		Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2			
Grupo	2311	Práctica	1A	Fecha	22/02/2025
Alumno/a		Gómez, Hernández, Marco			
Alumno/a		Haya, de la Vega, Juan			

Práctica 1: API REST en Django

Antes que nada, queremos destacar que como no se dice explícitamente que al desplegar el proyecto en las máquinas virtuales haya que configurar todo completamente como si fuese un entorno de producción real, hemos dejado por ejmplo cosas como `DEBUG=True` y `ALLOWED_HOSTS=['*']` (este sin metelo en env) por comodidad para nosotros y ustedes y porque no sabemos si ustedes, los profesores, usarán su propio fichero env, con lo que no queríamos arriesgarnos a que tengan un env con menos variables de las que nuestro programa lee del fichero env. Debido a esto en los ficheros env aunque estén en las VM, pone `DEBUG=True` (aunque es tan fácil como ponerlo a false porque hemos cambiado el settings para que lo lea correctamente), o hacer los cambios necesarios en los settings.py o en el fichero env para tener todo como un entorno de producción real.

Ejercicio número 1:

Incluya en la memoria de la practica pruebas de que se ha desplegado la aplicación web localmente y de que esta funciona. Al menos se debe incluir una copia del contenido del chero env así como una captura de pantalla en la que se muestre el resultado de interaccionar con cada uno de los endpoints. Las capturas deben mostrar el URL al que se conecta el navegador.

El fichero env es el siguiente:

```
# 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@localhost:5432/voto'
# The client does not need to store data in any database
# so let us define a sqlite in orden to avoid warning messages
DEBUG=True
SECRET_KEY = 'django-insecure-alczftnjl#$v%xmK@5j(n*px43c8kxgi_ua4%khc+t7g_)s9d'
```

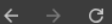
Tenemos este censo válido (lo vemos en DBeaver):

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

Cabe destacar que en estos pasos no se explica por qué las capturas mostradas demuestran que todo está

funcionando correctamente porque es obvio.

Accedemos al endpoint del censo y vemos que se muestran todo como debería, con lo que introducimos los datos de este censo para registrar un voto (se muestra tanto en la url que tiene censo al final como en la que no):

 127.0.0.1:8000/votoApp/

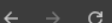
Introduzca la Informacion Censo de la Persona que Vota (votoSite)

Número de DNI:

Nombre:

Fecha de Nacimiento:

Código de Autorización:

 127.0.0.1:8000/votoApp/censo/

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


Número de DNI:

Nombre:

Fecha de Nacimiento:

Código de Autorización:

En ambas pantallas, al introducir datos correctos, nos permite registrar un voto:

 127.0.0.1:8000/votoApp/voto/

Registro de Voto (votoSite)

Introduzca los datos del nuevo voto a registrar:


ID Proceso Electoral:

ID Circunscripción:


ID Mesa Electoral:

Nombre Candidato Votado:

Pero si introducimos datos de un censo que no existe, nos sale (dependiendo de la url en la que estuviésemos):

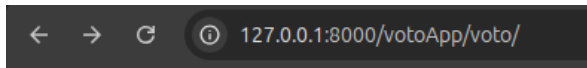
 127.0.0.1:8000/votoApp/censo/

¡Error: Votante no registrado en el Censo!

 127.0.0.1:8000/votoApp/

¡Error: Votante no registrado en el Censo!

Ahora, introducimos los datos de un voto (si hemos introducido un censo existente y que no ha votado ya):



Registro de Voto (votoSite)

Introduzca los datos del nuevo voto a registrar:

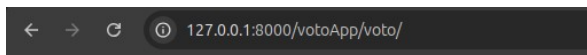
ID Proceso Electoral:

ID Circunscripcion:

ID Mesa Electoral:

Nombre Candidato Votado:

Y al enviar los datos, nos sale esto:



Voto Registrado con Éxito (votoSite)

Id: 3

Codigo Respuesta: 000

Marca Tiempo : Feb. 21, 2025, 8:26 p.m.

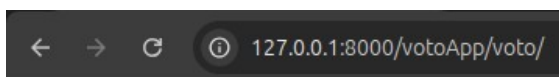
Id Circunscripcion : b

Id Mesa Electoral : c

Id Proceso Electoral : a

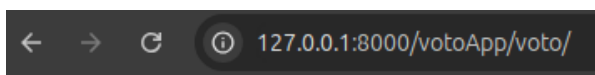
Nombre Candidato Votado: d

Probamos ahora a introducir el mismo censo válido y a registrar otro voto con el mismo id del proceso electoral tras esto para intentar que este tenga dos votos registrados en el mismo porceso electoral, y nos sale esto:



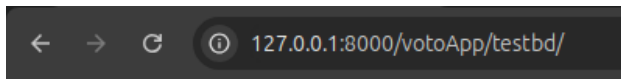
¡Error: al registrar voto!

O si intentamos registrar un voto sin haber accedido a esta url al verificar el censo (no hemos verificado ningún censo):



¡Error: DNI no encontrado en la sesión!

Ahora, accedemos al endpoint de 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

Introduzca el ID del voto a Borrar:

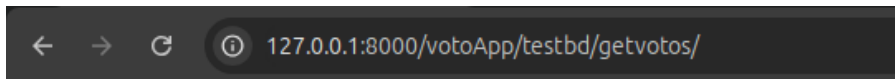
ID del Voto:

Test Base de Datos: Listado de Votos

Introduzca el ID del proceso electoral:

ID del Proceso Electoral:

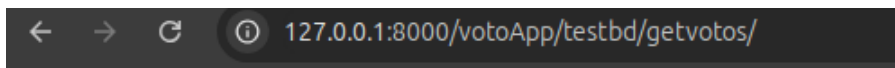
Vamos a probar a obtener el voto que hemos registrado, poniendo en el campo ID del Proceso Electoral 'a' (el del voto que hemos registrado):



Votos Registrados (votoSite)

id	IdCircunscripcion	IdMesaElectoral	Candidato Votado	Marca Tiempo	Codigo Respuesta
3	b	c	d	Feb. 21, 2025, 8:26 p.m.	000

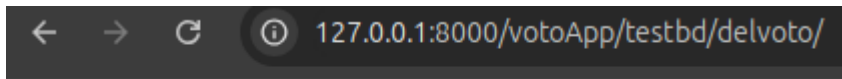
Si ponemos uno cualquiera no correspondiente a ningún voto:



Votos Registrados (votoSite)

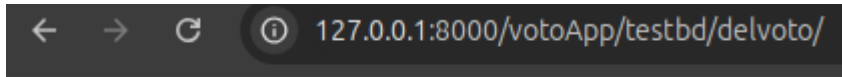
id	IdCircunscripcion	IdMesaElectoral	Candidato Votado	Marca Tiempo	Codigo Respuesta
----	-------------------	-----------------	------------------	--------------	------------------

Ahora probaremos a eliminar un voto, poniendo el id del registrado (3, como se mostró), y al enviar el formulario, nos sale esto:



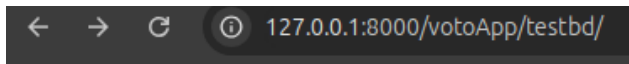
¡Voto eliminado correctamente!

Si probamos con un voto inexistente:



¡Error: al eliminar voto!

Ahora que hemos eliminado el voto, probamos a registrarlo de nuevo desde testbd:



Test Base de Datos: Registro de Voto (votoSitu

Introduzca los datos del nuevo voto a registrar:

ID Proceso Electoral:

ID Circunscripción:

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:

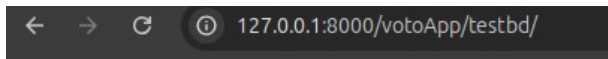
ID del Voto:

Test Base de Datos: Listado de Votos

Introduzca el ID del proceso electoral:

ID del Proceso Electoral:

Y al enviar los datos nos sale esto:



Voto Registrado con Éxito (votoSite)

Id: 5

Codigo Respuesta: 000

Marca Tiempo : Feb. 21, 2025, 8:34 p.m.

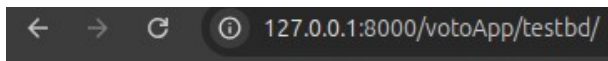
Id Circunscripcion : b

Id Mesa Electoral : c

Id Proceso Electoral : a

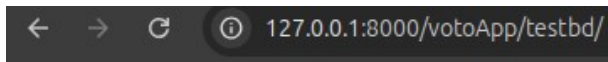
Nombre Candidato Votado: d

Y si intentamos volver a registrarlo (mismo censo y proceso electoral):



Error al registrar voto!

O introducimos datos de un censo inexistente:



¡Error: Votante no registrado en el Censo!

Ejercicio número 2:

Ejecuta los test proporcionados con el proyecto base 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). Los test proporcionados deben tomarse como requisitos extras del sistema.

Captura solicitada (el error que sale parece ser un problema del contenido de la base de datos, pero de eso se ha hecho todo con python manage.py populate, con lo que no parece un problema nuestro, y pone al final OK):

```
(si2 venv) marco@marco-laptop:~/Documents/SI2/p1/P1-base$ python manage.py test
Found 18 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..Error: Registrando voto: null value in column "censo_id" of relation "voto" violates not-null constraint
DETAIL: Failing row contains (1, C001, M001, P001, Candidato A, 2025-02-10 14:28:09.681587+00, 000, null).

.....
-----
Ran 18 tests in 0.198s

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

Ejercicio número 3:

Incluya en la memoria de la practica pruebas de que se ha desplegado la aplicación web P1-base usando las máquinas virtuales y de que esta funciona. Al menos se debe incluir una copia del contenido del fichero env así como una captura de pantalla en la que se muestre el resultado de registrar un voto. Esta última debe mostrar claramente el URL al que se conecta el navegador.

Cabe destacar que en estos pasos no se explica por qué las capturas mostradas demuestran que todo está funcionando correctamente porque es obvio.

Para poder registrar un voto, buscaremos un censo que exista con DBeaver y nos conectamos en el host al puerto de localhost correspondiente a Unicorn de la VM2 (28000). Cogemos este:

	A-Z numeroDNI	A-Z nombre	A-Z fechaNacimiento	A-Z anioCenso	A-Z codigoAutoriza
1	39739740E	Jose Moreno Locke	09/04/66	2025	729
2	83583583I	Restituta Sparrow Martin	31/08/73	2025	151

Nos conectamos al endpoint correspondiente e introducimos los datos de este censo válido:



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

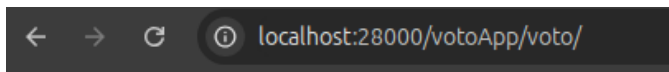
Número de DNI:

Nombre:

Fecha de Nacimiento:

Código de Autorización:

Introducimos los datos del voto en la página a la que se nos redirecciona (la de 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 se nos muestra esto:

localhost:28000/votoApp/voto/

Voto Registrado con Éxito (votoSite)

Id: 2

Codigo Respuesta: 000

Marca Tiempo : Feb. 16, 2025, 10:17 p.m.

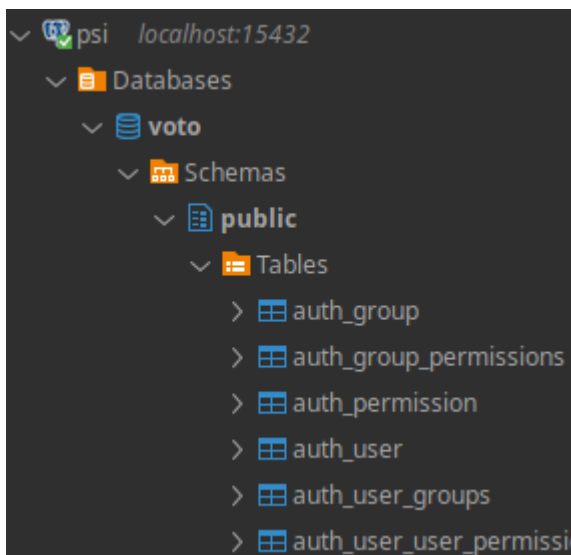
Id Circunscripcion : IDejemplo2

Id Mesa Electoral : IDejemplo3

Id Proceso Electoral : IDejemplo1

Nombre Candidato Votado: Nombre Ejemplo

El resultado se puede ver en Dbeaver (se ve aquí que es el puerto correspondiente a la base de datos de la VM1), en la tabla voto:



	id	idCircunscripc	idMesaElectc	idProcesoElec	nombreCandidatoVot	marcaTiempo	codigoRespuesta	censo_id
1	2	IDejemplo2	IDejemplo3	IDejemplo1	Nombre Ejemplo	2025-02-16 22:17:25.202 +0100	000	39739740E

El fichero env que tenemos en la VM2 es:

```
# 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@192.168.1.130:15432/voto'
# The client does not need to store data in any database
# so let us define a sqlite in order to avoid warning messages
DEBUG=True
SECRET_KEY = 'django-insecure-alczftnj1#$v%xmka5j(n*px43c8kxgi_ua4%khc+t7g_)s9d'
```

Cabe destacar que hemos puesto el puerto 15432 (puerto del host correspondiente a la base de datos en la VM1), y la ip es la ip de la interfaz de wifi de nuestro host (lo hicimos en nuestro ordenador personal).

Ejercicio número 4:

El script de python incluido en el apéndice C lee 1000 entradas de la base de datos voto una a una e imprime el tiempo de lectura. Ejecútalo desde el ordenador host y reporta en la memoria de la práctica los siguientes casos:

1. la base de datos está en la máquina virtual VM1.
2. la base de datos no esta en la red local. Crea un base de datos llamada voto en <http://www.neon.tech> y ejecuta el script contra esa base de datos
3. no se accede a la base de datos directamente. En Django, el mapeo de clases a tablas se implementa mediante el ORM (Object-Relational Mapping), reescribe el script para que los votos se lean usando el sistema de modelos de Django. Por ejemplo 'SELECT * FROM censo WHERE numeroDNI = %s' debería ser `Censo.objects.get(pk= %s)`. En este nuevo script no debe aparecer código SQL).

Repite cada medida 7 veces y reporta el valor medio de las mismas así como su desviación estándar. Además de los tiempos de lectura incluye en la memoria el script reescrito, el contenido del fichero env en cada caso así como un comentario sobre los resultados.

Resultados obtenidos:

Con la base de datos en la VM1:

```
(si2_venv) marco@marco-laptop:~/Documents/SI2/pl1$ python read_1000_entries_from_db.py
Tiempo invertido en buscar las 1000 entradas una a una: 0.532754 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/pl1$ python read_1000_entries_from_db.py
Tiempo invertido en buscar las 1000 entradas una a una: 0.530963 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/pl1$ python read_1000_entries_from_db.py
Tiempo invertido en buscar las 1000 entradas una a una: 0.531346 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/pl1$ python read_1000_entries_from_db.py
Tiempo invertido en buscar las 1000 entradas una a una: 0.642261 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/pl1$ python read_1000_entries_from_db.py
Tiempo invertido en buscar las 1000 entradas una a una: 0.572993 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/pl1$ python read_1000_entries_from_db.py
Tiempo invertido en buscar las 1000 entradas una a una: 0.626097 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/pl1$ python read_1000_entries_from_db.py
Tiempo invertido en buscar las 1000 entradas una a una: 0.541154 segundos
```

Estos tiempos dan una media de 0.568224 segundos. El fichero env utilizado para hacer la migración y poblar la base de datos es el siguiente (cambiamos sólo con respecto al env del ejercicio 1 el puerto para que sea el de postgres de la VM1 que ve el host, que es el 15432):

```
# 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@localhost:15432/voto'
# The client does not need to store data in any database
# so let us define a sqlite in orden to avoid warning messages
DEBUG=True
SECRET_KEY = 'django-insecure-alczftnjl#s$v%xmK@5j(n*px43c8kxgi_ua4%khc+t7g_)s9d'
```

Y usamos la configuración de la base de datos en el script siguiente:

```
# Configuración de la base de datos
db_config = {
    'dbname': 'voto', # Nombre de la base de datos
    'user': 'alumnodb', # Reemplaza con tu usuario de PostgreSQL
    'password': 'alumnodb', # Reemplaza con tu contraseña
    'host': 'localhost', # Cambia si el host es diferente
    'port': 15432, # Cambia si tu puerto es diferente
}
```

Con la base de datos fuera de la red local (en <http://www.neon.tech> en los servidores Azure Germany West Central (Frankfurt) que son los más cercanos):

```
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1$ python read_1000_entries_from_db.py
Tiempo invertido en buscar las 1000 entradas una a una: 44.508941 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1$ python read_1000_entries_from_db.py
Tiempo invertido en buscar las 1000 entradas una a una: 40.540339 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1$ python read_1000_entries_from_db.py
Tiempo invertido en buscar las 1000 entradas una a una: 44.327045 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1$ python read_1000_entries_from_db.py
Tiempo invertido en buscar las 1000 entradas una a una: 41.990620 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1$ python read_1000_entries_from_db.py
Tiempo invertido en buscar las 1000 entradas una a una: 40.613707 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1$ python read_1000_entries_from_db.py
Tiempo invertido en buscar las 1000 entradas una a una: 44.471027 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1$ python read_1000_entries_from_db.py
Tiempo invertido en buscar las 1000 entradas una a una: 43.673289 segundos
```

Estos tiempos dan una media de 42.87499542857143 segundos. El fichero env en este caso usado para hacer la migración y poblar la base de datos es el mismo que el apartado anterior de este ejercicio pero cambiando esto (la url de la base de datos por la que nos dan en la base creada en Neon):

```
DATABASE_SERVER_URL='postgresql://neondb_owner:npg_JYL40cw8xjBR@ep-steep-sky-a9s4mqw5-pooler.gwc.azure.neon.tech/voto?sslmode=require'
```

Y usamos esta configuración de la base de datos en el script basada en la URL que nos da neon que acabamos de decir:

```
# Configuración de la base de datos
db_config = {
    'dbname': 'voto', # Nombre de la base de datos
    'user': 'neondb_owner', # Reemplaza con tu usuario de PostgreSQL
    'password': 'npg_JYL40cw8xjBR', # Reemplaza con tu contraseña
    'host': 'ep-steep-sky-a9s4mqw5-pooler.gwc.azure.neon.tech', # Cambia si el host es diferente
    'port': 5432, # Cambia si tu puerto es diferente
    'sslmode': 'require' # Requerido por NeonDB para conexiones seguras
}
```

Si no accedemos a la base de datos directamente y lo hacemos mediante el sistema de consultas y de modelos de Django (aunque la base de datos la hemos puesto en la VM1 ya que en el enunciado no se especifica si debe estar ahí o en el host, y nos es más útil para comparar tiempos accediendo a la misma base de datos directa o indirectamente):

```
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-base$ python read_1000_entries_from_db_mine.py
Tiempo invertido en buscar las 1000 entradas una a una: 1.303767 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-base$ python read_1000_entries_from_db_mine.py
Tiempo invertido en buscar las 1000 entradas una a una: 1.318147 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-base$ python read_1000_entries_from_db_mine.py
Tiempo invertido en buscar las 1000 entradas una a una: 1.323519 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-base$ python read_1000_entries_from_db_mine.py
Tiempo invertido en buscar las 1000 entradas una a una: 1.289199 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-base$ python read_1000_entries_from_db_mine.py
Tiempo invertido en buscar las 1000 entradas una a una: 1.308386 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-base$ python read_1000_entries_from_db_mine.py
Tiempo invertido en buscar las 1000 entradas una a una: 1.420128 segundos
(si2_venv) marco@marco-laptop:~/Documents/SI2/p1/P1-base$ python read_1000_entries_from_db_mine.py
Tiempo invertido en buscar las 1000 entradas una a una: 1.308778 segundos
```

Estos tiempos dan una media de 1.3245605714285715 segundos. El fichero env en este caso usado para hacer la migración y poblar la base de datos es el mismo que el del apartado 1 de este ejercicio (por lo que hemos dicho de que hemos usado la base de datos en la VM1), y el script que hemos creado debe ejecutarse en el directorio raíz del proyecto y es este (le hemos puesto el mismo nombre que al otro script pero con `_mine` al final) (no comentamos más porque se comenta en el propio código):

```
# read the first 1000 entries
# then perform 1000 queries retrieving each one of the entries
# one by one. Measure the time required for the 1000 queries

import time
import os
import django

try:
    # Configurar el entorno de Django
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "votoSite.settings")
    django.setup()

    # Importar el modelo Censo
    from votoApp.models import Censo

    # Leer las claves primarias de las primeras 1000 entradas de Censo
    censo_pks = [censo_entrie.numeroDNI for
                 | censo_entrie in Censo.objects.all()[:1000]]

    # Medir el tiempo de inicio
    start_time = time.time()

    # Realizar búsquedas una a una
    for censo_pk in censo_pks:
        | Censo.objects.get(pk=censo_pk)

    # Medir el tiempo de finalización
    end_time = time.time()

    # Mostrar los resultados
    print("Tiempo invertido en buscar las 1000 entradas una a una: "
        | f"{end_time - start_time:.6f} segundos")

except Exception as e:
    print(f"Error: {e}")
```

Conclusiones:

Podemos ver que lo más rápido para acceder a la base de datos es que se encuentre en la VM1 y que accedamos directamente (0.57 segundos), luego lo es que la base de datos se encuentre en la VM1 y accedamos indirectamente con las consultas de django (1.32 segundos, más de el doble de rápido; probablemente debido a que supone más pasos para acceder a lo mismo), y por último lo más lento (por mucho) para acceder es tener la base de datos en Neon (42.87 segundos, probablemente porque la base de datos está muy lejos).

Cuestión número 1:

¿En el ejercicio anterior 7 repeticiones de cada medida proporcionan una estimación fiable?
¿Porqué?

Siete repeticiones de una medida de tiempo de acceso a una base de datos desde un ordenador de los laboratorios proporcionan una estimación **parcialmente fiable**, pero no del todo precisa. Aunque pueden dar una idea general del rendimiento, el tamaño de la muestra es pequeño y no captura variaciones causadas por factores como la carga del sistema, fluctuaciones en la red o cachés en la base de datos. Para obtener una estimación más robusta, sería recomendable aumentar el número de repeticiones, descartar valores atípicos y calcular medidas estadísticas como la media y la desviación estándar.

Ejercicio número 5:

Ejecuta los test proporcionados con el proyecto P1-base 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 incluyendo el comando utilizado para lanzarlos. En esta ejecución la base de datos debe estar en VM1 y la aplicación web en VM2.

Captura solicitada (el error que sale parece ser un problema del contenido de la base de datos, pero se ha poblado todo con python manage.py populate, con lo que no parece un problema nuestro, además de que se muestra un OK al final):

```
si2@si2-ubuntu-vm-3:~/repo/plbase/pl/P1-base$ python manage.py test
Found 18 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..Error: Registrando voto: null value in column "censo_id" of relation "voto" violates not-null constraint
DETAIL: Failing row contains (1, C001, M001, P001, Candidato A, 2025-02-16 20:26:07.689446+00, 000, null).

.....
-----
Ran 18 tests in 0.452s

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

Cabe destacar que nuestro repositorio contiene una carpeta llamada p1 que contiene la carpeta P1-base con el proyecto base en vez de estar directamente el proyecto base. Esto lo hemos hecho así para poder usar el mismo repositorio en todas las prácticas de la asignatura, por eso sale la ruta distinta a como cabría esperar.

Ejercicio número 6:

Ejecuta los test proporcionados con el proyecto P1-ws-server aplicación votoAppWSSEServer 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 junto con el comando utilizado. En esta ejecución la base de datos debe estar en VM1 y la aplicación web en VM2.

Captura solicitada:

```
si2@si2-ubuntu-vm-3:~/repo/plbase/pl/P1-ws-server$ python manage.py test
Found 6 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 6 tests in 0.868s

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

Ejercicio número 7:

Conéctate a los diferentes endpoints usando el navegador (<http://<hostname>:28000/restapiserver/xxxx/>) y muestra las pantallas resultantes de usar cada uno de ellos. Suministra al sistema datos válidos e inválidos. Junto a cada imagen debes incluir: (1) el URL al que te conectas, (2) el método usado (POST/GET/DELETE) y (3) en caso de usar el método POST los datos enviados en la petición.

Cabe destacar que la base de datos está poblada de cero para este apartado, y mencionar que no se irá explicando el contenido de las capturas y por qué es correcto porque es obvio (mirar los json de envío, los de respuesta y los códigos de las respuestas que salen en pantalla). También queremos recalcar que hay respuestas en caso de error no programadas por nosotros que saltan cuando el json enviado está en un formato incorrecto. Ya que con los casos que mostramos se puede ver que todo funciona correctamente en los casos de error razonables, no hemos mostrado estos otros que son más específicos y son externos a nuestro código.

Primero buscamos un censo válido en DBEaver:

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

Luego, ponemos estos datos en json en el endpoint de censo (datos de un censo):

localhost:28000/restapiserver/censo/

Django REST framework

Censo

API endpoint to collect censo information.
and check if the person is in the censo.

GET /restapiserver/censo/

HTTP 405 Method Not Allowed
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

```
{  "detail": "Method \"GET\" not allowed."}
```

Media type: application/json

Content: [{"numeroDNI": "39739740E", "nombre": "Jose Moreno Locke", "fechaNacimiento": "09/04/66", "codigoAutorizacion": "729"}]

POST

Tras ponerlos, pulsamos en POST y sale esto (nos llega un json que dice que los datos se han encontrado en la base de datos):

localhost:28000/restapiserver/censo/

Django REST framework

Censo

API endpoint to collect censo information.
and check if the person is in the censo.

OPTIONS

POST /restapiserver/censo/

HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "message": "Datos encontrados en Censo."
}
```

Media type: application/json

Content:

POST

Si, por el contrario, nos inventamos un censo:

localhost:28000/restapiserver/censo/

Django REST framework

Censo

API endpoint to collect censo information.
and check if the person is in the censo.

OPTIONS

GET /restapiserver/censo/

HTTP 405 Method Not Allowed
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "detail": "Method \"GET\" not allowed."
}
```

Media type: application/json

Content:

```
{\"numeroDNI\": \"39739740E\", \"nombre\": \"Maria\", \"fechaNacimiento\": \"09/04/66\", \"codigoAutorizacion\": \"729\"}
```

POST

Y pulsamos en POST, nos pone que no se encuentra en la base de datos, lo que es correcto:

localhost:28000/restapiserver/censo/

Django REST framework

Censo

Censo

API endpoint to collect censo information.
and check if the person is in the censo.

OPTIONS

POST /restapiserver/censo/

HTTP 404 Not Found
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

```
{  "message": "Datos no encontrados en Censo."}
```

Media type: application/json

Content:

POST

Si nos metemos en el endpoint de voto y ponemos los datos del voto en formato json (id del censo es el DNI del censo antes visto, que sabemos que existe en la base de datos):

localhost:28000/restapiserver/voto/

Django REST framework

Voto

Voto

API endpoint to collect voto information.
and save it in the database.

DELETE OPTIONS

GET /restapiserver/voto/

HTTP 405 Method Not Allowed
Allow: POST, DELETE, OPTIONS
Content-Type: application/json
Vary: Accept

```
{  "detail": "Method \"GET\" not allowed."}
```

Media type: application/json

Content:

```
{ "idCircunscripcion": "a", "idMesaElectoral": "b", "idProcesoElectoral": "c", "nombreCandidatoVotado": "Manuel Carrasco", "censo_id": "39739740E" }
```

POST

Y pulsamos en POST, nos sale la info del voto que se acaba de registrar para que sepamos que se ha hecho esto:

localhost:28000/restapiserver/voto/

Django REST framework

Voto

Voto

DELETE OPTIONS

API endpoint to collect voto information.
and save it in the database.

POST /restapiserver/voto/

HTTP 200 OK
Allow: POST, DELETE, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "id": 9,
  "idCircunscripcion": "a",
  "idMesaElectoral": "b",
  "idProcesoElectoral": "c",
  "nombreCandidatoVotado": "Manuel Carrasco",
  "marcaTiempo": "2025-02-21T18:38:12.826503+01:00",
  "codigoRespuesta": "000",
  "censo": "39739740E"
}
```

Media type: application/json

Content:

Si, por el contrario, ponemos un censo_id que no existe:

localhost:28000/restapiserver/voto/

Django REST framework

Voto

Voto

DELETE OPTIONS

API endpoint to collect voto information.
and save it in the database.

GET /restapiserver/voto/

HTTP 405 Method Not Allowed
Allow: POST, DELETE, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "detail": "Method \"GET\" not allowed."
}
```

Media type: application/json

Content:

POST

Y pulsamos en POST:

localhost:28000/restapiserver/voto/

Django REST framework

Voto

Voto

DELETEOPTIONS

API endpoint to collect voto information.
and save it in the database.

POST /restapiserver/voto/

HTTP 404 Not Found
Allow: POST, DELETE, OPTIONS
Content-Type: application/json
Vary: Accept

```
{  "message": "Entry not found in Censo."}
```

Media type: application/json

Content:

POST

O si dejamos alguna entrada del diccionario json vacía o volvemos a introducir el mismo voto (del mismo censo/votante y mismo proceso electoral, cosa que debería dar error porque un votante no puede votar más de una vez en el mismo proceso electoral), sale esto:

localhost:28000/restapiserver/voto/

Django REST framework

Voto

Voto

DELETEOPTIONS

API endpoint to collect voto information.
and save it in the database.

POST /restapiserver/voto/

HTTP 400 Bad Request
Allow: POST, DELETE, OPTIONS
Content-Type: application/json
Vary: Accept

```
{  "message": "Bad request, invalid data."}
```

Media type: application/json

Content:

POST

En cuanto a la eliminación de votos, primero probamos a poner la url de un voto inexistente:

localhost:28000/restapiserver/voto/2/

Django REST framework

Voto / Voto

Voto

DELETE OPTIONS

API endpoint to collect voto information.
and save it in the database.

GET /restapiserver/voto/2/

HTTP 405 Method Not Allowed
Allow: POST, DELETE, OPTIONS
Content-Type: application/json
Vary: Accept

```
{  "detail": "Method \"GET\" not allowed."}
```

Media type: application/json

Content:

POST

Y pulsamos en DELETE (se nos dice que el voto no existe):

localhost:28000/restapiserver/voto/2/

Django REST framework

Voto / Voto

Voto

DELETE OPTIONS

API endpoint to collect voto information.
and save it in the database.

DELETE /restapiserver/voto/2/

HTTP 404 Not Found
Allow: POST, DELETE, OPTIONS
Content-Type: application/json
Vary: Accept

```
{  "message": "Voto not found."}
```

Media type: application/json

Content:

POST

Pero si hacemos lo mismo (ir la url y pulsar DELETE) pero con un voto que existe (ponemos en la url el id 9, que es el del voto registrado, como se vió en una de las capturas anteriores):

localhost:28000/restapiserver/voto/9/

Django REST framework

Voto / Voto

Voto

API endpoint to collect voto information.
and save it in the database.

DELETE /restapiserver/voto/9/

HTTP 200 OK
Allow: POST, DELETE, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "message": "Voto deleted successfully."
}
```

Media type: application/json

Content:

POST

Ahora, registramos dos votos con el mismo idProcesoElectoral:

id	idCircunscripcion	idMesaElectoral	idProcesoElectoral	nombreCandidatoVotado	marcaTiempo	codigoRespuesta	censo_id
10	a	b	c	Manuel Carrasco	2025-02-21T19:05:49.502+0100	000	39739740E
11	l	m	c	Juan Haya	2025-02-21T19:06:32.873+0100	000	83583583L

Y nos metemos en el endpoint siguiente (con el id del proceso electoral común a estos votos) (que nos devuelve los votos de este proceso electoral, con id 'c') (esto es un GET aunque no se ve):

localhost:28000/restapiserver/procesoelectoral/c/

Django REST framework

Proceso Electoral

Proceso Electoral

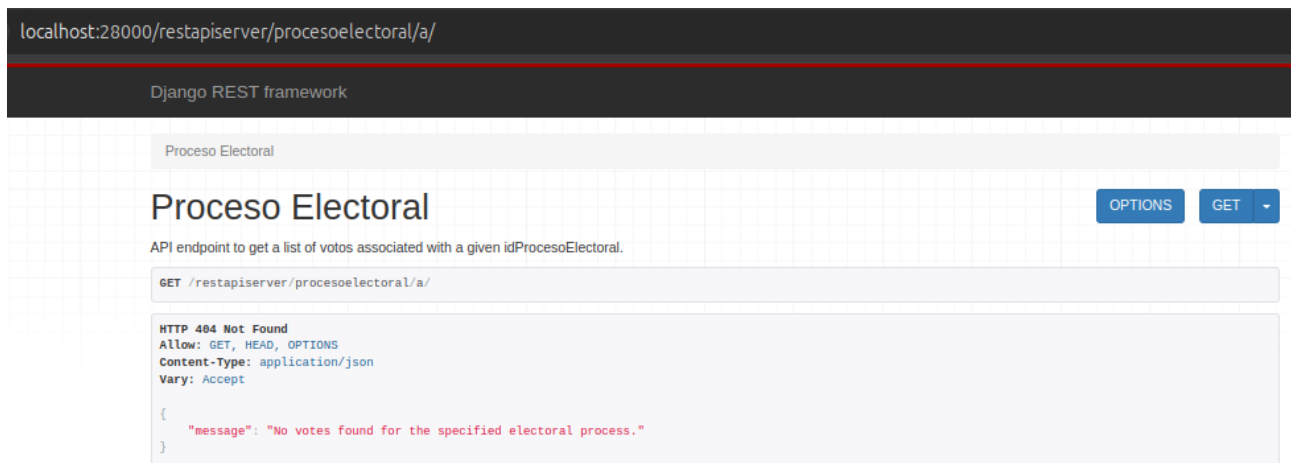
API endpoint to get a list of votos associated with a given idProcesoElectoral.

GET /restapiserver/procesoelectoral/c/

HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
[
  {
    "id": 10,
    "idCircunscripcion": "a",
    "idMesaElectoral": "b",
    "idProcesoElectoral": "c",
    "nombreCandidatoVotado": "Manuel Carrasco",
    "marcaTiempo": "2025-02-21T19:05:49.502721+01:00",
    "codigoRespuesta": "000",
    "censo": "39739740E"
  },
  {
    "id": 11,
    "idCircunscripcion": "l",
    "idMesaElectoral": "m",
    "idProcesoElectoral": "c",
    "nombreCandidatoVotado": "Juan Haya",
    "marcaTiempo": "2025-02-21T19:06:32.873270+01:00",
    "codigoRespuesta": "000",
    "censo": "83583583L"
  }
]
```

Si probamos con un id que no corresponde al proceso electoral de ningún voto:



No se ha mostrado el resultado de pulsar otros botones como OPTIONS o GET (cuando no debería poderse hacer nada con GET) porque no es relevante para nosotros, y tampoco se obtiene nada interesante o que no se muestre ya en lo que acabamos de ver.

Ejercicio número 8:

Ejecuta los test proporcionados con el proyecto P1-ws-client aplicación votoAppWSclient 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. En esta ejecución la base de datos debe estar en VM1 y la aplicación web en VM2.

Capturas solicitadas:

```
si2@si2-ubuntu-vm-3:~/repo/plbase/pl/P1-ws-client$ python manage.py test
Found 8 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
deleting votes
DELETE 2
This test only works if (1) http://192.168.1.130:28000/restapiserver/
server is up and running and (2) the database is populated.
Very likely in the future this test should be done with mocks
and not real calls to the server.
.deleting votes
DELETE 0
This test only works if (1) http://192.168.1.130:28000/restapiserver/
server is up and running and (2) the database is populated.
Very likely in the future this test should be done with mocks
and not real calls to the server.
.deleting votes
DELETE 0
This test only works if (1) http://192.168.1.130:28000/restapiserver/
server is up and running and (2) the database is populated.
Very likely in the future this test should be done with mocks
and not real calls to the server.
.deleting votes
DELETE 1
This test only works if (1) http://192.168.1.130:28000/restapiserver/
server is up and running and (2) the database is populated.
Very likely in the future this test should be done with mocks
and not real calls to the server.
INSERT 0 1
.deleting votes
DELETE 1
This test only works if (1) http://192.168.1.130:28000/restapiserver/
server is up and running and (2) the database is populated.
Very likely in the future this test should be done with mocks
and not real calls to the server.
INSERT 0 1
```

```

INSERT 0 1
.deleting votes
DELETE 0
This test only works if (1) http://192.168.1.130:28000/restapiserver/
server is up and running and (2) the database is populated.
Very likely in the future ths test should be done with mocks
and not real calls to the server.

INSERT 0 1
INSERT 0 1
INSERT 0 1
.deleting votes
DELETE 3
This test only works if (1) http://192.168.1.130:28000/restapiserver/
server is up and running and (2) the database is populated.
Very likely in the future ths test should be done with mocks
and not real calls to the server.

.deleting votes
DELETE 1
This test only works if (1) http://192.168.1.130:28000/restapiserver/
server is up and running and (2) the database is populated.
Very likely in the future ths test should be done with mocks
and not real calls to the server.

-----
Ran 8 tests in 2.237s

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

```

Ejercicio número 9:

Incluya en la memoria de la practica pruebas de que se ha desplegado las aplicaciones web P1-ws-client y P1-ws-server y de que estas funcionan. Al menos se debe incluir una copia del contenido del fichero env así como una captura de pantalla en la que se muestre el resultado de interaccionar con cada uno de los endpoints del cliente. Las capturas deben mostrar el URL al que se conecta el navegador.

Borramos todos los votos registrados de anteriores apartados, y tenemos este censo válido (lo vemos en DBeaver):

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

Cabe destacar que en estos pasos no se explica por qué las capturas mostradas demuestran que todo está funcionando correctamente porque es obvio.

Accedemos al endpoint del censo y vemos que se muestran todo como debería, con lo que introducimos los datos de este censo para registrar un voto (se muestra tanto en la url que tiene censo al final como en la que no):

← → ↻ localhost:38000/votoAppWSClient/

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

Número de DNI:

Nombre:

Fecha de Nacimiento:

Código de Autorización:

localhost:38000/votoAppWSClient/censo/

Introduzca la Información Censo de la Persona que Vota (votoSite)

Número de DNI:

Nombre:

Fecha de Nacimiento:

Código de Autorización:

En ambas pantallas, al introducir datos correctos, nos permite registrar un voto:

localhost:38000/votoAppWSClient/voto/

Registro de Voto (votoSite)

Introduzca los datos del nuevo voto a registrar:

ID Proceso Electoral:

ID Circunscripción:

ID Mesa Electoral:

Nombre Candidato Votado:

Pero si introducimos datos de un censo que no existe, nos sale (dependiendo de la url en la que estuviésemos):

localhost:38000/votoAppWSClient/

¡Error: Votante no registrado en el Censo!

localhost:38000/votoAppWSClient/censo/

¡Error: Votante no registrado en el Censo!

Ahora, introducimos los datos de un voto (si hemos introducido un censo existente y que no ha votado ya):

localhost:38000/votoAppWSClient/voto/

Registro de Voto (votoSite)

Introduzca los datos del nuevo voto a registrar:

ID Proceso Electoral:

ID Circunscripción:

ID Mesa Electoral:

Nombre Candidato Votado:

Y al enviar los datos, nos sale esto:

← → ↻ ⓘ localhost:38000/votoAppWSClient/voto/

Voto Registrado con Éxito (votoSite)

Id: 14

Codigo Respuesta: 000

Marca Tiempo : 2025-02-21T19:50:24.015644+01:00

Id Circunscripcion : b

Id Mesa Electoral : c

Id Proceso Electoral : a

Nombre Candidato Votado: d

Probamos ahora a introducir el mismo censo válido y a registrar otro voto con el mismo id del proceso electoral tras esto para intentar que este tenga dos votos registrados en el mismo porceso electoral, y nos sale esto:

← → ↻ ⓘ localhost:38000/votoAppWSClient/voto/

¡Error: al registrar voto!

O si intentamos registrar un voto sin haber accedido a esta url al verificar el censo (no hemos verificado ningún censo):

← → ↻ ⓘ localhost:38000/votoAppWSClient/voto/

¡Error: DNI no encontrado en la sesión!

Ahora, accedemos al endpoint de testbd:

← → ↻ ⓘ localhost:38000/votoAppWSClient/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

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:

Vamos a probar a obtener el voto que hemos registrado, poniendo en el campo ID del Proceso Electoral 'a' (el del voto que hemos registrado):

← → ↻ ⓘ localhost:38000/votoAppWSClient/testbd/getvotos/

Votos Registrados (votoSite)

Id	IdCircunscripcion	IdMesaElectoral	Candidato Votado	Marca Tiempo	Codigo Respuesta
14	b	c	d	2025-02-21T19:50:24.015644+01:00	000

Si ponemos uno cualquiera no correspondiente a ningún voto:

← → ↻ ⓘ localhost:38000/votoAppWSClient/testbd/getvotos/

Votos Registrados (votoSite)

Id	IdCircunscripcion	IdMesaElectoral	Candidato Votado	Marca Tiempo	Codigo Respuesta
----	-------------------	-----------------	------------------	--------------	------------------

Ahora probaremos a eliminar un voto, poniendo el id del registrado (14, como se mostró), y al enviar el formulario, nos sale esto:

← → ↻ ⓘ localhost:38000/votoAppWSClient/testbd/delvoto/

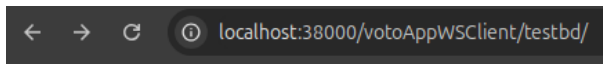
¡Voto eliminado correctamente!

Si probamos con un voto inexistente:

← → ↻ ⓘ localhost:38000/votoAppWSClient/testbd/delvoto/

¡Error: al eliminar voto!

Ahora que hemos eliminado el voto, probamos a registrarlo de nuevo 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

Introduzca el ID del voto a Borrar:

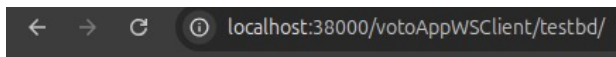
ID del Voto:

Test Base de Datos: Listado de Votos

Introduzca el ID del proceso electoral:

ID del Proceso Electoral:

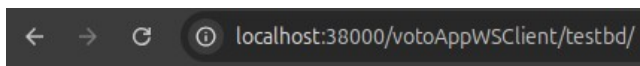
Y al enviar los datos nos sale esto:



Voto Registrado con Éxito (votoSite)

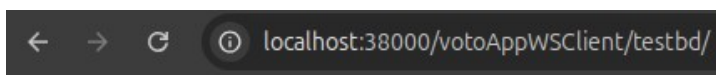
Id: 15
Codigo Respuesta: 000
Marca Tiempo : 2025-02-21T20:05:41.080747+01:00
Id Circunscripcion : b
Id Mesa Electoral : c
Id Proceso Electoral : a
Nombre Candidato Votado: d

Y si intentamos volver a registrarlo (mismo censo y proceso electoral):



Error al registrar voto!

O introducimos datos de un censo inexistente:



¡Error: Votante no registrado en el Censo!

El fichero env utilizado en el servidor (API REST) en la VM2 es (la ip es la de la interfaz wifi del host y el puerto de la base de datos es el de la base de datos de la VM1 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@192.168.1.130:15432/voto'
# The client does not need to store data in any database
# so let us define a sqlite in order to avoid warning messages
DEBUG=True
SECRET_KEY = 'django-insecure-alczftnjl#sv%xmkk@5j(n*px43c8kxgi_ua4%khc+t7g_)s9d'

```

Y el utilizado en el cliente en la VM3 es (la ip es la de la interfaz wifi del host, el puerto de la base de datos es el de la base de datos de la VM1 desde el host, y el puerto de la API REST es el de Unicorn de la VM2 desde el host) (se ha puesto como base de datos la que usa todo el sistema, que es la de la VM1; ya que sino los tests fallaban):

```

## 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@192.168.1.130:15432/voto'
# The client does not need to store data in any database
# so let us define a sqlite in order to avoid warning messages
DEBUG=True
SECRET_KEY = 'django-insecure-alczftnjl#sv%xmkk@5j(n*px43c8kxgi_ua4%khc+t7g_)s9d'
RESTAPIBASEURL = 'http://192.168.1.130:28000/restapiserver/'

```