Escuela Politécnica Nacional
Facultad De Ingeniería De Sistemas
Aplicaciones Web
Periodo Académico 2025B

Examen

APIs REST con swagger (OpenAPI)

Aplicaciones web

Christian Aragón

12/12/2025

Contenido

1. Objetivos del Informe

El objetivo principal de este informe es documentar de manera integral una API REST simulada (JSONPlaceholder) utilizando el estándar OpenAPI Specification (OAS 3.0) y las herramientas de Swagger. Se busca demostrar la comprensión de los fundamentos teóricos, la estructura práctica del YAML de OAS y la capacidad de probar los diferentes métodos HTTP (GET, POST, PUT, PATCH, DELETE).

2. Introducción a la Documentación de APIs

La documentación es el pilar para la usabilidad y colaboración en el desarrollo de APIs. Una API, por muy bien diseñada que esté, es inútil si los desarrolladores no pueden entender cómo interactuar con ella. En la arquitectura de microservicios y REST, las APIs actúan como contratos entre el frontend y el backend, y el documento de especificación (el "contrato") es crucial para la eficiencia del equipo.

3. ¿Por qué es fundamental documentar una API?

- **Facilita la Adopción:** Permite que desarrolladores externos (terceros) o nuevos miembros del equipo comiencen a consumir el servicio rápidamente sin necesidad de revisar el código fuente.
- **Acelera el Desarrollo:** Permite el desarrollo simultáneo (paralelo) del frontend y el backend (Design First approach).
- **Asegura la Calidad (Testing):** Herramientas como Swagger UI permiten realizar pruebas de los endpoints en tiempo real, validando que los parámetros y las respuestas coincidan con lo especificado.
- **Reduce Errores:** Una documentación clara minimiza las ambigüedades sobre los tipos de datos, los códigos de estado y los campos obligatorios.

4. Definición y Componentes de Swagger

Swagger es un conjunto de herramientas de código abierto basadas en la especificación **OpenAPI (OAS)**. El término "Swagger" a menudo se utiliza indistintamente con "OpenAPI Specification".

Los componentes clave son:

- **OpenAPI Specification (OAS):** El estándar formal (el archivo YAML o JSON) que describe la estructura de la API.
- **Swagger Editor:** Herramienta para escribir código OAS en YAML o JSON y verlo renderizado en tiempo real.
- **Swagger UI:** Herramienta de visualización que convierte el archivo OAS en una interfaz web interactiva y amigable para el usuario.

5. Conceptos Clave del Estándar OpenAPI (OAS 3.0)

5.1. Estructura Base de OAS

El archivo OAS siempre comienza con los siguientes campos obligatorios para definir los metadatos y el entorno de la API:

- openapi: Versión de la especificación OAS utilizada (ej., 3.0.0).
- info: Contiene metadatos de la API (title, description, version, contact).
- servers: Define el o los servidores base donde reside la API (ej., https://jsonplaceholder.typicode.com).

- paths: Contiene todos los endpoints (/posts, /users, etc.) y sus métodos HTTP asociados.
- components: Contiene objetos reutilizables como esquemas de datos (schemas), respuestas o parámetros comunes.

5.2. Estructura de un Path (Endpoint)

Un path (ruta o endpoint) define una URL específica y los métodos HTTP que acepta.

```yaml
paths:
  /posts:            # La ruta del endpoint
    get:             # El método HTTP
      summary: Obtener todos los posts
      tags: [Posts]
      responses:     # Las respuestas que puede retornar (ej. 200, 404, 500)
        '200':
          description: Lista de posts obtenida exitosamente
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Post' # Referencia al esquema de
```

5.3. Tipos de Parámetros

Los parámetros definen la información adicional que se puede enviar o requerir en una petición:

Tipo	Clave en OAS	Ubicación en la Petición	Uso Común
Path	in: path	Parte de la URL (ej., /users/{id})	Identificadores únicos de recursos.
Query	in: query	Después del signo de interrogación (ej., /comments?postId=1)	Filtrado, paginación, o búsqueda.
Header	in: header	Cabecera HTTP	Autenticación (Authorization), tipos de contenido.
Cookie	in: cookie	Cookie HTTP	Sesiones y autenticación.

Tipo	Clave en OAS	Ubicación en la Petición	Uso Común
Body (Request Body)	requestBody	Cuerpo de la petición	Usado en POST, PUT, y PATCH para enviar datos JSON/XML.

5.4. Diferencia entre PUT y PATCH

Ambos se usan para actualizar recursos, pero se distinguen por su idempotencia y comportamiento:

- **PUT (Actualización Completa):** Reemplaza el recurso completo en la ubicación especificada. Es **idempotente** (repetir la petición no tiene efectos secundarios adicionales). El requestBody debe incluir *todos* los campos del recurso, incluso los que no cambian.
- **PATCH (Actualización Parcial):** Aplica modificaciones parciales a un recurso. Es **no-idempotente** (aunque se intenta que lo sea). El requestBody solo debe incluir los campos que se desean modificar.

6. Comparación de Herramientas: Bruno vs Swagger

Característica	Swagger (OAS + UI)	Bruno (o Postman)
Propósito Principal	Documentación, diseño de contratos y visualización interactiva.	Pruebas, ejecución de peticiones HTTP y organización de colecciones.
Enfoque	"Design First" (Diseñar la API antes de codificarla).	"Code First" (Probar la API después o durante la codificación).
Salida	Archivo estático YAML/JSON (el contrato).	Colección dinámica de peticiones (el flujo de pruebas).
Ventaja Clave	Genera una documentación que actúa como verdad única y permite la generación de código.	Permite scripting avanzado, encadenamiento de peticiones y variables de entorno.

7. Evidencia Práctica y Documentación de Métodos

El siguiente trabajo práctico se basó en el archivo **swagger-documentation.yaml** (Anexo A) y se verificó en el Swagger Editor y Swagger UI. Se documentaron los métodos CRUD (Crear, Leer, Actualizar, Eliminar) utilizando la API de JSONPlaceholder.

7.1. Prueba de Lectura (GET)

Endpoint Documentado	Descripción	Evidencia (OAS YAML)
/posts (Todos)	Obtiene la lista completa de publicaciones.	Define el tipo de respuesta como array de $ref: Post.
/posts/{id} (Específico)	Obtiene un solo post por ID.	Requiere el parámetro id en path, con validación minimum: 1 y maximum: 100.
/comments?postId (Filtrado)	Obtiene todos los comentarios, permitiendo filtrar por el parámetro opcional postId en query.	Incluye el parameter postId con in: query y required: false.

7.2. Prueba de Creación (POST)

- **Endpoint:** /posts
- **Método:** POST
- **Descripción:** Documenta la creación de un nuevo post.
- **Esquema Usado:** Se define el esquema PostCreate, que requiere title, body y userId, pero **no** requiere el id (ya que se espera que el servidor lo asigne).
- **Respuesta Esperada:** El código 201 (Created) que devuelve el recurso creado, incluyendo el nuevo id (ejemplo: id: 101).

7.3. Prueba de Actualización (PUT y PATCH)

- **Endpoint:** /posts/{id}
- **PUT (Completa):**

    - Requiere que el requestBody utilice el esquema completo Post.
    - Documenta que si se omite un campo (ej. title), el valor de ese campo en el recurso original será reemplazado por null o un valor vacío.
- **PATCH (Parcial):**
    - Requiere que el requestBody utilice el esquema parcial PostPatch.
    - Documenta que solo se envían y modifican los campos especificados (ej. solo title).

7.4. Prueba de Eliminación (DELETE)

- **Endpoint:** /posts/{id}
- **Método:** DELETE
- **Descripción:** Elimina un recurso por su ID.
- **Respuesta Esperada:** El código 200 (OK) con un cuerpo vacío ({}) o un mensaje de confirmación, simulando que el recurso ha sido exitosamente eliminado del servidor.

8. Conclusiones

La documentación de APIs con el estándar OpenAPI y las herramientas de Swagger es una práctica indispensable en el desarrollo de software moderno. El ejercicio práctico demostró que:

1. El formato **YAML de OAS 3.0** es robusto y permite definir con precisión cada aspecto de la API (esquemas, parámetros, respuestas).
2. La definición de esquemas reutilizables (components/schemas) es clave para la consistencia y escalabilidad de la documentación.
3. Se logró documentar todas las operaciones CRUD y las sutiles diferencias (como entre PUT y PATCH), facilitando a cualquier consumidor la comprensión del contrato de la API.