Documentación

El proyecto consta de una API REST usando el lenguaje de programación Python, el framework Django, base de datos relacional Postgresql, base de datos no relacional Redis (para el websocket) y Docker para desplegar la infraestructura.

Para desplegar el entorno, ejecutar los siguiente comandos:

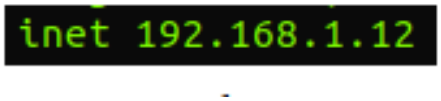
1) Para crear la Base de Datos Relacional Postgresql:

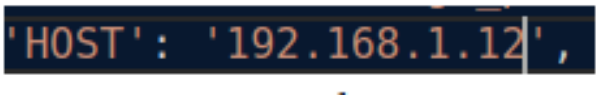
docker run --name db_charge -p 5432:5432 -e POSTGRES_PASSWORD=charge_p4ss -e POSTGRES_USER=charge_us3r -e POSTGRES_DB=charge_db -d postgres

2) Para crear la Base de Datos No Relacional Redis:

docker run --name=redis-server -p 6379:6379 -d redis

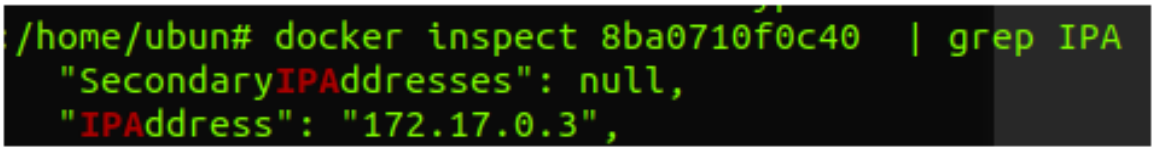
3) Ver la ip del equipo, copiarla y pegarla en la configuración de la base de datos en el “settings.py”:





4) Conectar Redis a Django, para esto, ejecutamos el siguiente comando:

docker inspect <container_id> | grep IPA



y copiamos la IP del Docker de Redis a la configuración en el “settings.py”

5) Para crear una Imagen Docker a partir del Dockerfile, ir dentro de la carpeta “django“ donde se encuentra la carpeta de "charge" junto con el archivo Dockerfile, ejecutar el siguiente comando:

docker build -t docker-django .

6) Para crear el Contenedor de Django exponiendo el puerto 8000 :

docker run --name django_charge -p 8000:8000 -d docker-django

7) Para migrar los modelos de Django en la Base de Datos:

docker exec <container_id> bash -c "python3 manage.py migrate"

8) Para crear un Usuario Administrador en Django:

docker exec -it <container_id> python3 manage.py createsuperuser

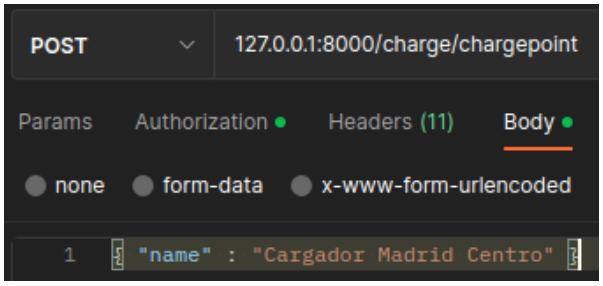
Una vez ejecutado todos los comandos, tendremos lista la API REST en el puerto 8000.

Luego hay que iniciar sesión en Django Admin y en el apartado de status 127.0.0.1:8000/charge/status tenemos una conexión Websocket, que nos muestra de cualquier cambio en el estado de los cargadores indicando el id y su nombre.

