		<b>Escuela Politécnica Superior</b> <b>Ingeniería Informática</b> <b>Prácticas de Sistemas Informáticos 2</b>			
<b>Grupo</b>	<b>2313</b>	<b>Práctica</b>	1B	<b>Fecha</b>	09/03/2025
<b>Alumno/a</b>	Victorero, Lomas, Victoria				
<b>Alumno/a</b>	Martín, Burgos, Beatriz				

## Práctica 1B: RPC, Rabbit y Pika.

### Cuestión 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.

Evidencias de haber realizado los pasos descritos:

```

> P1-base
> P1-rpc-client
v P1-rpc-server
  > votoAppRPCServer
  > votoSite
  $ build.sh
  ≡ env
  + manage.py
  ⓘ README.md
  ≡ requirements.txt
  + tests_rpc_server.py
v P1-rpc-client

```

Como vemos, dentro de P1-rpc-server hemos copiado la carpeta P1-base. Además, hemos cambiado el nombre de la carpeta VotoApp a votoAppRPCServer.

```

> P1-base
> P1-rpc-client
v P1-rpc-server
  v votoAppRPCServer
    > __pycache__
    > management
    > migrations
    + __init__.py
    + admin.py
    + apps.py
    + models.py
    + read_1000_entries_from_db.py
    + urls.py
    + votoDB.py
  v votoSite
    > __pycache__
    + __init__.py
    + asgi.py
    + settings.py
    + urls.py
    + wsgi.py
    $ build.sh
    ≡ env
    + manage.py
    ⓘ README.md
    ≡ requirements.txt
    + tests_rpc_server.py

```

En esta captura, demostramos que hemos eliminado:

- votoAppRPCServer/forms.py
- votoAppRPCServer/views.py
- votoAppRPCServer/templates
- votoAppRPCServer/tests\_views.py
- votoAppRPCServer/tests\_models.py

```

35
36 ALLOWED_HOSTS = ['*']
37
38
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     'rest_framework',
50     "modernrpc",
51 ]
52
53 MODERNRPC_METHODS_MODULES = [
54     "votoAppRPCServer.votoDB"
55 ]

```

Por último, dentro de settings.py, hemos añadido en INSTALLED\_APPS “modernrpc” y MODERNRPC\_METHODS\_MODULES = [ "votoAppRPCServer.votoDB" ]

La razón por la que no se necesitan los formularios Django y las plantillas en el servidor RPC es que en un servidor RPC, la comunicación se realiza a través de llamadas a procedimientos remotos, donde el cliente envía solicitudes y el servidor responde con unos resultados determinados. En un servidor RPC, las funciones son invocadas desde el cliente, sin necesidad de disponer de una interfaz gráfica.

## Cuestión 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.**

A continuación, mostramos evidencias de la realización de los pasos descritos:

```

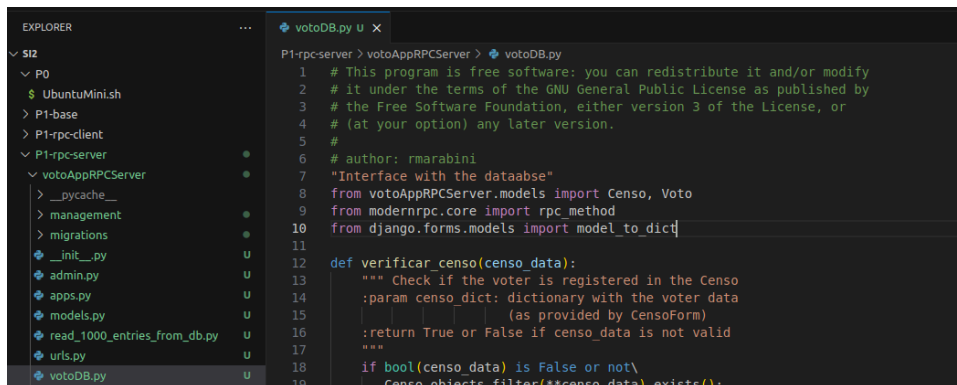
1  """
2  URL configuration for VotingProj project.
3
4  The 'urlpatterns' list routes URLs to views. For more information please see:
5  https://docs.djangoproject.com/en/4.2/topics/http/urls/
6  Examples:
7  Function views
8      1. Add an import: from my_app import views
9      2. Add a URL to urlpatterns: 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
18 from modernrpc.views import RPCEntryPoint
19
20 urlpatterns = [path("rpc/", RPCEntryPoint.as_view(), name="rpc")]
21

```

En primer lugar, hemos modificado el fichero urls.py que se encuentra en la carpeta VotoAppRPCServer.

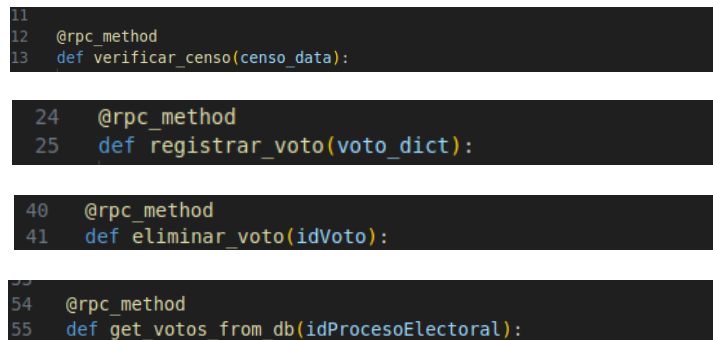
Seguidamente, hemos añadido 2 imports en el fichero votoDB.py.

- from modernrpc.core import rpc\_method
- from django.forms.models import model\_to\_dict



```
EXPLORER
v SI2
  P0
    $ UbuntuMinish
  > P1-base
  > P1-rpc-client
  > P1-rpc-server
    votoAppRPCServer
      > __pycache__
      > management
      > migrations
      $ __init__.py
      $ admin.py
      $ apps.py
      $ models.py
      $ read_1000_entries_from_db.py
      $ urls.py
      $ votoDB.py
P1-rpc-server > votoAppRPCServer > votoDB.py
1 # This program is free software: you can redistribute it and/or modify
2 # it under the terms of the GNU General Public License as published by
3 # the Free Software Foundation, either version 3 of the License, or
4 # (at your option) any later version.
5 #
6 # author: rmarabini
7 "Interface with the dataabase"
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 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     if bool(censo_data) is False or not\
19         Censo.objects.filter(**censo_data).exists():
```

A continuación, en el mismo fichero (votoDB.py), hemos añadido encima de cada función “@rpc\_method”:

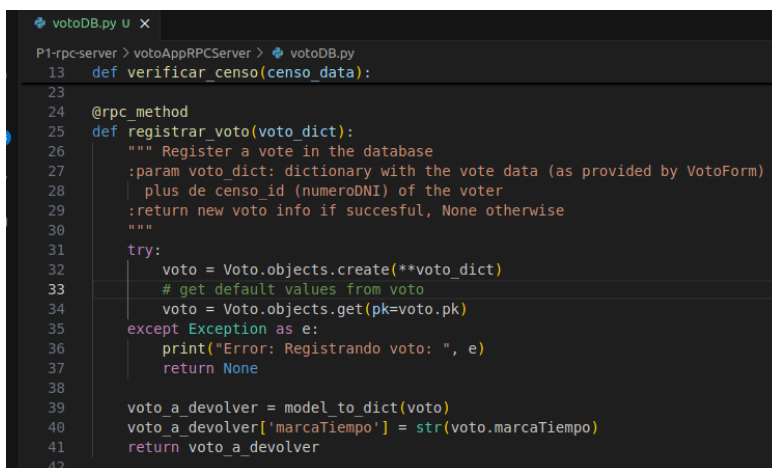


```
11
12 @rpc_method
13 def verificar_censo(censo_data):

24 @rpc_method
25 def registrar_voto(voto_dict):

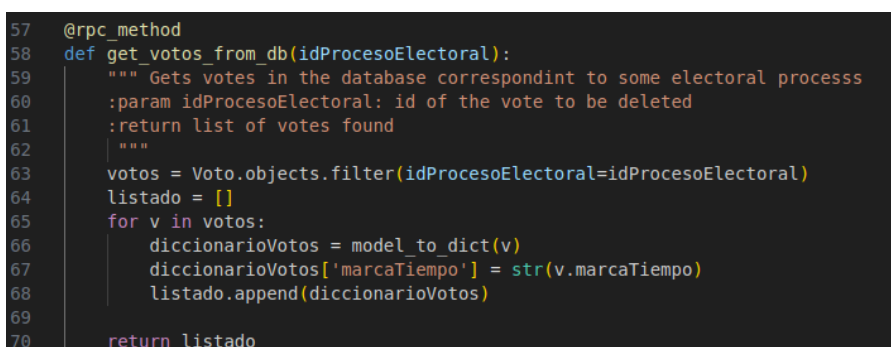
40 @rpc_method
41 def eliminar_voto(idVoto):

54 @rpc_method
55 def get_votos_from_db(idProcesoElectoral):
```



```
votoDB.py X
P1-rpc-server > votoAppRPCServer > votoDB.py
13 def verificar_censo(censo_data):
23
24 @rpc_method
25 def registrar_voto(voto_dict):
26     """ Register a vote in the database
27     :param voto_dict: dictionary with the vote data (as provided by VotoForm)
28                       plus de censo_id (numeroDNI) of the voter
29     :return new voto info if succesful, None otherwise
30     """
31     try:
32         voto = Voto.objects.create(**voto_dict)
33         # get default values from voto
34         voto = Voto.objects.get(pk=voto.pk)
35     except Exception as e:
36         print("Error: Registrando voto: ", e)
37         return None
38
39     voto_a_devolver = model_to_dict(voto)
40     voto_a_devolver['marcaTiempo'] = str(voto.marcaTiempo)
41     return voto_a_devolver
42
```

En la imagen de la izquierda, mostramos que también hemos modificado el método registrar\_voto, devolviendo un diccionario con el voto registrado.



```
57 @rpc_method
58 def get_votos_from_db(idProcesoElectoral):
59     """ Gets votes in the database correspondint to some electoral processs
60     :param idProcesoElectoral: id of the vote to be deleted
61     :return list of votes found
62     """
63     votos = Voto.objects.filter(idProcesoElectoral=idProcesoElectoral)
64     listado = []
65     for v in votos:
66         diccionarioVotos = model_to_dict(v)
67         diccionarioVotos['marcaTiempo'] = str(v.marcaTiempo)
68         listado.append(diccionarioVotos)
69
70     return listado
```

Al igual que en la imagen anterior, cambiamos la función get\_votos\_from\_db para que devuelva una lista de diccionarios.

La razón por la que es necesario utilizar `model_to_direct` es porque convierte un objeto en un diccionario, lo cual facilita la serialización y la transmisión de datos en un formato que el cliente podrá entender.

### Cuestión 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.

Evidencias de haber llevado a cabo los pasos descritos:

En primer lugar, comprobamos que en el fichero `env` tenemos la variable solicitada:

```
$ env
P1-rpc-server > $ 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 DATABASE_SERVER_URL='postgres://alumnodb:alumnodb@localhost:15432/voto'
16 #DATABASE_SERVER_URL='postgresql://alumnodb:npg_luUM7s6YbnpH@ep-wild-mountain-a8j6d
17 # The client does not need to store data in any database
18 # so let us define a sqlite in order to avoid warning messages
19 DEBUG=True
20 SECRET_KEY = 'django-insecure-alczftnjl1#5v%xmK@5j(n*px43c8kxgi_ua4%khc+t7g_)s9d'
21
22 RESTAPIBASEURL="http://10.0.2.15:8000/restapiserver/"
```

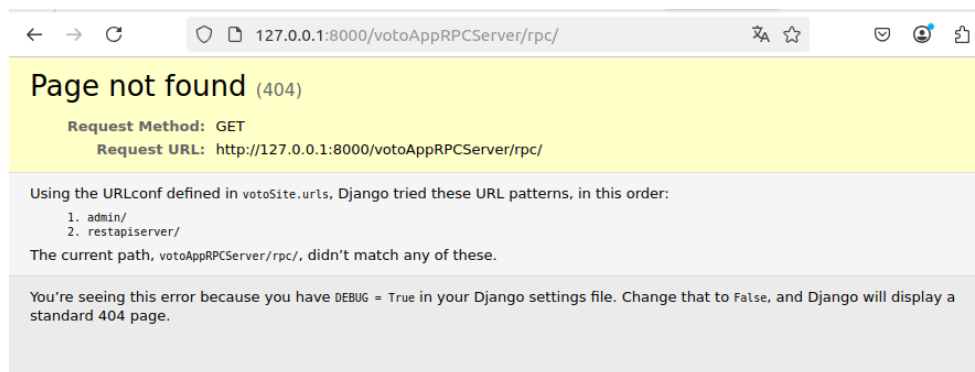
Tras ello, hemos eliminado la base de datos “voto” y la hemos vuelto a crear. Además, hemos ejecutado las migraciones y hemos poblado de nuevo la base de datos.

```
si2@si2-ubuntu-vm-1:~$ sudo -u postgres psql
psql (16.6 (Ubuntu 16.6-0ubuntu0.24.04.1))
Type "help" for help.

postgres=# DROP DATABASE voto;
DROP DATABASE
postgres=# CREATE DATABASE voto;
CREATE DATABASE
postgres=# \l

          List of databases
  Name | Owner  | Encoding | Locale Provider | Collate | Ctype  | ICU Locale | ICU Rules | Access privileges
-----+-----+-----+-----+-----+-----+-----+-----+-----
DB     | alumnodb | UTF8     | libc             | es_ES.UTF-8 | es_ES.UTF-8 |             |             |
postgres | postgres | UTF8     | libc             | es_ES.UTF-8 | es_ES.UTF-8 |             |             |
template0 | postgres | UTF8     | libc             | es_ES.UTF-8 | es_ES.UTF-8 |             |             |
template1 | postgres | UTF8     | libc             | es_ES.UTF-8 | es_ES.UTF-8 |             |             |
voto    | postgres | UTF8     | libc             | es_ES.UTF-8 | es_ES.UTF-8 |             |             |
(5 rows)
```

Una vez hecho esto, tenemos que ejecutar el comando “`python3 manage.py runserver`”. Una vez el servidor está corriendo, en el navegador ponemos :  
“<http://127.0.0.1:8000/votoAppRPCServer/rpc/>”, sin embargo observamos que obtenemos lo siguiente:



Como vemos, tenemos que cambiar la URL de VotoSite de la siguiente manera:

```
urls.py
P1-rpc-server > votoSite > urls.py
1 """
2 URL configuration for votoSite 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.contrib import admin
18 from django.urls import include, path
19
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path("votoAppRPCServer/", include("votoAppRPCServer.urls")),
23 ]
24
```

Una vez hemos realizado este cambio, volvemos a poner en el navegador:  
“<http://127.0.0.1:8000/votoAppRPCServer/rpc/>”, sin embargo, obtenemos lo esperado:

```
Method Not Allowed (GET): /votoAppRPCServer/rpc/
Method Not Allowed: /votoAppRPCServer/rpc/
[26/Feb/2025 23:28:47] "GET /votoAppRPCServer/rpc/ HTTP/1.1" 405 0
```

Ahora, realizaremos la segunda parte del ejercicio, donde se nos solicita ejecutar los tests y comprobar que no hay errores:

```
(venv) bmb_04@bmb-MS-7A63: ~/Escritorio/SI2/SI2/P1-rpc-server$ python manage.py test votoAppRPCServer.tests_rpc_server
Database URL: postgres://alumnodb:alumnodb@localhost:15432/voto
Found 4 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 4 tests in 0.022s

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

### Cuestión número 4:

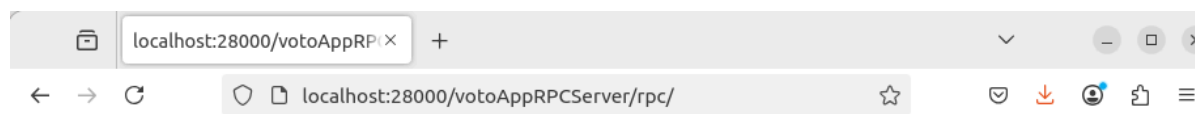
Ejecuta los pasos descritos más arriba. Prueba a acceder desde el navegador del PC del laboratorio a la URL `localhost:28000/votoAppRPCServer/rpc` para comprobar que el despliegue de la aplicación es correcto. Incluye evidencias en la memoria en forma de capturas de pantalla mostrando que el acceso a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

En primer lugar, vamos a correr en la máquina vm2 el servidor:

```
bmb_04@bmb-MS-7A63: ~/Escritorio/SI2/SI2/P1-rpc-server$ python3 manage.py runserver 0.0.0.0:8000
Database URL: postgres://alumnodb:alumnodb@192.168.1.58:15432/voto
Database URL: postgres://alumnodb:alumnodb@192.168.1.58:15432/voto
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 27, 2025 - 18:53:56
Django version 4.2.13, using settings 'votoSite.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CONTROL-C.
```

A continuación, buscamos en el navegador : "<http://localhost:28000/votoAppRPCServer/rpc>".



Además, al igual que en el ejercicio anterior, obtenemos un error 405:

```
bmb_04@bmb-MS-7A63: ~/Escritorio/SI2/SI2/P1-rpc-server$ python3 manage.py runserver 0.0.0.0:8000
Database URL: postgres://alumnodb:alumnodb@192.168.1.58:15432/voto
Database URL: postgres://alumnodb:alumnodb@192.168.1.58:15432/voto
Watching for file changes with StatReloader
Performing system checks...

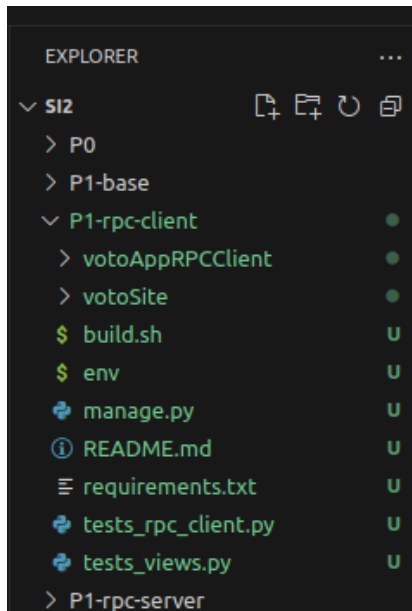
System check identified no issues (0 silenced).
February 27, 2025 - 18:53:56
Django version 4.2.13, using settings 'votoSite.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CONTROL-C.

Method Not Allowed (GET): /votoAppRPCServer/rpc/
Method Not Allowed: /votoAppRPCServer/rpc/
[27/Feb/2025 19:09:13] "GET /votoAppRPCServer/rpc/ HTTP/1.1" 405 0
```

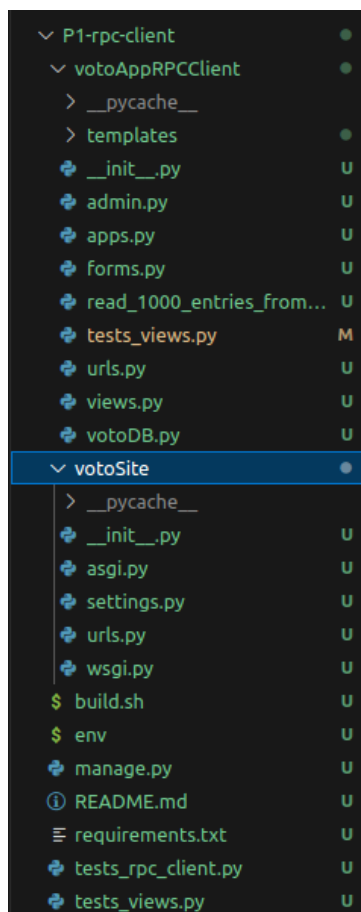
### Cuestión número 5:

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.

Vamos a mostrar las evidencias de haber realizado los pasos descritos.



En primer lugar, mostramos que hemos creado el proyecto partiendo de P1-base.



Seguidamente, hemos eliminado los ficheros solicitados:

- votoAppRPCClient/models.py
- votoAppRPCClient/migrations
- votoAppRPCClient/management
- votoAppRPCClient/tests\_models.py

Además, hemos añadido en el fichero venv la URL:

“`RPCAPIBASEURL=http://localhost:28000/votoAppRPCServer/rpc/`”

```
$ 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 DATABASE_SERVER_URL='postgres://alumnodb:alumnodb@192.168.1.52:15432/voto'
16 #DATABASE_SERVER_URL='postgresql://alumnodb:npg_luUM7s6YbnpH@ep-wild-mountain-a8j6d'
17 # The client does not need to store data in any database
18 # so let us define a sqlite in order to avoid warning messages
19 DEBUG=True
20 SECRET_KEY = 'django-insecure-alczftn)j1#$v%xmK@5j(n*px43c8kxgi_ua4%khc+t7g_)s9d'
21
22 RESTAPIBASEURL="http://10.0.2.15:8000/restapiserver/"
23
24 RPCAPIBASEURL=http://localhost:28000/votoAppRPCServer/rpc/
25
```

Y en el fichero settings.py hemos agregado la variable solicitada:

“`RPCAPIBASEURL = os.environ.get("RPCAPIBASEURL")`”

```
settings.py
P1-rpc-client > votoSite > settings.py
128
129 STATIC_URL = 'static/'
130
131 # Default primary key field type
132 # https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field
133
134 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
135
136 # store session in memory instead on in database
137 SESSION_ENGINE = "django.contrib.sessions.backends.cache"
138
139 CACHES = {
140     "default": {
141         "BACKEND": "django.core.cache.backends.locmem.LocMemCache",
142         "LOCATION": "unique-sessions", # Unique name for cache store
143     }
144 }
145
146 DATABASE_SERVER_URL = os.environ.get("DATABASE_SERVER_URL")
147 STATIC_ROOT = BASE_DIR / "staticfiles"
148
149 RPCAPIBASEURL = os.environ.get("RPCAPIBASEURL")
```

Finalmente, contestaremos a la pregunta de por qué no será necesario hacer uso de los modelos de datos Django en la aplicación cliente RPC.

La respuesta es que los modelos, se usan para interactuar con la base de datos, permitiendo definir



la estructura de los datos y realizar consultas, enviando solicitudes y recibiendo respuestas. Sin embargo, un cliente RPC, no almacena los datos localmente en una base de datos propia. En su lugar, el cliente se comunica con el servidor RPC, el cual maneja los datos y la base de datos.

Por lo tanto, el cliente y el servidor están desacoplados. El cliente no necesita conocer la estructura de la base de datos del servidor ni definir modelos para interactuar con ella. Solo necesita saber cómo comunicarse con el servidor RPC.

### ***Cuestión 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**

Para realizar este ejercicio, en primer lugar hemos añadido 2 imports al inicio del fichero:

```
votoDB.py u x
P1-rpc-client > votoAppRPCClient > votoDB.py
1  # This program is free software: you can redistribute it and/or modify
2  # it under the terms of the GNU General Public License as published by
3  # the Free Software Foundation, either version 3 of the License, or
4  # (at your option) any later version.
5  #
6  # author: rmarabini
7  "Interface with the dataabse"
8  from django.conf import settings
9  from xmlrpc.client import ServerProxy
10
```

Además, hemos modificado las funciones para que invoquen a los procedimientos remotos y hemos comentado cada función describiendo los aspectos solicitados:

```
votoDB.py u x
P1-rpc-client > votoAppRPCClient > votoDB.py
7  "Interface with the dataabse"
8  from django.conf import settings
9  from xmlrpc.client import ServerProxy
10
11 def verificar_censo(censo_data):
12     """ Esta funcion verifica si un votante esta registrado en el censo.
13     Para comprobarlo, llama al servidor RPC
14
15     :param censo_data: Diccionario que contiene los datos del votante
16     :return: True si está registrado y es correcto
17             False en caso contrario
18     """
19     with ServerProxy(settings.RPCAPIBASEURL) as proxy:
20         return proxy.verificar_censo(censo_data)
21
22
23 def registrar_voto(voto_dict):
24     """ Esta funcion registra un voto a traves del servidor RPC
25
26     :param voto_dict: Diccionario con los datos del voto
27     :return: Informacion del voto registrado si es exitoso
28             None en caso de error.
29     """
30     with ServerProxy(settings.RPCAPIBASEURL) as proxy:
31         return proxy.registrar_voto(voto_dict)
32
33
34 def eliminar_voto(idVoto):
35     """ Esta funcion elimina un voto en el servidor RPC
36
37     :param idVoto: ID del voto que se quiere eliminar
38     :return: True si se ha eliminado correctamente
39             False en caso contrario
40     """
41     with ServerProxy(settings.RPCAPIBASEURL) as proxy:
42         return proxy.eliminar_voto(idVoto)
43
44
45 def get_votos_from_db(idProcesoElectoral):
46     """ Esta funcion obtiene los votos de un proceso electoral desde el servidor RPC
47
48     :param idProcesoElectoral: ID del proceso electoral
49     :return: Lista de votos encontrados para ese ID
50     """
51     with ServerProxy(settings.RPCAPIBASEURL) as proxy:
52         return proxy.get_votos_from_db(idProcesoElectoral)
53
```

En las diapositivas de teoría, observamos que pueden ser 3 tipos: Estático, dinámico o automático.

En este caso, el cliente RPC utiliza binding dinámico. Esto significa que la conexión con el servidor RPC y la invocación de los procedimientos remotos se realizan en tiempo de ejecución. En este caso, estamos utilizando “ServerProxy” para conectarse dinámicamente al servidor RPC y llamar a los métodos remotos sin necesidad de conocer previamente la implementación específica de estos métodos.

### ***Cuestión 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).**

```
$ env U X
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 DATABASE_SERVER_URL='postgres://alumnodb:alumnodb@192.168.1.58:15432/voto'
16 #DATABASE_SERVER_URL='postgresql://alumnodb:npg_luUM7s6YbnpH@ep-wild-mountain-a8j6dy6y-pooler.eastus2.azure.neon.tech/voto?sslmode=re
17 # The client does not need to store data in any database
18 # so let us define a sqlite in orden to avoid warning messages
19 DEBUG=True
20 SECRET_KEY = 'django-insecure-alczftnjl#$v%xmK@5j(n*px43c8kxgi_ua4%khc+t7g_)s9d'
21
22 RESTAPIBASEURL="http://10.0.2.15:8000/restapiserver/"
23
24 RPCAPIBASEURL='http://localhost:28000/votoAppRPCServer/rpc/'
25
```

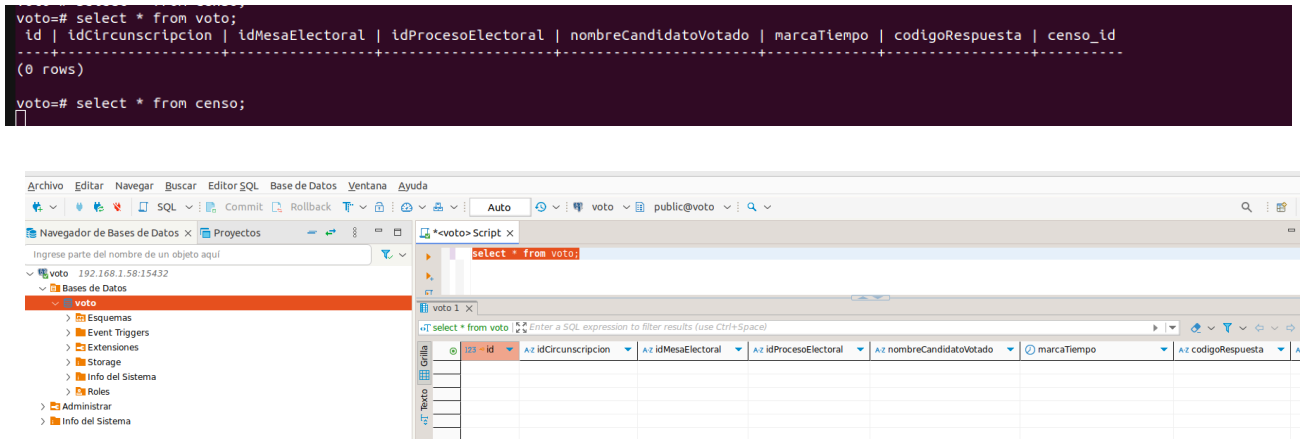
```
(venv) bmb_04@bmb-MS-7A63:~/Escritorio/S12/S12/P1-rpc-client$ python manage.py test
Database URL: postgres://alumnodb:alumnodb@192.168.1.58:15432/voto
Found 26 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..voto {'id': 30, 'idCircunscripcion': 'CIRC123', 'idMesaElectoral': 'MESA123', 'idProcesoElectoral': 'ELEC123', 'nombreCandidatoVotado': 'Candidate A', 'censo': '23', 'codigoRespuesta': '000', 'marcaTien
po': '2025-03-01 00:22:22.978344+00:00'}
.....num_rows: 1
voto_id: 45
.....num_rows: 1
voto_id: 49
.....
Ran 26 tests in 0.643s
OK
Destroying test database for alias 'default'...
```

## Cuestión 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

A continuación, vamos a probar si funciona accediendo a la URL “`localhost:38000/votoAppRPCClient/`”.

En primer lugar, comprobamos que no tengamos votos registrados, tanto en postgres como en dbeaver:



Seguidamente, vamos a registrar un voto:

The image shows a web browser at `localhost:38000/votoAppRPCClient/voto/` displaying the 'Registro de Voto (votoSite)' form. The form contains the following fields and values:

- ID Proceso Electoral: 66
- ID Circunscripcion: 87
- ID Mesa Electoral: 3
- Nombre Candidato Votado: Marcos Martinez

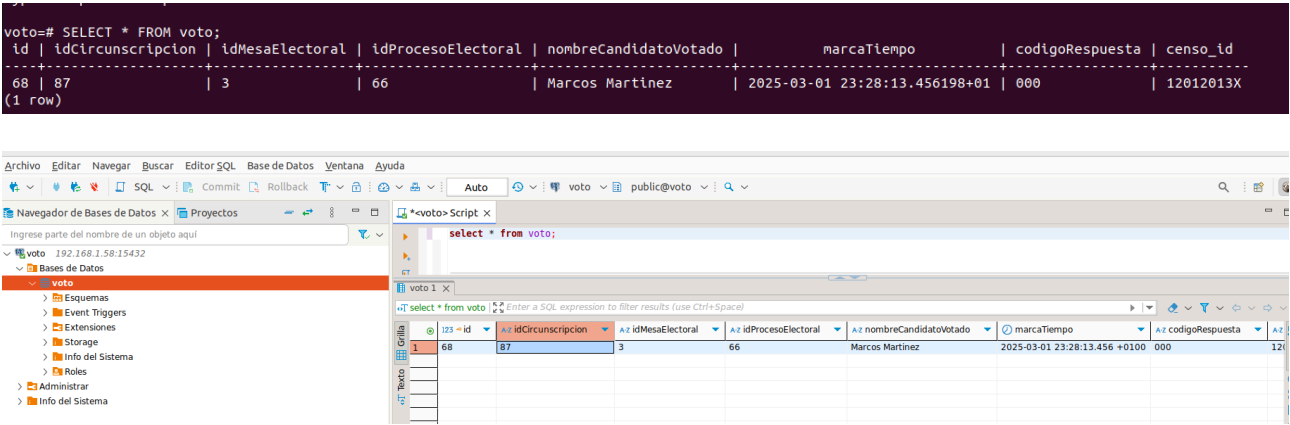
Below the form is a button labeled 'Enviar Información Voto'.

The image shows the same web browser at `localhost:38000/votoAppRPCClient/voto/` displaying the 'Voto Registrado con Éxito (votoSite)' page. The page shows the following details:

- Id: 68
- Codigo Respuesta: 000
- Marca Tiempo : 2025-03-01 22:28:13.456198+00:00
- Id Circunscripcion : 87
- Id Mesa Electoral : 3
- Id Proceso Electoral : 66
- Nombre Candidato Votado: Marcos Martinez

Una vez realizado este registro, en postgres encontramos que tenemos guardado 1 voto, tanto en postgres como en dbeaver:



Ahora, vamos a probar el funcionamiento con la URL :  
“localhost:38000/votoAppRPCClient/testbd”.

Vamos a registrar un voto:

localhost:38000/votoAppRPCClient/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:

localhost:38000/votoAppRPCClient/testbd/

### Voto Registrado con Éxito (votoSite)

Id: 69

Codigo Respuesta: 000

Marca Tiempo : 2025-03-01 22:31:41.957843+00:00

Id Circunscripcion : 14

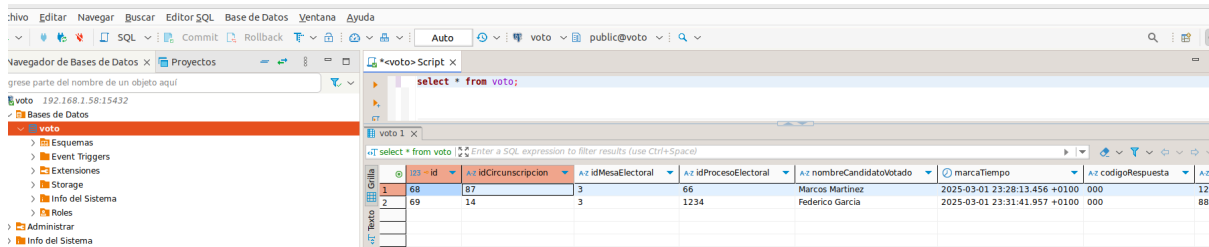
Id Mesa Electoral : 3

Id Proceso Electoral : 1234

Nombre Candidato Votado: Federico Garcia

Como podemos observar en las siguientes capturas, el voto se ha registrado correctamente.

