


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

Las tablas del $-\infty$ al $+\infty$

Resumen

- Recordaremos qué son los *query params*
- Veremos cómo se interpretan en Django los *query params*
- Añadiremos una nueva función a `endpoints.py` que se *mapeará* a una URL no-parametrizada. Será la nueva función quien compruebe ella sola si ha recibido *query params*
 - Devolverá en JSON una tabla de multiplicar

Descripción

Hemos visto cómo *mapear* y usar *path params* en Django. Pero recordemos que en las URL hay otro tipo de parámetros de mucha relevancia: Los ***query params***.

¿Qué son?

Parámetros de la URL que viajan en formato `clave=valor` detrás del carácter *interrogante* (`?`). Si hay varios, se anexan con *ampersand* (`&`). Por ejemplo:

- <http://localhost:8000/dashboards?size=10&search=new>

(Viajan [URL-encoded](#) en las peticiones HTTP)

¿Y por qué hay dos tipos distintos de parámetros?

- Los ***path params*** son **parte esencial** de la URL
- Los ***query params*** son **parámetros adicionales** que permiten especificar valores añadidos a la petición, por motivos de paginación, búsqueda, *tracking*,...

Una clave para entenderlo es pensar en una búsqueda de Google (u otro buscador)

- Sería razonable que estas dos páginas aparezcan como distintos *resultados de búsqueda*:
 - <https://twitter.com/elonmusk>
 - <https://twitter.com/neiltyson>
- Pero **no** sería lógico que estas dos páginas apareciesen como distintos *resultados de búsqueda*:
 - https://twitter.com/neiltyson?last_tweets=5
 - https://twitter.com/neiltyson?last_tweets=10

Lo entiendo

Excelente.

Venga, un pequeño ejemplo

De las siguientes URLs, ¿qué partes crees que son *path params*?

¿Qué partes crees que son *query params*? Y de estos, ¿cuál es la *clave* y el *valor* de cada uno?

- <https://www.listaspam.com/busca.php?Telefono=5555555>
- https://www.elmundo.es/television/momentvs/2022/07/12/62ccffc3fdddf92098b45c9.html?utm_source=marca
- https://es.wikipedia.org/wiki/Protestas_en_Sri_Lanka_de_2022
- <https://duckduckgo.com/?t=ffab&q=programación&ia=web>

Y en Django... ¿cómo se *leen* los *query params* que hemos recibido?

Dado que los *query params* se consideran muy variables (digamos, volátiles), **no** se indican en `urls.py`.

Es decir, tanto la URL:

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

...como:

- http://localhost:8000/ejemplo?una_cosa=cualquiera

...serían **procesadas** por el mismo **endpoint**:

```
urlpatterns = [
    # ...
    path('ejemplo', endpoints.mi_funcion),
]
```

Vale, entonces...

En Django, ¿cómo se *leen* los *query params*?

Dentro de la función, accedemos a ellos a través del diccionario de **GET** en el objeto **request**.

Es decir:

```
def mi_funcion(request):
    # Siguiendo el ejemplo de arriba,
    # Esto imprimirá 'cualquiera'
    print(request.GET["una_cosa"])
```

Acceder de forma segura

Si el *query param* **no** viene en la petición, entonces el código de arriba **lanzará una excepción** **KeyError**.

Podemos controlarla con **try-except**.

Alternativamente, podemos usar el método **.get()** para **acceder** al contenido del diccionario.

1. Se indica la clave *a acceder* y un *valor por defecto*: **.get("clave", "Valor por defecto")**
2. En caso de que la clave *no exista* en el diccionario, entonces toma el *valor por defecto*.

Por ejemplo:

```
def mi_funcion(request):
    # Para: http://localhost:8000/ejemplo?una_cosa=cualquiera
    # ...imprimirá: 'cualquiera'
    print(request.GET["una_cosa", "ninguna"])
    # Pero para: http://localhost:8000/ejemplo
    # ...imprimirá 'ninguna'
```

Hagamos algo

Vamos a **añadir** en nuestro **urls.py** un nuevo *endpoint*:

```
urlpatterns = [
    # ...
    path('v3/multiplication', endpoints.multiplication_table_query_param),
]
```

Y en **endpoints.py** ...

...añadiremos la función asociada.

Asignará a una **variable** el valor del *query param* **d**, con valor por defecto **None**:

```
def multiplication_table_query_param(request):
    number = request.GET.get("d", None)
```

O sea que podemos esperar responder a URLs como:

- <http://localhost:8000/v3/multiplication?d=5>
- <http://localhost:8000/v3/multiplication?d=14>

Siguiente paso

Si el valor **no** es **None**, fabricaremos y devolveremos una **JsonResponse** adecuada:

```
def multiplication_table_query_param(request):
    number = request.GET.get("d", None)
    if number is not None:
        result = []
        for i in range(1, 11):
            result.append(number * i)
        return JsonResponse(result, safe=False)
```

¿Y si el *query param* no está?

Si no lo recibimos, entonces la variable **number** tendrá el valor por defecto **None**.

Devolvemos un JSON indicando al cliente el error. El código HTTP adecuado es **400 Bad Request**:

```
def multiplication_table_query_param(request):
    number = request.GET.get("d", None)
    if number is not None:
        result = []
        for i in range(1, 11):
            result.append(number * i)
        return JsonResponse(result, safe=False)
    else:
        return JsonResponse({"error": "Missing 'd' parameter"}, status=400)
```

¡Tarea *casi* completada!

¡Excelente!

Qué curioso eso de **.GET**

Sí.

Más adelante veremos que si el verbo *HTTP* de la petición es diferente (POST, PUT, DELETE), debemos usar **.POST**, **.PUT**, **.DELETE** consecuentemente.

Falta verificar cómo funciona

Lanza el servidor y navega a:

- <http://localhost:8000/v3/multiplication?d=2>

Es posible que veas algo como esto:

```
[
  "2",
  "22",
  "222",
  "2222",
  "22222",
  "222222",
  "2222222",
  "22222222",
  "222222222",
  "2222222222"
]
```

¡Ups!

Parece que el operador *multiplicación* (*****) **no** está funcionando como esperamos.

¿Te resulta familiar este error?

Es un problema conocido. Deriva de que **number** es un **string** para Python, en lugar de un **int**.

Vaya...

Claro. Piensa que podríamos recibir cualquier cosa *no numérica*. Por ejemplo:

- <http://localhost:8000/v3/multiplication?d=texto>

¿Y qué hay que hacer?

Se pide que además de arreglar el problema de arriba, **devuelvas** el siguiente JSON con un código 400:

```
{"error": "Parameter 'd' must be a number"}
```

...si el cliente HTTP envía algo que *no es convertible a número entero* como *query param* **d** .

Por último

Verifica que tu código pasa los tests asociados 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



Ania Blanco @ania.blanco mentioned in commit [312e60dc](#) 6 hours ago