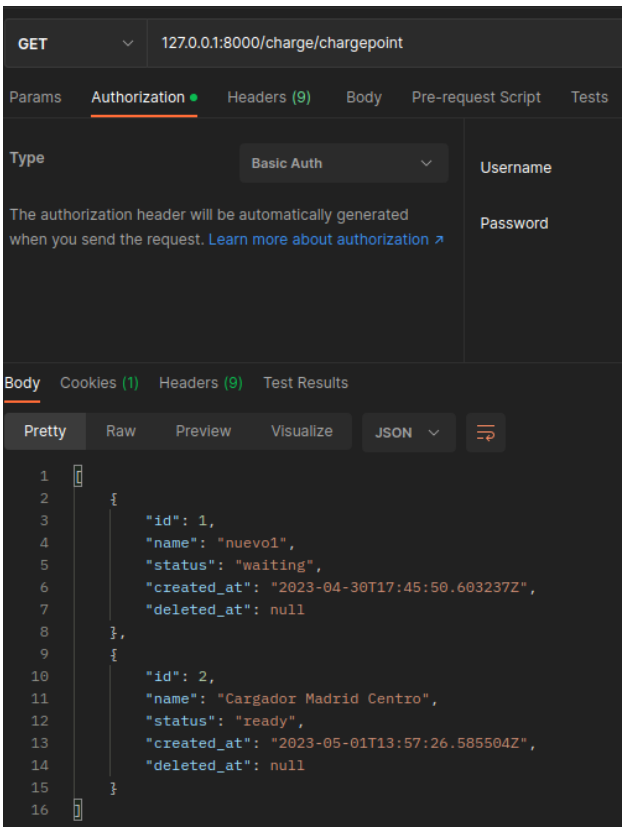
Para todos los Endpoints necesitamos estar autenticados (Basic Auth) con usuario y contraseña. (por ejemplo el usuario administrador).

Métodos de la API:

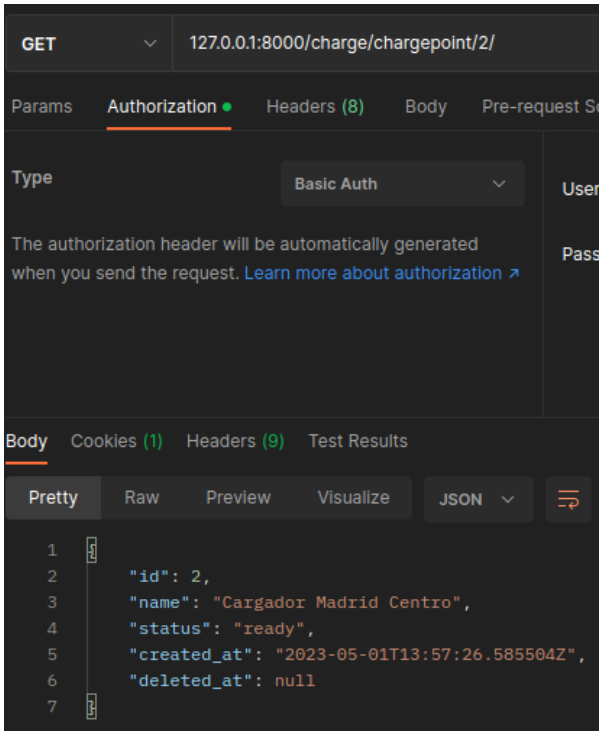
POST: Al enviar una petición POST por la api, con un json indicando solamente su campo “name”, se rellenan todos los campos necesarios para crear el modelo ChargePoint.



GET: al hacer una petición, se devuelven todos los puntos de carga guardados en la base de datos, menos los que tienen “deleted_at” porque se suponen que están eliminados esos.

