



Evaluación práctica (Django + DRF)

Objetivo

Demostrar comprensión de:

- **RESTful API** (recursos, verbos, códigos, stateless, versionado).
- **Arquitectura REST** por capas en un proyecto Django.
- Implementación mínima con **DRF**.

Entregables (3 archivos obligatorios)

1. **README.md** (1 página)
 - Recursos y URIs: `/api/v1/tasks/`, `/api/v1/tasks/{id}/`.
 - Verbos y códigos: qué hace cada endpoint y códigos esperados.
2. **Código Django** (carpeta del proyecto) con:
 - Modelo, serializer, viewset/urls funcionando.
 - Config DRF básica
3. **tests.md** con **5 comandos curl** que muestren: listar, crear, detalle, actualizar `done`, `404`, o en su defecto capturas de pantalla de Postman con cada endpoint funcionando

Nota: Se evalúa que corra localmente y responda JSON.

Requisitos funcionales mínimos

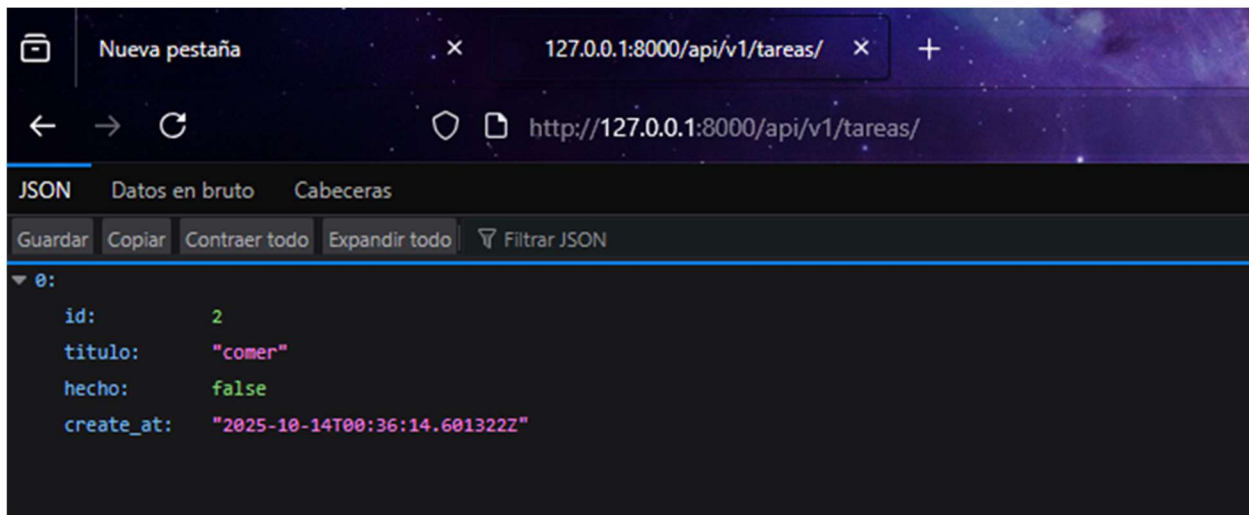
Recurso principal

Tarea con campos:

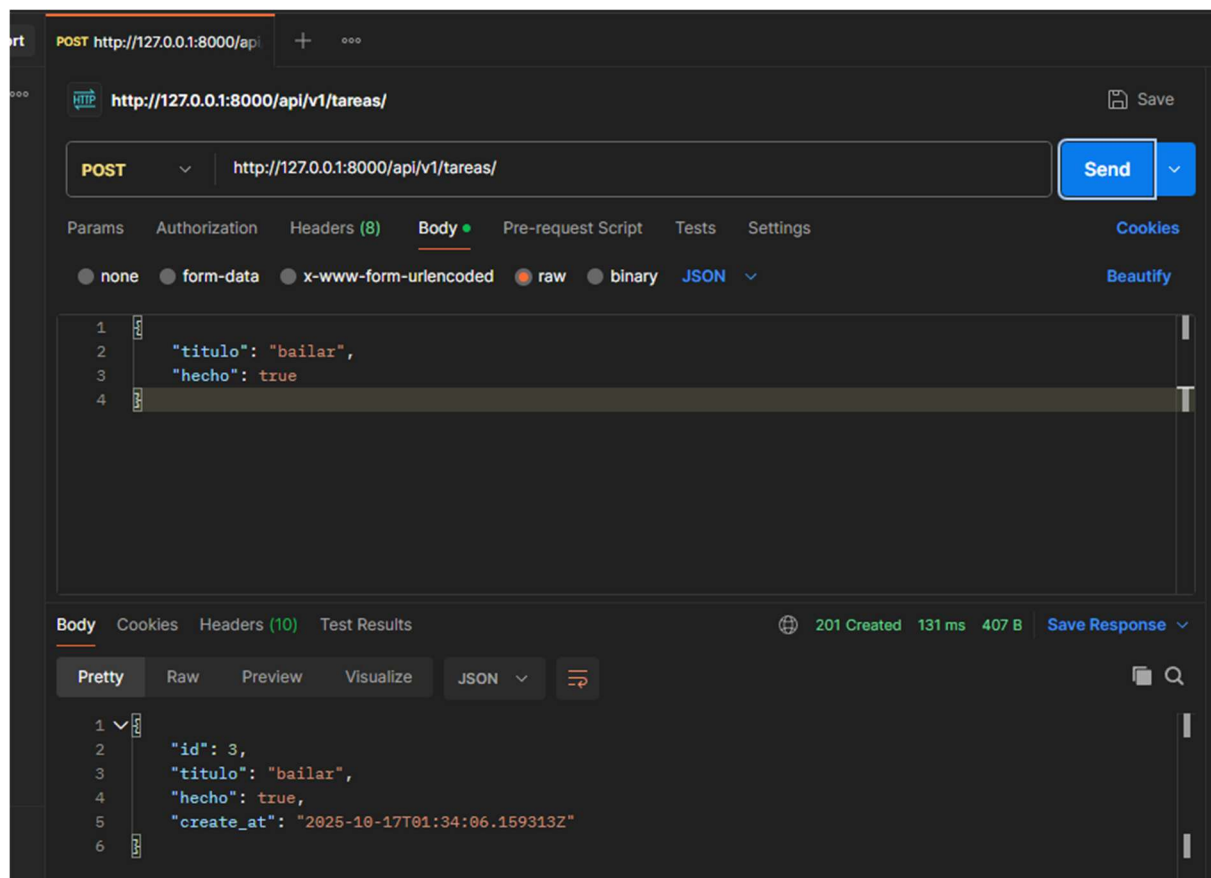
- `id` (auto)
- `titulo` (string, requerido)
- `hecho` (boolean, por defecto `false`)
- `created_at` (auto)

Endpoints (DRF)

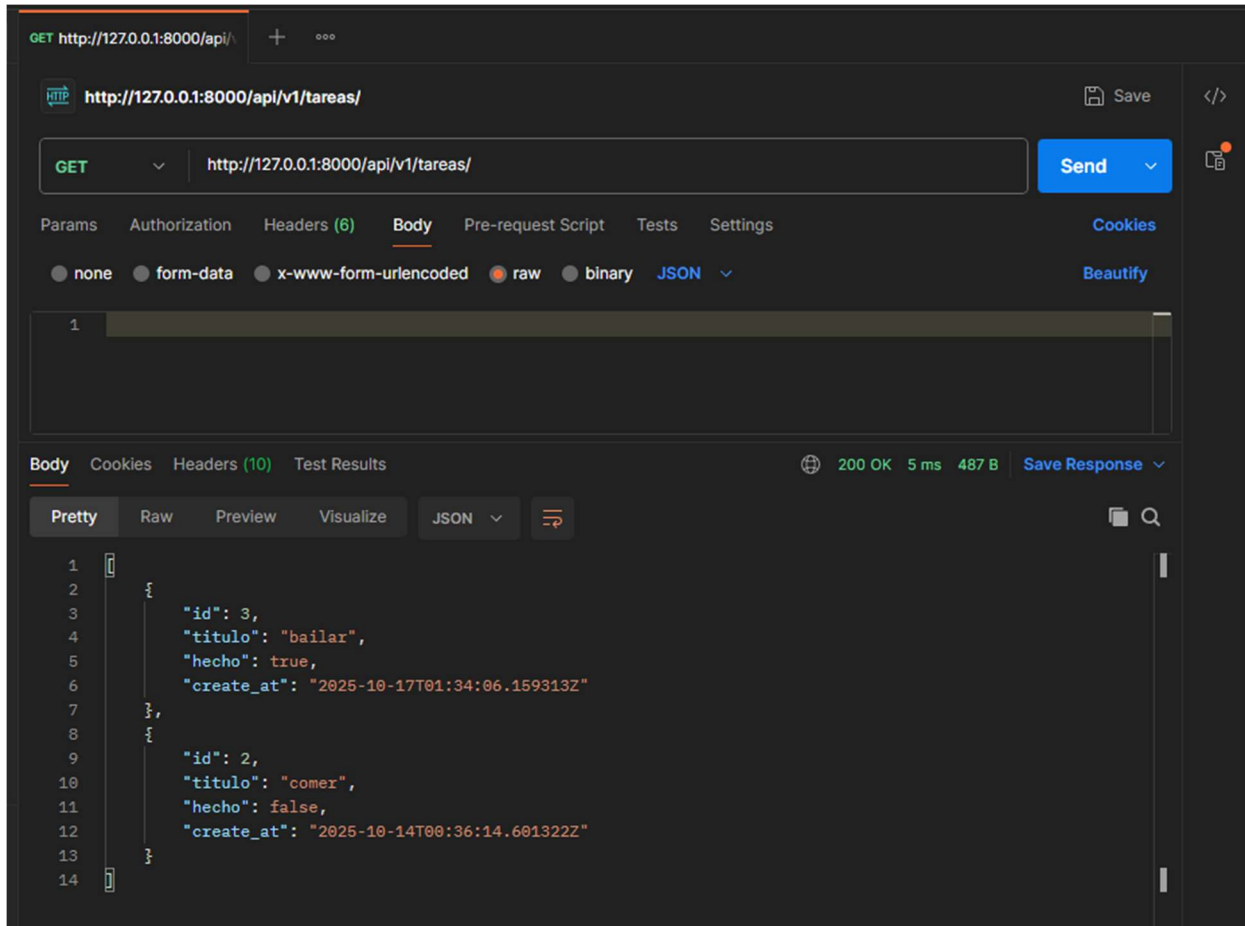
- `GET /api/v1/tareas/` → lista paginada.



- POST `/api/v1/tareas/` → crea.



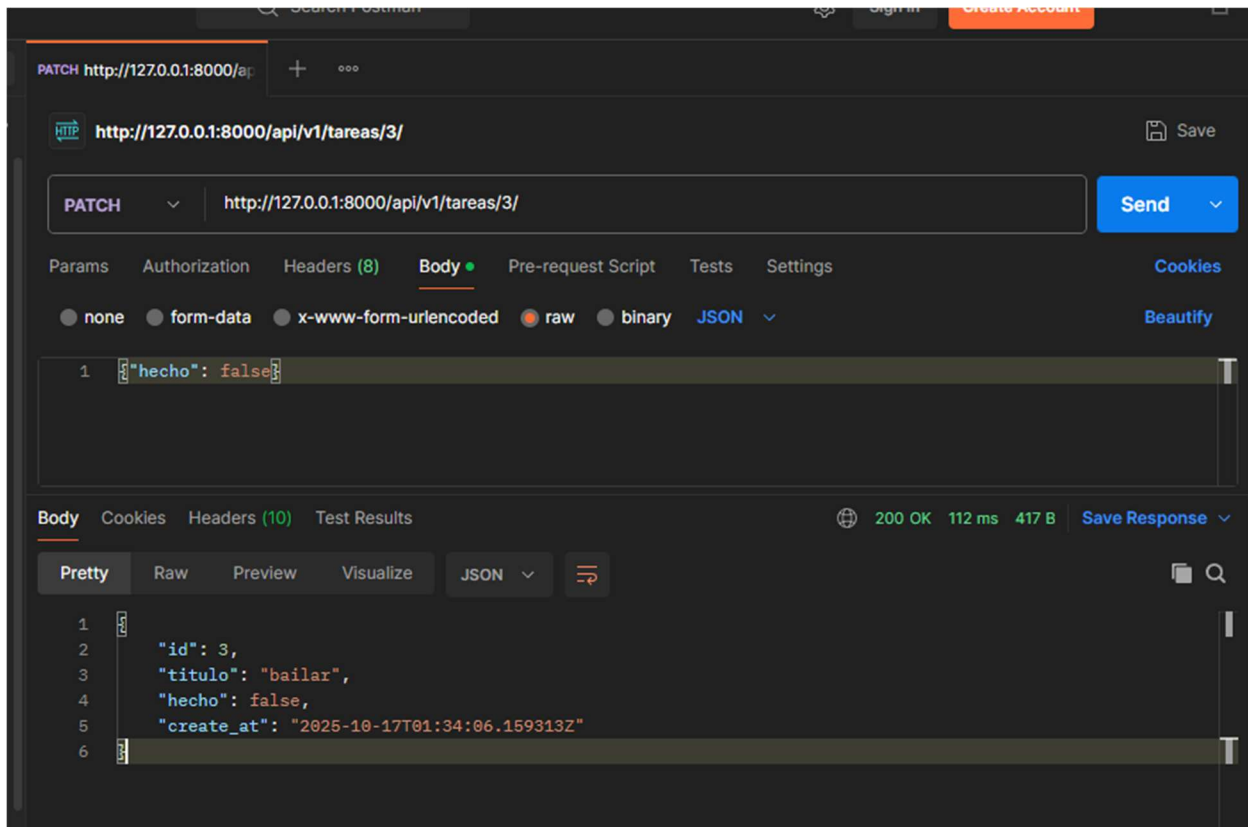
- GET /api/v1/tareas/{id}/ → detalle.



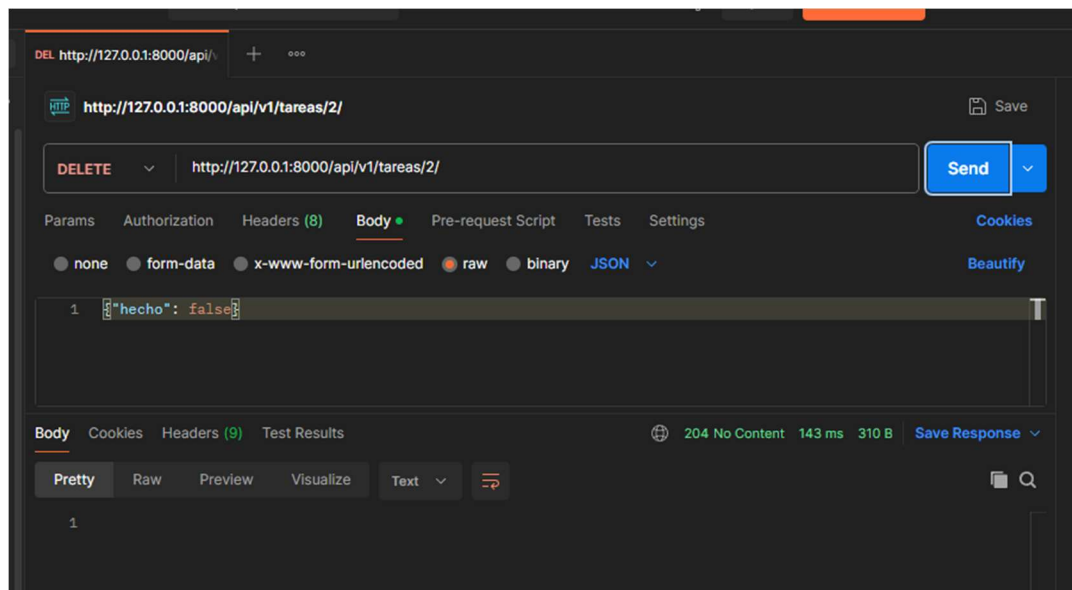
The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:8000/api/v1/tareas/`. The response is a JSON array of two task objects, displayed in the 'Body' tab with the 'Pretty' format. The status is 200 OK, with a response time of 5 ms and a size of 487 B.

```
1 {  
2   {  
3     "id": 3,  
4     "titulo": "bailar",  
5     "hecho": true,  
6     "create_at": "2025-10-17T01:34:06.159313Z"  
7   },  
8   {  
9     "id": 2,  
10    "titulo": "comer",  
11    "hecho": false,  
12    "create_at": "2025-10-14T00:36:14.601322Z"  
13  }  
14 }
```

- PATCH /api/v1/tareas/{id}/ → actualiza title o done.



- DELETE /api/v1/tareas/{id}/ → 204.



GET http://127.0.0.1:8000/api/ + ...

HTTP http://127.0.0.1:8000/api/v1/tareas/ Save

GET http://127.0.0.1:8000/api/v1/tareas/ Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary JSON Beautify

1

Body Cookies Headers (10) Test Results 200 OK 5 ms 405 B Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 3,
4     "titulo": "bailar",
5     "hecho": false,
6     "create_at": "2025-10-17T01:34:06.159313Z"
7   }
8 ]
```



Códigos HTTP esperados

- 200 OK (GET/PATCH), 201 Created (POST), 204 No Content
- 400 Bad Request (validación), 404 Not Found (id inexistente).

Reglas REST que debes mencionar y explicar con tus palabras en README

- **Stateless:** *se refiere a que no hay sesiones guardadas y que cada request tiene que llevar su identificación completa*
- **JSON** (Content-Type: application/json): *vendría siendo como un formato que entienden todos los lenguajes y dispositivos*
- **Versionado** en la ruta: /api/v1/. . . : *se coloca la versión en la ruta para no romper cosas, talvez la nueva versión viene con mas cosas pero las cosas actuales aun siguen usando la v1*
- **Idempotencia:** GET no cambia estado; PATCH repetido deja mismo estado: *algunas operaciones puedes repetirlas sin cambiar el resultado*

Diagrama de arquitectura (inclúyelo en README)

```
[Cliente (curl/SPA)]
    |
    HTTP
    /JSO
    N|
[ API /api/v1 (DRF ViewSets/URLs) ]
    |
[ Lógica/Serializers (validación) ]
    |
[ Modelo Django (ORM) ]
    |
[ DB SQLite (local) ]
```

Una línea por capa describiendo su función.

Cliente: Es el que pide las cosas

HTTP/JSON: http podría ser el camino y json seria el idioma universal

API /api/v1 (DRF ViewSets/URLs): es el que recibe el pedido y identifica lo que se esta pidiendo

Lógica/Serializers(Validación): prepara la información y valida que lo que se pide sea correcto

Modelo Django (ORM): busca lo que se este pidiendo dentro de la base de datos y lo presenta si es correcto



DB SQLite (local): vendría siendo donde se guarda todo ordenado para cuando se realicen consultas