsando el botón:



## Votos Registrados (votoSite)

id	IdCircunscripcion	IdMesaElectoral	Candidato Votado	Marca Tiempo	Codigo Respuesta
46	b	c	d	2025-02-25 21:30:08.898425+00:00	000

Probamos ahora a eliminar el voto también desde testbd poniendo el id 46 (se ve en Dbeaver que es el id de nuestro voto):



¡Voto eliminado correctamente!

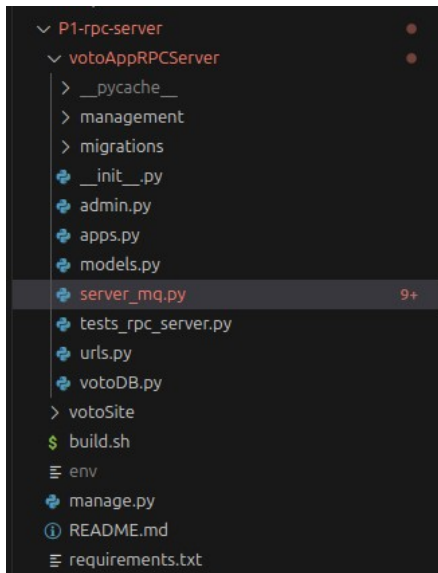
En Dbeaver vemos ahora que se ha borrado correctamente tras refrescar:

id	idCircunscripcion	idMesaElectoral	idProcesoElectoral	nombreCandidatoVotado	marcaTiempo	codigoRespuesta	censo_id

## Ejercicio número 9:

Ejecuta los pasos descritos más arriba. Incluye evidencias en la memoria en forma de capturas de pantalla del código desarrollado.

Aquí se puede ver lo que hemos creado (se destaca que hemos cambiado ligeramente el main que se nos daba para poder interrumpir el programa con Crtl+C sin que falle), que no comentaremos demasiado por su sencillez y porque ya se comenta en el propio código:

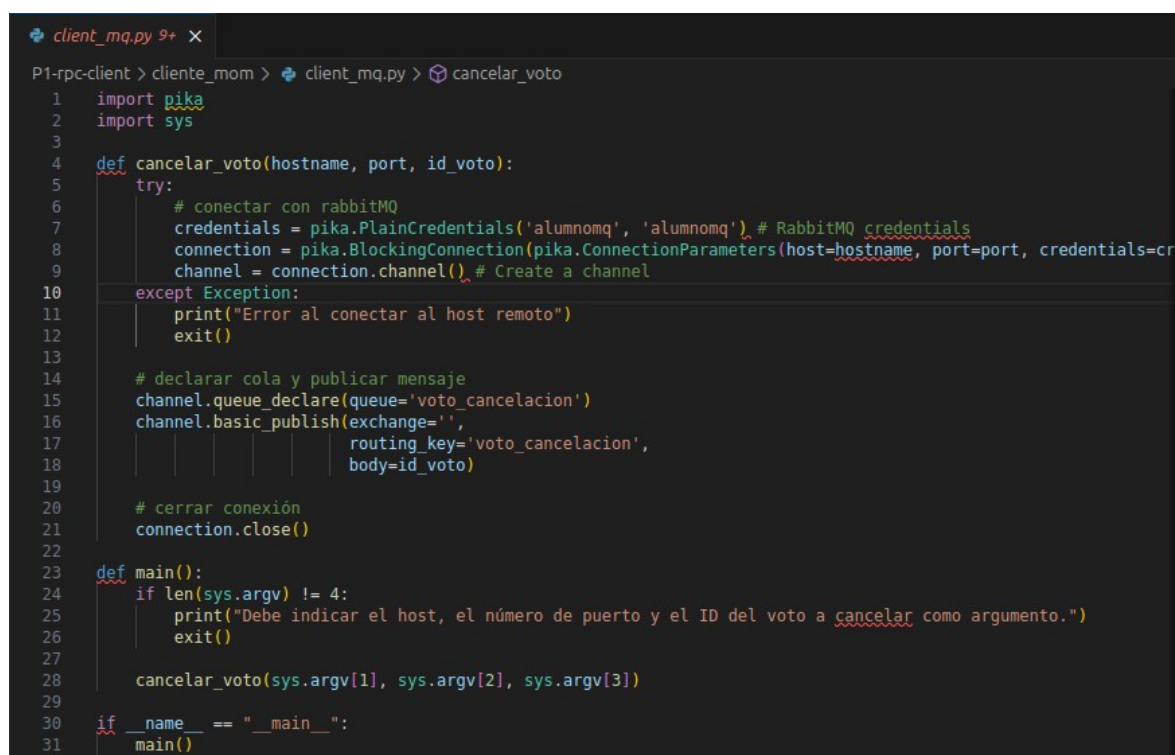
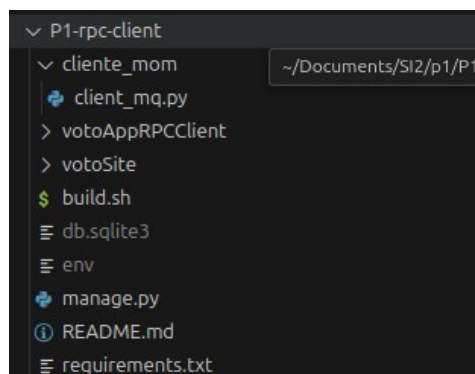


```
server_mq.py 9+ x
P1-rpc-server > votoAppRPCServer > server_mq.py > ...
1  # Uses RabbitMQ as the server
2
3  import os
4  import sys
5  import django
6  import pika
7
8  BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
9  sys.path.append(BASE_DIR)
10
11  os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'votoSite.settings')
12  django.setup()
13
14  from votoAppRPCServer.models import Censo, Voto
15
16  def main():
17      if len(sys.argv) != 3:
18          print("Debe indicar el host y el puerto")
19          exit()
20
21      hostname = sys.argv[1]
22      port = sys.argv[2]
23
24      credentials = pika.PlainCredentials('alumnomq', 'alumnomq') # RabbitMQ credentials
25      connection = pika.BlockingConnection(pika.ConnectionParameters(host=hostname, port=port, credentials=credentials)) # Connect to
26      channel = connection.channel() # Create a channel
27
28      channel.queue_declare(queue='voto_cancelacion') # Declare a queue
29
30      # Callback function to handle incoming messages
31      def callback(ch, method, properties, body):
32          voto_id = int(body.decode()) # Get the vote ID from the message
33          print(f"Received vote cancellation request for vote ID: {voto_id}")
34          try:
35              voto = Voto.objects.get(id=voto_id) # Get the vote object
36              voto.codigoRespuesta = '111' # Set the response code to '111'
37              voto.save(update_fields=['codigoRespuesta']) # Save the vote object
38              print(f"Vote ID {voto_id} cancelled successfully.")
39          except Voto.DoesNotExist:
40              print(f"Vote ID {voto_id} does not exist.")
41
42      channel.basic_consume(queue='voto_cancelacion', on_message_callback=callback, auto_ack=True) # Consume mess
43
44      print('Waiting for messages. To exit press CTRL+C')
45      channel.start_consuming() # Start consuming messages
46
47  if __name__ == '__main__':
48      try:
49          main()
50      except KeyboardInterrupt: # Handle keyboard interrupt
51          print('Interrupted')
52          try:
53              sys.exit(0)
54          except SystemExit:
55              os._exit(0)
```

## Ejercicio número 10:

Ejecuta los pasos descritos más arriba. Incluye evidencias en la memoria en forma de capturas de pantalla del código desarrollado.

Aquí se puede ver lo que hemos creado, que no comentaremos demasiado por su sencillez y porque ya se comenta en el propio código:



## Ejercicio número 11:

En este ejercicio probaremos a ejecutar el código desarrollado en los pasos anteriores.

1. Ejecuta el cliente del servicio de cancelación de votos en el PC del laboratorio. Para ello, escribir `python cliente_mq.py` seguido de la dirección IP, número de puerto de la VM1 para RabbitMQ, e identificador de voto a cancelar. De esta forma el servidor RabbitMQ que se usará será el de la VM1.
2. Como todavía no se ha arrancado el servidor de colas de mensajes, el mensaje enviado se almacenará en la cola. Probar a listar las colas y el número de mensajes en cada cola en la VM1 ejecutando el comando `sudo rabbitmqctl list_queues`. Comprobar que aparece la cola de mensajes asociada al servicio y que contiene un mensaje.
3. Ejecuta el servidor del servicio de cancelación de votos en la VM donde se esté ejecutando el servidor RPC. Debería ser la VM2. Para ello, escribir `python servidor_mq.py` seguido de la dirección IP y número de puerto para RabbitMQ en la VM1. De esta forma usaremos el servidor RabbitMQ que se está ejecutando en la VM1.
4. Comprobar que se ha cancelado el voto correspondiente en la base de datos. Para ello, probad a



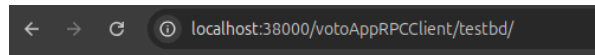
listar los votos asociados a dicho proceso electoral usando el cliente RPC ejecutándose en la VM3.

5. Probar a listar las colas y el número de mensajes de cada cola, disponibles en la VM1, de nuevo. Comprobar que ya no aparecen mensajes almacenados en la cola.