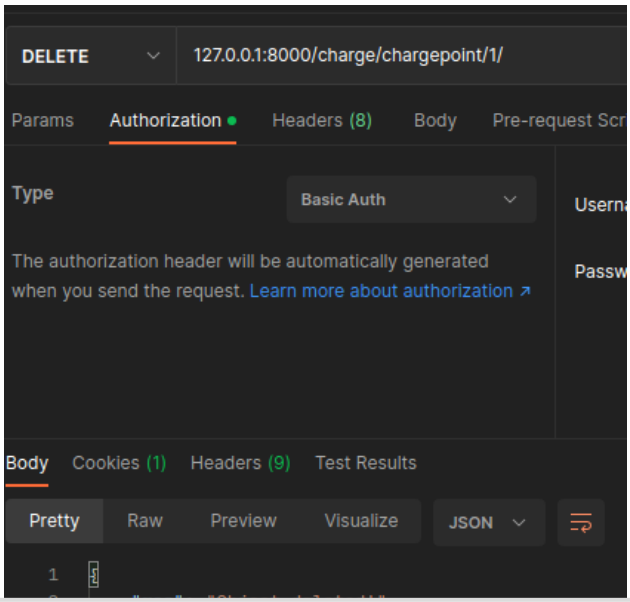


GET by ID: al hacer una petición pasando como argumento el ID, se devuelven todos los datos que pertenecen a dicho ID.

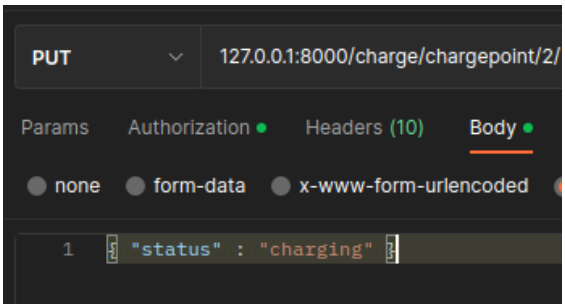


DELETE: by ID al enviar una petición con el

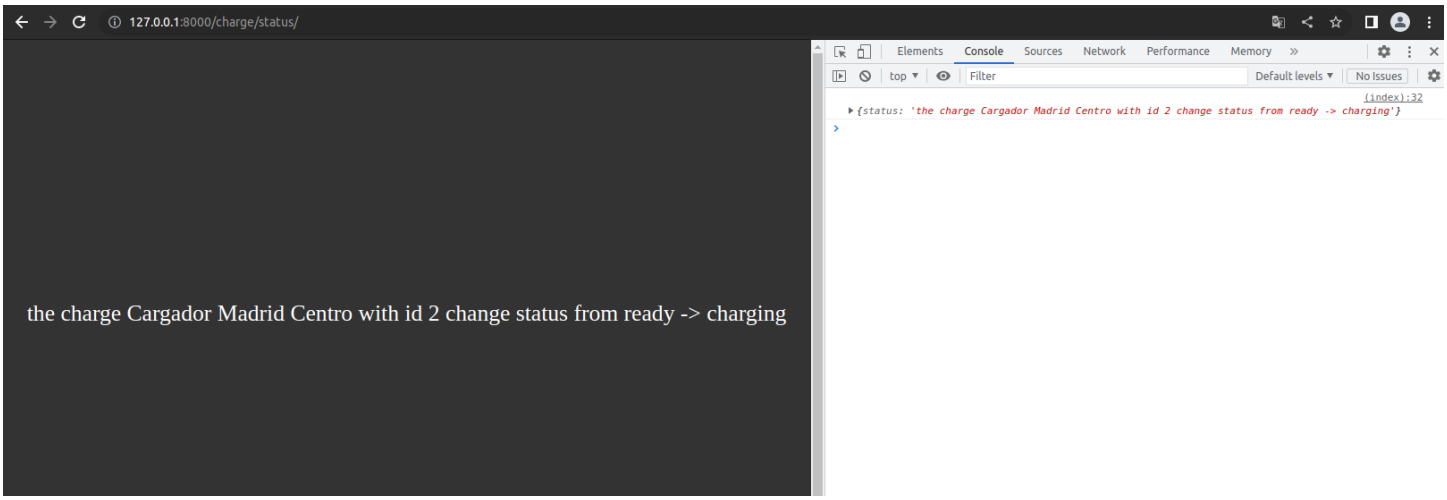
método DELETE, lo que se hace es asignarle al campo “delete_at” el timenow en ese momento.



PUT by ID: el último endpoint que tiene la api, es el método PUT, que sirve para actualizar los estados de los puntos de carga.



Y el **Websocket** nos informa en tiempo real, el cambio de estado y en que cargador ocurrió:



Por último, podemos ver que el programa pasa todos los **Tests**: