T3: OpenAPI

jueves, 15 de mayo de 2025 17:49

OpenAPI (previamente *Swagger*) es un estándar para la descripción de **APIs RESTful**.

Un documento OpenAPI, es un normalmente un archivo "*.yaml*" o "*.json*" que describe los aspectos de una API, su estructura incluye varios objetos:

```
openapi : 3.1.0                 --> una string obligatoria, que indica la versión del estándar OAS
info:                           --> un objeto que contiene la info. Básica de la API
    title: My first API         --> nombre de la API
    version: 0.0.1
    description: OpenAPI example  --> es opcional
paths : {                       --> describe los endpoints/rutas disponibles en la API
    /books:                     --> cada entrada de un Path Object empieza por su "/"
        get:                    --> dentro de cada Path Object hay Path Items Object, que representan a cualquier método HTTP (Se dice que se definen con Operation Objects)
            summary: Get the book list
            description: Retrieves the book catalog
            responses:              --> Códigos de estado de posibles respuestas para la petición Response Object
                "200":
                    description: "OK"  --> Obligatoria descripción de la respuesta
                    content:           --> Media Map , se utiliza para especificar el contenido que devuelve la API, o para indicar el contenido que espera recibir (en el resquetBody).
                        application/json:    -->  Media Type Objects Las claves de estos objetos son MIME
                            schema:
                                $ref: '#components/schemas/books'
                            example:
                                id: 123
                                name: The picture of Dorian Gray
                "4XX":              --> También se pueden usar placeholders XX para abarcar los códigos de un estilo
                    ...
                "default":

    /book/{id}:         --> un Path Object también puede llevar un placeholder "{}" que parametrice de forma específica la ruta/el objeto
        get:
            parameters:             --> Un Parameter Object puede usarse para las querys/paths/headers
              - name: id            --> nombre del parámetro
                in: path            --> para identificar si es en el path/query/...
                required: true      --> siempre true si está in: path
                schema:             --> formato del dato esperado
                    type: number
            ...
    /book/:
        post:
            summary: Adds a book
            requestBody:
                required: true
                content:
                    application/json:
                        schema:
                            $ref: '#components/schemas/book'
            responses:
                "201":
                    ...

}
servers:                    --> URLs base donde está alojada la API
webhooks:                   --> endpoints externos que la API puede llamar
security:                   --> mecanismos de seguridad, como autentificación por tokens...
components:                 --> sección para definir elementos reutilizables como schemas, parámetros...
```

```
default:
  description: Unexpected error
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/Error'
```

```
examples:
  objectExample:
    $ref: '#/components/examples/objectExample'
```

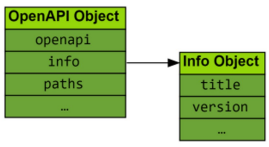
Tener cuidado con:

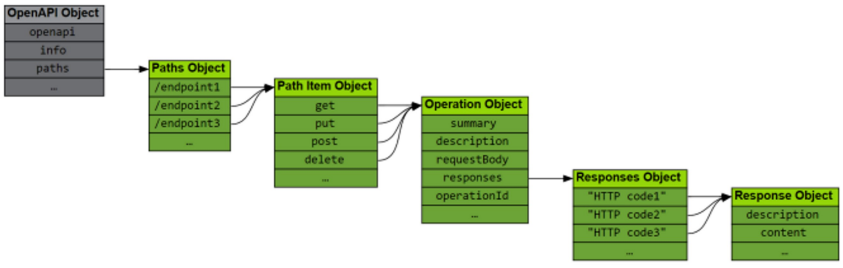
- Rutas ambiguas:
  - /books/{id}
  - /books/{ref}
  - /{something}/x

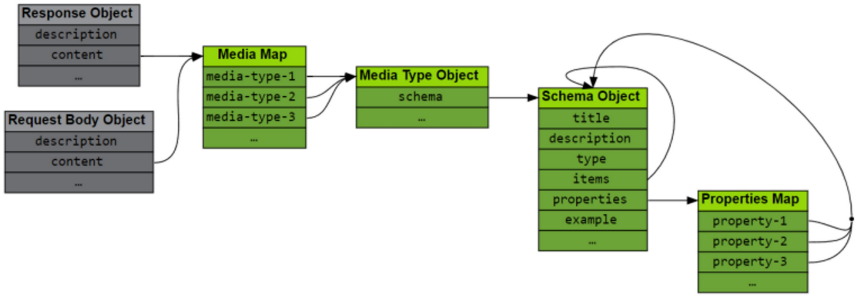
style:
explode:

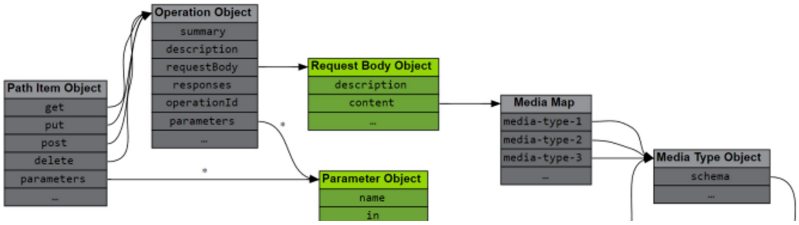
Son Parameters Object que se usan en API complejas, e indican cómo se va a serializar el objeto en el caso de que sean arrays por ejemplo:

```
parameters:
  - name: tags
    in: query
    description: Lista de etiquetas
    required: false
    style: form
    explode: true
    schema:
      type: array
      items:
        type: string
```









