


Open · Opened 2 days ago by  [Rubén Montero](#)

Un primer endpoint real

Resumen

- Usaremos el método `.objects.all()` para hacer un `SELECT` de todas las filas de una tabla
- Crearemos un primer *endpoint* que devuelve **todas** las `Entry`

Descripción

Ahora que hemos puesto en marcha todo el tinglado del **mapeo objeto-relacional** en nuestro proyecto Django, podemos sacar provecho.

Creemos un archivo `endpoints.py` dentro de `wallrest04app` y añadamos una primera función:

```
from django.http import JsonResponse

def all_entries(request):
    if request.method != "GET":
        return JsonResponse({"error": "HTTP method not supported"}, status=405)
    # ...
```

Ahora, desde aquí, **accederemos** a todas las filas de la tabla `Entry` de la base de datos.

Marchando SQL...

¡No tan rápido!

¿No?

No.

Basta con **importar** la clase `Entry` :

```
from django.http import JsonResponse
from .models import Entry # Es necesario el punto (.)
```

...e **invocar** `.objects.all()` sobre una instancia la **clase**:

```
def all_entries(request):
    if request.method != "GET":
        return JsonResponse({"error": "HTTP method not supported"}, status=405)
    all_rows = Entry.objects.all()
```

Como ves, devuelve un resultado que estamos almacenando en `all_rows` .

Recorriendo las filas

Este resultado es de un tipo especial `QuerySet []` que podemos **iterar** con un bucle `for` de Python:

```
def all_entries(request):
    if request.method != "GET":
        return JsonResponse({"error": "HTTP method not supported"}, status=405)
    json_response = []
    all_rows = Entry.objects.all()
    for row in all_rows:
        # Iteramos sobre cada fila SQL de la tabla Entry
    # ...
```

Accediendo a los campos de cada fila

Aquí viene uno de los pasos **clave** entendiendo el *mapeo objeto-relacional* (✨)

- Acceder a un **atributo** de una *instancia* de `Entry` es como acceder a un **campo** de la fila SQL asociada

En otras palabras

Cada `row` (dentro del bucle `for`) es una de las filas de la tabla SQL y **también** una *instancia* de `Entry`.

Podemos acceder a `row.id`, `row.title`, `row.content` y `row.publication_date` para inspeccionar el contenido.

¡Hagámoslo!

Vamos a guardar el título de cada `Entry` en una lista `json_response` y devolverla finalmente:

```
def all_entries(request):
    if request.method != "GET":
        return JsonResponse({"error": "HTTP method not supported"}, status=405)
    all_rows = Entry.objects.all()
    json_response = [] # Esta variable sirve como acumulador
    for row in all_rows:
        # Iteramos sobre cada fila SQL de la tabla Entry
        json_response.append(row.title)
    return JsonResponse(json_response, safe=False)
```

¡Función lista! Ya sólo falta conectarla en...

urls.py

```
from wallrest04app import endpoints

urlpatterns = [
    path('admin/', admin.site.urls),
    path('entries', endpoints.all_entries),
]
```

¡Endpoint listo!

Lanza el servidor y visita:

- <http://localhost:8000/entries>

¿Tiene sentido el resultado?

Una pequeña mejora

De momento, sólo se devuelve el `title` de cada fila.

Podemos mejorarlo devolviendo **todos** los datos.

Una buena práctica es hacer esto **creando un nuevo método** en nuestra clase `Entry` para que *serialice* sus datos.

¿Se puede hacer?

Sí.

¿Cómo?

```
class Entry(models.Model):
    title = # ...
    content = # ...
    publication_date = # ...

    def to_json(self):
        return {
            "title": self.title,
            "content": self.content,
            "created": self.publication_date,
        }
```

(¿Observas que las **claves** del JSON las decidimos libremente? `created != publication_date`)

Gracias a esto...

Basta reemplazar **una** línea para que nuestro API REST genere una respuesta mucho más completa:

```
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.title)
+         json_response.append(row.to_json())
    return JsonResponse(json_response, safe=False)
```

Probemos de nuevo qué responde nuestro API:

- <http://localhost:8000/entries>

¡Enhorabuena! Has creado tu primer API REST que accede a datos mediante *mapeo objeto-relacional*.

Este es un paso **muy importante**, y estás muy cerca de poder crear fachadas REST dignas de estar en producción.

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