```
voto=# SELECT * FROM voto;
id | idCircunscripcion | idMesaElectoral | idProcesoElectoral | nombreCandidatoVotado | marcaTiempo | codigoRespuesta | censo_id
-----+-----+-----+-----+-----+-----+-----+-----
68 | 87 | 3 | 66 | Marcos Martinez | 2025-03-01 23:28:13.456198+01 | 000 | 12012013X
69 | 14 | 3 | 1234 | Federico Garcia | 2025-03-01 23:31:41.957843+01 | 000 | 88888888M
(2 rows)
```

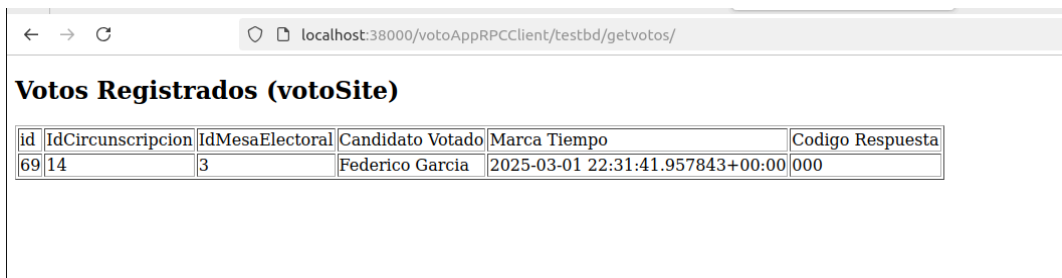


Además, vamos a listar los votos pertenecientes al proceso electoral con ID 1234.

## Test Base de Datos: Listado de Votos

Introduzca el ID del proceso electoral:

ID del Proceso Electoral:

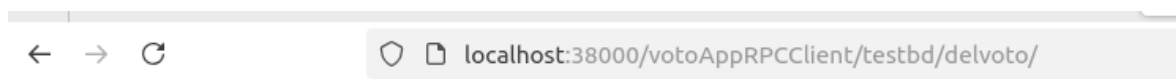


Por último, vamos a eliminar el voto con ID 69:

## Test Base de Datos: Borrado de Voto

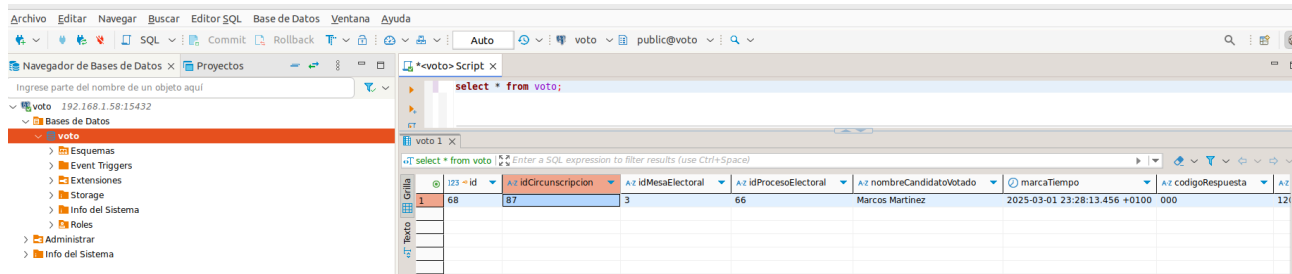
Introduzca el ID del voto a Borrar:

ID del Voto:



Como podemos comprobar en postgres y dbeaver, los votos se han eliminado correctamente.

```
voto=# SELECT * FROM voto;
id | idCircunscripcion | idMesaElectoral | idProcesoElectoral | nombreCandidatoVotado | marcaTiempo | codigoRespuesta | censo_id
-----+-----+-----+-----+-----+-----+-----+-----
68 | 87                | 3               | 66                 | Marcos Martínez      | 2025-03-01 23:28:13.456198+01 | 000             | 12012013X
(1 row)
```



## Cuestión número 9:

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

Lo primero se ha hecho es coger el código base aportado en el enunciado. Tras ello, se han incluido una serie de cambios.

Primero se han establecido las credenciales, el nombre de usuario es “alumnomq” y la contraseña “alumnomq”. Para ello se ha utilizado la función **pika.PlainCredentials()**. Después se ha creado la conexión con el servidor RabbitMQ con la función **pika.BlockingConnection()**, como argumentos recibe la IP y el port que recibe como argumentos y las credenciales establecidas anteriormente. Esto se guarda en connection, y sobre este objeto se invoca el método que crea el canal de comunicación **channel()**.

Por otro lado, se crea la cola de mensajes. La cola de mensajes se crea con la función **queue\_declare()** con el nombre “voto\_cancelacion”.

Es necesario crear la función **callback()**. Esta función procesa los mensajes recibidos y verifica la existencia del voto. En caso de que exista modifica el código respuesta del mensaje que se desea cancelar a “111” y devuelve un mensaje con el id del voto indicando que se pudo cancelar correctamente. En caso contrario, indicará que hubo un error al cancelar el código.

Por último, se utiliza la función **basic\_consume()**. En ella se indica el nombre de la cola de mensajes que se desea consumir, la función a ejecutar cada vez de que se reciba un mensaje y **auto\_ack=True** que indica a la función que no debe esperar confirmación manual para consumir los mensajes.

