API Endpoints Tables

URL_BASE: https://my_server_name/api/v1/

PAYLOAD: mismo ejemplo del documento
data = {
“table_name”: “movie”,
“fields”: {} ..
…
}

requests.post(‘https://my_server_name/api/v1/tables’, data=data)

Al utilizar **POST** me permite crear una tabla o registro de una estructura de tabla, además de poder disponer de las otras acciones PUT , PATCH y DELETE.

GET: https://my_server_name/api/v1/tables (Lista todas las tablas)
PUT: https://my_server_name/api/v1/tables/1 (actualizar registro con ID=1)
PATCH: https://my_server_name/api/v1/tables/1 (actualizar un campo del registro con ID=1)
DELETE: https://my_server_name/api/v1/tables/1 (borrar registro con ID=1)

Por detrás la vista que resuelva cada petición se encargaría de las acciones necesarias y para una posible solución en las vistas considero estas opciones:

- al crear una tabla se podría llamar a las instrucciones de Django para crear una app y escribir el modelo y luego crear el makemigrations (crea el archivo de migraciones) y luego crear la tabla con el migrate (registra en la tabla de migraciones el cambio), mantendría las dependencias en orden.

- Por otro lado, se podría utilizar directamente instrucciones SQL para hacerlo, pero además se deberían actualizar las tablas de migrate de Django y esto podría tener un costo extra.

API Endpoints Movies

Para crear películas pienso que se debería manejar así:

POST: https://my_server_name/api/v1/movies

Mismo ejemplo del documento:

data = {
title = “Apocalypse Now”,
“director”: {
...
}
}

requests.post('https://my_server_name/api/v1/movies’, data=data)

GET: https://my_server_name/api/v1/movies (Lista todas las peliculas)
PUT: https://my_server_name/api/v1/movies/1 (actualizar registro con ID=1)
PATCH: https://my_server_name/api/v1/movies/1 (actualizar un campo del registro con ID=1)
DELETE: https://my_server_name/api/v1/movies/1 (borrar registro con ID=1)

Modelos

Modelo de **Tabla**:

Las tablas podrían tener esta estructura:
id (campo que identifica como único al registro)
Nombre (guardará el nombre de la tabla)
Field (guardará el nombre del campo)
Attribute (podría guardar un diccionario con info de tipo de campo o si es un FK, unique, etc.)

Modelo de **Movie**:

id (campo que identifica como único al registro) podría ser un uuid.
title (guardará el titulo de la pelicula) un charfield
director (en el ejemplo hace referencia a que es FK del modelo Director)
release_date (guardará fecha de lanzamiento de la pelicula) DateField
IMDB_tanking (guardará una puntuación) podría ser un campo Integer.

Modelo **Director** (agrego sólo los 2 datos básicos para el ejemplo)

id (campo que identifica como único al registro) podría ser un uuid.
name(guardará el nombre del director) un charfield