Aparte de ser un formato estandarizado, OpenAPI, proporciona **ventajas** claras como:

- Descripción, Validación, Guía y Generador de documentación y código
- Permite levantar Mock Servers
- Permite realizar análisis de seguridad basados en la estructura del servicio

Algunas **alternativas** a OpenAPI:

- GraphQL (complejo para servicios sencillos)
- gRPC (sin cliente, consiste en invocar métodos remotos como si fueran locales)
- Webhooks
  - Es un mecanismo de comunicación entre dos sistemas, normalmente se utiliza para que un servidor notifique a otros cuando ocurre un evento.
    Actúa entonces como una especie de API inversa -donde el primer -y único paso- lo da el servidor.
    ¿Cómo funciona?
    Un cliente expone un endpoint (una URL pública) para recibir notificaciones;
    Registra dicha URL en el servidor;
    Y cuando **ocurre un evento** el s**ervidor envía** una petición **HTTP POST al endpoint**.
    Ofrece **ventajas** como:
    - Evita el polling
    - Eficiencia
    - Reactivo
    - Automatiza procesos de comunicación y triggers

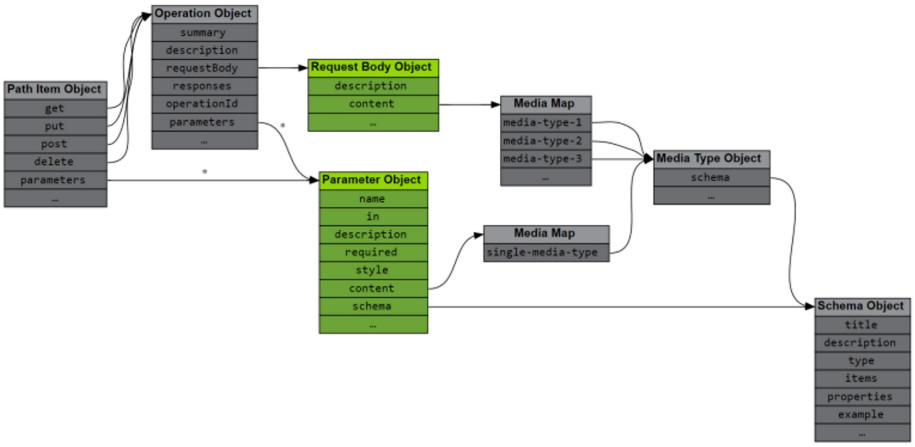
    Sin embargo su mayor **desventaja** si se puede ver así, es que son por consecuencia **unilaterales**.

    Webhook con OpenAPI
    Aunque se pueden añadir al documento de OpenAPI, **no añaden funcionalidad** al sistema, sino, que actúan como **mecanismo de documentación**, de qué notificaciones enviará el sistema a otros.

🔹 Si el **cliente inicia la petición**, va en `paths:`
🔹 Si **tu API inicia la petición al cliente**, va en `webhooks:`

🔹 **Casos típicos reales donde sí deberías definir un webhook:**

Escenario	¿Qué notifica tu API?	Webhook
E-commerce	Cuando se crea un nuevo pedido	`orderCreated`
App de reservas	Cuando se cancela una cita	`reservationCancelled`
CRM o SaaS	Cuando se actualiza el perfil de un cliente	`customerUpdated`
Plataforma de pagos	Cuando un pago se confirma	`paymentConfirmed`
Sistema de tareas	Cuando se asigna una tarea a un usuario	`taskAssigned`



Escalado de una API

"Scaling Throughput"/Rendimiento por segundo:

Y se refiere al número de llamadas que puede manejar la API/seg, y las estrategias que se pueden seguir para detectar posibles cuellos de botella -como la monitorización del uso de los recursos-. Y posibles soluciones como -procesamiento asíncrono, bbdd indexadas, cachés, escalado de recursos...-

Evolución de diseño de APIs

- Nuevos patrones de acceso a datos: Pasar de polling a Webhooks
- Soporte para bulk endpoints (operaciones por lotes)
- Nuevas opciones de filtrado y ordenación

Paginación en APIs

Permite controlar la cantidad de datos que se devuelven en una respuesta -para evitar saturar al servidor-.

Offset-Based Pagination	Es el modelo tradicional que envía número de pag. Y cantidad por página.
Cursor-Based Pagination	El cursor es una posición en la lista de resultados, es fiable cuando los datos cambian constantemente .

```
GET https://api.github.com/user/repos?page=5&per_page=10
```

```
GET /followers/ids.json?user_id=12345&count=50
→ { "ids": [...], "next_cursor": 1374004777531007833 }

GET /followers/ids.json?user_id=12345&count=50&cursor=1374004777531007
```

Rate Limiting en APIs

Consiste en limitar el número de peticiones que un cliente hace a la API en un periodo de tiempo, es especialmente útil, para protegerlo de ataques DoS, Spam, gestionar picos de tráfico...

Las principales estrategias para evitarlo son:

Token Bucket	Trabaja en función de X tokens por Z tiempos
Fixed-Window Counter	Ventana de X peticiones por Z tiempos

Para informar al cliente de estas condiciones normalmente se usan cabeceras HTTP:

Cabecera	¿Qué indica?
`X-RateLimit-Limit`	Límite total permitido
`X-RateLimit-Remaining`	Cuántas peticiones quedan
`X-RateLimit-Reset`	Cuándo se reinicia el contador (timestamp UNIX)

Con código *"429: Too many Requests"* por excesos.