


Open   Opened 2 days ago by  **Rubén Montero**

# Un endpoint para añadir datos

## Resumen

- Modificaremos el *endpoint* `/entries` para que permita **añadir** datos a la base de datos si recibe un **POST**

## Descripción

Hemos visto cómo a través del *mapeo objeto-relacional* podemos recuperar datos de una base de datos y exponerlos en una fachada REST.

Nuestro *endpoint* para `/entries` quedó así tras la tarea anterior:

```
def all_entries(request):
    if request.method != "GET":
        return JsonResponse({"error": "HTTP method not supported"}, status=405)
    all_rows = Entry.objects.all()
    json_response = []
    for row in all_rows:
        json_response.append(row.to_json())
    return JsonResponse(json_response, safe=False)
```

## Soportando la operación Create

Ahora permitiremos que los clientes HTTP puedan **crear** *Entradas*.

Comenzaremos:

- Añadiendo `@csrf_exempt`
- Englobando en un `if` el código anterior, asociado a `"GET"`
- Añadiendo `if request.method == "POST"`
- Devolviendo un `405` en el `else:` final

```
from django.views.decorators.csrf import csrf_exempt
from .models import Entry

@csrf_exempt
def all_entries(request):
    if request.method == "GET":
        all_rows = Entry.objects.all()
        json_response = []
        for row in all_rows:
            json_response.append(row.to_json())
        return JsonResponse(json_response, safe=False)
    elif request.method == "POST":
        # Aquí procesamos un POST
        # ...
    else:
        return JsonResponse({"error": "HTTP method not supported"}, status=405)
```

## Un momento... ¿con qué datos creamos la *Entrada*?

Buena pregunta.

El cliente HTTP enviará en el **cuerpo de la petición** los datos necesarios (**título** y **contenido**). Los esperaremos en formato JSON con claves como en el siguiente ejemplo:

```
{
  "new_title": "Una entrada nueva para el muro",
  "new_content": "¡Aunque no muy relevante!"
}
```

## cURL

Es decir, **al terminar nuestro endpoint**, el siguiente POST enviado desde **cURL** debería generar una nueva **Entry** en la base de datos:

```
curl http://localhost:8000/entries -d '{"new_title": "Una entrada nueva para el muro", "new_content": "¡Aunque no muy
```

## Vamos allá

### 1) Obtener los datos de la petición HTTP

Vamos a asignar a dos variables el contenido esperable del JSON del cliente HTTP:

```
elif request.method == "POST":
    # Aquí procesamos un POST
    client_json = json.loads(request.body)
    entry_title = client_json.get("new_title", None)
    entry_content = client_json.get("new_content", None)
    # ...
```

(Recuerda `import json`)

Podemos aprovechar a devolver un **400 Bad Request** si el cuerpo es incorrecto:

```
elif request.method == "POST":
    # Aquí procesamos un POST
    client_json = json.loads(request.body)
    entry_title = client_json.get("new_title", None)
    entry_content = client_json.get("new_content", None)
    if entry_title is None or entry_content is None:
        return JsonResponse({"error": "Missing new_title or new_content"}, status=400)
    # ...
```

### 2) Crear una nueva instancia del modelo

Lo mismo que hacíamos en Java usando `new`.

Pero esta vez, al estilo Python y pasando los valores de los atributos que hemos sacado de la petición HTTP:

```
elif request.method == "POST":
    # Aquí procesamos un POST
    client_json = json.loads(request.body)
    entry_title = client_json.get("new_title", None)
    entry_content = client_json.get("new_content", None)
    if entry_title is None or entry_content is None:
        return JsonResponse({"error": "Missing new_title or new_content"}, status=400)
    # Creamos una nueva instancia de Entry
    new_entry = Entry(title=entry_title, content=entry_content)
```

### 3) Invocamos `save()`

Esta simple línea se puede invocar sobre *cualquier* instancia de `models.Model`, y sirve para **guardarla** en la base de datos.

```
elif request.method == "POST":
    # Aquí procesamos un POST
    client_json = json.loads(request.body)
    entry_title = client_json.get("new_title", None)
    entry_content = client_json.get("new_content", None)
    if entry_title is None or entry_content is None:
        return JsonResponse({"error": "Missing new_title or new_content"}, status=400)
    # Creamos una nueva instancia de Entry
    new_entry = Entry(title=entry_title, content=entry_content)
    new_entry.save()
```

## Código HTTP 201

**201 Created** es el código de respuesta apropiado cuando hemos añadido un nuevo **dato** en el servidor:

```
elif request.method == "POST":
    # Aquí procesamos un POST
    client_json = json.loads(request.body)
    entry_title = client_json.get("new_title", None)
    entry_content = client_json.get("new_content", None)
    if entry_title is None or entry_content is None:
        return JsonResponse({"error": "Missing new_title or new_content"}, status=400)
    # Creamos una nueva instancia de Entry
    new_entry = Entry(title=entry_title, content=entry_content)
    new_entry.save()
    return JsonResponse({"it_was_successful": True}, status=201)
```

## ¡Terminado!

Tu primer *endpoint* POST que sirve para guardar un nuevo **dato** ya está listo. Si lanzas el servidor y el `curl` de arriba, crearás una nueva fila en la tabla.

Además, los datos que añadas usando POST puedes consultarlos con un GET.

## Endpoints reales

Estos *endpoints* son perfectamente válidos para poner a andar en un servidor de producción para una aplicación.

Permitirían a **usuarios de todo el mundo acceder** y **actualizar** concurrentemente la misma base de datos. ¡Así es como funcionan las aplicaciones distribuidas!

## Por último

Comprueba que tu código pasa el *test* asociado a la tarea.

Haz `commit` y `push` para subir los cambios al repositorio.



Rubén Montero @ruben.montero changed milestone to [%Sprint 4](#) 2 days ago