```

server_mq.py X
P1-rpc-server > votoAppRPCServer > server_mq.py
1  import os
2  import sys
3  import django
4  import pika
5
6  BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
7  sys.path.append(BASE_DIR)
8
9  os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'votoSite.settings')
10 django.setup()
11
12 from votoAppRPCServer.models import Censo, Voto
13
14 def main():
15     if len(sys.argv) != 3:
16         print("Debe indicar el host y el puerto")
17         exit()
18
19     hostname = sys.argv[1]
20     port = sys.argv[2]
21
22     # Configurar credenciales
23     credentials = pika.PlainCredentials('alumnomq', 'alumnomq')
24
25     connection = pika.BlockingConnection(
26         pika.ConnectionParameters(host=hostname, port=int(port), credentials=credentials)
27     )
28     channel = connection.channel()
29
30     channel.queue_declare(queue='voto_cancelacion', durable=True)
31
32     def callback(ch, method, properties, body):
33         try:
34             # Decodificar el mensaje recibido (que debería ser el ID del voto)
35             voto_id = body.decode()
36             print(f" [x] Se ha recibido el voto {voto_id}")
37

```

```

        # Buscar el voto en la base de datos
        voto = Voto.objects.filter(id=voto_id).first()

        if voto:
            # Modificar el código de respuesta del voto a '111' (cancelado)
            voto.codigoRespuesta = '111'
            voto.save()

            print(f" [x] El voto {voto_id} se ha cancelado correctamente.")
        else:
            print(f" [x] El voto {voto_id} no se ha encontrado.")

    except Exception as e:
        print(f" [x] Error al procesar el voto: {e}")

    channel.basic_consume(queue='voto_cancelacion', on_message_callback=callback, auto_ack=True)

    print(' [*] Waiting for messages. To exit press CTRL+C')
    channel.start_consuming()

if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)

```

## Cuestión número 10:

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

Tras incluir el código base del enunciado en el fichero `client_mq.py`, se han realizado algunas modificaciones.

Primero se ha creado la conexión con el servidor RabbitMQ de forma análoga a en el servidor. Para ello, se han creado las credenciales con la misma función. Al igual que en el servidor, `pika.ConnectionParameters()` recibe el host y puerto que se mandan como argumento y las credenciales creadas anteriormente. Sobre el objeto creado con esta función, se aplica el método `channel()`, para crear el canal de comunicación.

Con la función `queue_declare()` se crea la cola de mensajes con el nombre “voto\_cancelacion”. Además, el parámetro durable se pone a True para que la cola sea persistente si Rabbit se reinicia. Además, en la función `basic_publish()`, con la que se envía el mensaje, se incluye la línea de código `properties=pika.BasicProperties(delivery_mode=2)`, para que los mensajes también sean persistentes.

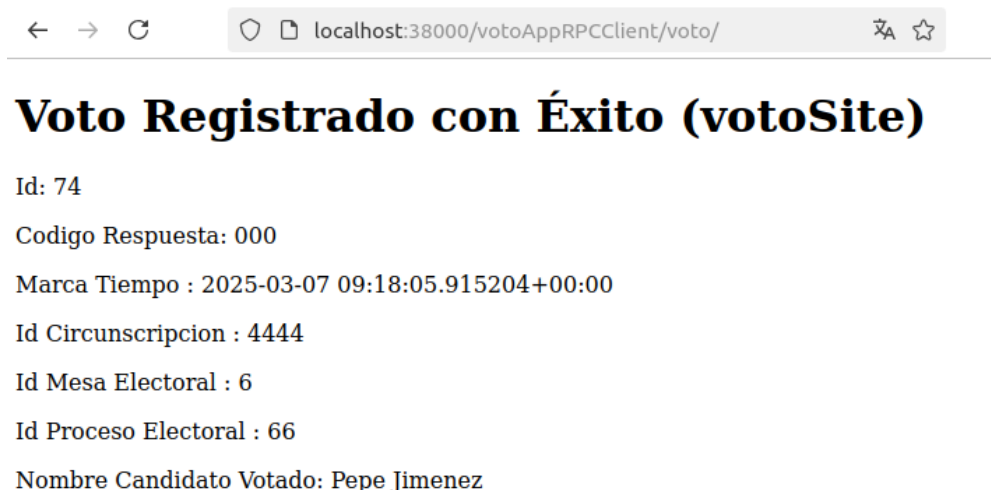
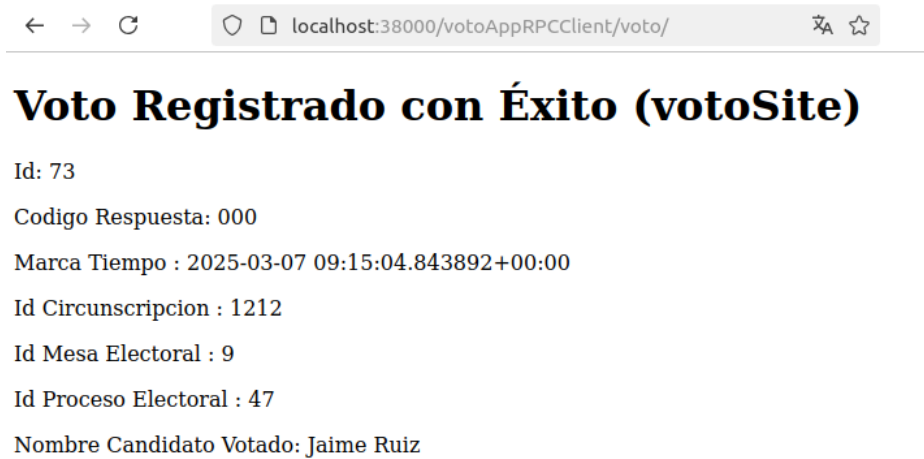
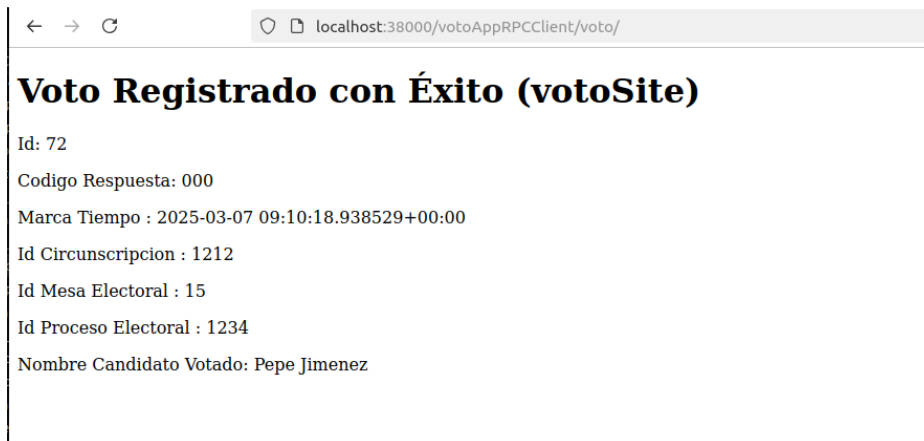
El último argumento recibido hace referencia al id del voto que se desea cancelar.