Incluid en la memoria capturas de pantallas y evidencias de haber llevado a cabo este ejercicio.

Se destaca que todo se ha hecho a través del cliente de la VM3.

Primero, registraremos un voto (con un censo existente que sacamos de DBeaver, como ya se vió) y veremos que no está cancelado:



### Test Base de Datos: Registro de Voto (votoSite)

Introduzca los datos del nuevo voto a registrar:

ID Proceso Electoral:

ID Circunscripcion:

ID Mesa Electoral:

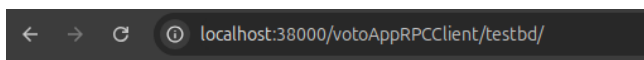
Nombre Candidato Votado:

Número de DNI:

Nombre:

Fecha de Nacimiento:

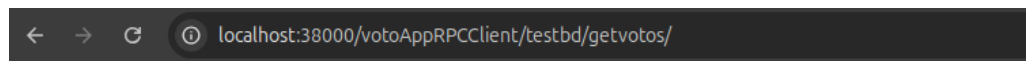
Código de Autorización:



### Voto Registrado con Éxito (votoSite)

Id: 41  
Codigo Respuesta: 000  
Marca Tiempo : 2025-03-05 21:11:04.277422+00:00  
Id Circunscripcion : b  
Id Mesa Electoral : c  
Id Proceso Electoral : a  
Nombre Candidato Votado: d

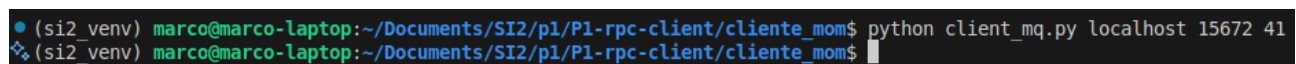
Vamos ahora a ver cómo sale su código de respuesta al listarlo si ponemos su id del proceso electoral (a) desde testbd (es 000 y no 111, que sería cancelado):



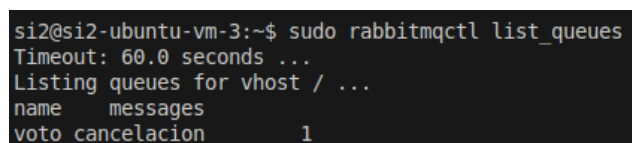
### Votos Registrados (votoSite)

id	IdCircunscripcion	IdMesaElectoral	Candidato Votado	Marca Tiempo	Codigo Respuesta
41	b	c	d	2025-03-05 21:11:04.277422+00:00	000

Ejecutamos el cliente en el host con el puerto de RabbitMQ de la VM1 visto desde el host, y no salta ningún error:



Ahora, listamos las colas desde la VM1, y vemos que sale la nuestra y que tiene un mensaje, que es el que el cliente acaba de crear y que todavía el servidor no ha consumido:



Ahora, ejecutamos el servidor (habiendo hecho un push antes) en la VM en la que se está corriendo el servidor RPC, que es la VM2 (la ip que pone es la de la interfaz wifi de nuestro host, y el puerto es, al igual que antes, el de RabbitMQ de la VM1 visto desde el host) (hacemos Ctrl+C para que pare después de ver que se ha cancelado el voto):

```

si2@si2-ubuntu-vm-3:~/repo/plbase/pl/P1-rpc-server/votoAppRPCServer$ python server_mq.py 192.168.1.149 15672
Waiting for messages. To exit press CTRL+C
Received vote cancellation request for vote ID: 41
Vote ID 41 cancelled successfully.
^CInterrupted

```

Se destaca que hemos hecho el programa para que para pararlo haya que pulsar Ctrl+C. Se puede ver que todo sale como debería.

Vemos ahora que listando el voto desde testbd (poniendo el proceso electoral, que es a), sale que se ha cancelado ya que su código de respuesta ha cambiado de 000 a 111, lo que nos muestra que ha cambiado en la base de datos como debería:

```
localhost:38000/votoAppRPCClient/testbd/getvotos/
```

### Votos Registrados (votoSite)

id	IdCircunscripcion	IdMesaElectoral	Candidato Votado	Marca Tiempo	Codigo Respuesta
41	b	c	d	2025-03-05 21:11:04.277422+00:00	111

Ahora, volvemos a ejecutar el siguiente comando en la VM1 para ver las colas y sus mensajes, y se puede ver que el servidor ha consumido correctamente el mensaje y ya no está en la cola (tiene 0 mensajes la cola):

```

si2@si2-ubuntu-vm-3:~$ sudo rabbitmqctl list_queues
Timeout: 60.0 seconds ...
Listing queues for vhost / ...
name      messages
voto_cancelacion  0

```