```
client_mq.py M X
P1-rpc-client > cliente_mom > client_mq.py
1 import pika
2 import sys
3
4 def cancelar_voto(hostname, port, id_voto):
5     try:
6         credentials = pika.PlainCredentials('alumnomq', 'alumnomq')
7         connection = pika.BlockingConnection(
8             pika.ConnectionParameters(host=hostname, port=int(port), credentials=credentials)
9         )
10        channel = connection.channel()
11    except Exception as e:
12        print(f"Error al conectar al host remoto: {e}")
13        exit()
14
15    channel.queue_declare(queue='voto_cancelacion', durable=True)
16
17    mensaje = f"{id_voto}"
18    channel.basic_publish(
19        exchange='',
20        routing_key='voto_cancelacion',
21        body=mensaje,
22        properties=pika.BasicProperties(delivery_mode=2) # Mensaje persistente
23    )
24
25    print(f" [X] Sent cancelar voto '{mensaje}'")
26    connection.close()
27
28 def main():
29     if len(sys.argv) != 4:
30         print("Debe indicar el host, el número de puerto y el ID del voto a cancelar.")
31         exit()
32
33     cancelar_voto(sys.argv[1], sys.argv[2], sys.argv[3])
34
35 if __name__ == "__main__":
36     main()
```



### ***Cuestión número 11:***

Para comprobar el correcto funcionamiento del código implementado en el ejercicio 9 y 10 primero se han creado tres votos.



Como se puede apreciar tanto en las imágenes de creación de cada uno de los votos como en la siguiente imagen, el código de respuesta de todos los votos es “000”.

```
voto=# select * from voto;
id | idCircunscripcion | idMesaElectoral | idProcesoElectoral | nombreCandidatoVotado | marcaTiempo | codigoRespuesta | censo_id
-----+-----+-----+-----+-----+-----+-----+-----
72 | 1212 | 15 | 1234 | Pepe Jimenez | 2025-03-07 10:10:18.938529+01 | 000 | 39739740E
73 | 1212 | 9 | 47 | Jaime Ruiz | 2025-03-07 10:15:04.843892+01 | 000 | 83583583L
74 | 4444 | 6 | 66 | Pepe Jimenez | 2025-03-07 10:18:05.915204+01 | 000 | 51151151X
(3 rows)
```

Desde el cliente (VM3) se envían tres solicitudes de cancelación de mensajes, una por cada voto creado anteriormente.

```
si2@si2-ubuntu-vm-3:~/P1-rpc-client/cliente_mom$ python3 client_mq.py 192.168.1.58 15672 72
[x] Sent cancelar voto '72'
si2@si2-ubuntu-vm-3:~/P1-rpc-client/cliente_mom$ python3 client_mq.py 192.168.1.58 15672 73
[x] Sent cancelar voto '73'
si2@si2-ubuntu-vm-3:~/P1-rpc-client/cliente_mom$ python3 client_mq.py 192.168.1.58 15672 74
[x] Sent cancelar voto '74'
```

Tras ello, en la VM1 se muestran las colas de mensajes de Rabbit con el comando `sudo rabbitmqctl`. Como se puede apreciar, la cola con nombre “voto\_cancelacion” tiene tres mensajes, lo que coincide con el número de solicitudes del cliente ya que el servidor todavía no ha consumido ninguna.

```
si2@si2-ubuntu-vm-1:~$ sudo rabbitmqctl list_queues
Timeout: 60.0 seconds ...
Listing queues for vhost / ...
name      messages
voto_cancelacion 3
```

Después, se ejecuta `server_mq.py` en el servidor (VM2). Tras esta ejecución se muestran los tres mensajes recibidos. Dado que los tres id existen, la cancelación se lleva a cabo correctamente.

```
si2@si2-ubuntu-vm-1:~/repo/pibase/P1-rpc-server/votoAppRPCServer$ python3 server_mq.py 192.168.1.58 15672
Database URL: postgres://alumnodb:alumnodb@192.168.1.58:15432/voto
[*] Waiting for messages. To exit press CTRL+C
[x] Se ha recibido el voto 72
[x] El voto 72 se ha cancelado correctamente.
[x] Se ha recibido el voto 73
[x] El voto 73 se ha cancelado correctamente.
[x] Se ha recibido el voto 74
[x] El voto 74 se ha cancelado correctamente.
```

Se vuelve de nuevo a la VM1 y con el mismo comando que en el caso anterior se listan las colas de mensajes de Rabbit. Como se puede apreciar, ya no hay mensajes ya que el servidor ya los ha recibido como se muestra en la imagen anterior.

```
si2@si2-ubuntu-vm-1:~$ sudo rabbitmqctl list_queues
Timeout: 60.0 seconds ...
Listing queues for vhost / ...
name      messages
voto_cancelacion 0
```

Por último, se muestra la base de datos. En este caso se pueden observar los tres mismos votos que

en el caso anterior, sin embargo, en este caso su nuevo código de respuesta es el “111”, lo que indica que la cancelación se llevó a cabo correctamente.

id	idCircunscripcion	idMesaElectoral	idProcesoElectoral	nombreCandidatoVotado	marcaTiempo	codigoRespuesta	censo_id
72	1212	15	1234	Pepe Jimenez	2025-03-07 10:24:16.040744+01	111	39739740E
73	1212	9	47	Jaime Ruiz	2025-03-07 10:24:16.043658+01	111	83583583L
74	4444	6	66	Pepe Jimenez	2025-03-07 10:24:16.045972+01	111	51151151X

(3 rows)