Examen

“APIs REST con Swagger (OpenAPI)”

Aplicaciones Web

Albán Lucas Dorian Joel

Escuela Politécnica Nacional

11 de noviembre de 2025

Tabla de contenido

1. Objetivos del Informe

Este informe tiene como finalidad establecer los fundamentos teóricos y conceptuales de la documentación de APIs REST utilizando el estándar Swagger/OpenAPI, cumpliendo con los siguientes puntos:

I. **Establecer la Importancia:** Comprender la relevancia de la documentación de APIs en el ciclo de vida del desarrollo de software.

II. **Definir el Ecosistema:** Identificar los componentes clave de la suite Swagger (Editor, UI, Codegen) y su relación con el estándar OpenAPI Specification (OAS).

III. **Entender el Estándar:** Desglosar los conceptos fundamentales de OAS 3.0, incluyendo la estructura de *paths*, la definición de *schemas* y el uso de referencias ($ref).

IV. **Analizar el Flujo de Trabajo:** Comparar las herramientas de testing (Bruno) con las de documentación (Swagger) para definir las mejores prácticas en el desarrollo de APIs.

2. Introducción a la Documentación de APIs

La documentación de APIs (Application Programming Interfaces) es un elemento crítico en el desarrollo de software. No se trata solo de un requisito, sino de una herramienta fundamental para la **colaboración, el mantenimiento y la adopción** de cualquier servicio web moderno.

3. ¿Por qué es fundamental documentar una API?

Una documentación clara y precisa transforma una API de un código oscuro a un producto consumible.

I. Facilita el Entendimiento y la Adopción
Permite que cualquier desarrollador, ajeno al equipo de backend, comprenda inmediatamente cómo interactuar con el servicio, qué datos enviar y qué respuestas esperar. Esto acelera la integración de la API en proyectos de frontend, aplicaciones móviles o sistemas de terceros.

II. Mejora la Colaboración y Sincronía
Sirve como un contrato claro entre los equipos de desarrollo (frontend, backend) y calidad (QA). Al tener un punto de referencia único y preciso, se minimizan los malentendidos sobre parámetros, códigos de estado y formatos de datos.

III. Reduce Errores y Costos de Mantenimiento

Una documentación exhaustiva previene errores comunes de integración, ya que define explícitamente los tipos de datos, los formatos esperados (validaciones) y las posibles respuestas de error. Esto reduce el tiempo y los costos asociados a la depuración.

IV. **Permite el Testing Integrado**
Las herramientas de documentación moderna, como Swagger UI, ofrecen la capacidad de probar los *endpoints* directamente desde la interfaz, validando el comportamiento de la API en tiempo real sin necesidad de herramientas externas.

4. Definición y Componentes de Swagger

Swagger es un *framework* de código abierto que facilita el diseño, la construcción, la documentación y el consumo de servicios web REST. Se utiliza el formato **YAML** o **JSON** para describir la estructura de la API.

El pilar de Swagger es el **OpenAPI Specification (OAS)**, que es un estándar universal para describir APIs REST.

Componentes Clave del Ecosistema Swagger

Componente	Función Principal
Swagger Editor	Es el entorno donde se **escribe** y se **valida** la especificación OAS (YAML/JSON). Ofrece validación y previsualización en tiempo real.
Swagger UI	Transforma el archivo OAS en una **interfaz gráfica interactiva y navegable**. Permite a los usuarios finales explorar y probar la API.
OpenAPI Specification (OAS)	El estándar formal (la "gramática") que define la estructura del archivo YAML (versión actual utilizada: 3.0.0).
Swagger Codegen	Genera **código cliente o servidor** a partir del archivo OAS, acelerando el desarrollo en múltiples lenguajes (Java, Python, Node.js, etc.).

5. Conceptos Clave del Estándar OpenAPI (OAS 3.0)

El archivo swagger.yaml de este taller sigue la estructura definida por el estándar OAS 3.0. Comprender estos conceptos es crucial para leer y escribir documentación de APIs.

5.1. Estructura Base de OAS

El archivo YAML se organiza en secciones principales que definen el contexto global de la API:

Campo	Descripción	Ejemplo de Uso
openapi: 3.0.0	Especifica la versión de OpenAPI utilizada.	Control de versiones del estándar.
info	Metadata de la API (título, descripción, versión de la API).	Muestra la información general en la parte superior de Swagger UI.
servers	Define la URL base de la API (ruta raíz).	url: https://jsonplaceholder.typicode.com
paths	Contiene la definición de **todos los endpoints** disponibles en la API.	Define las rutas como /posts o /users/{id}.
components	Almacena estructuras reutilizables (schemas, parámetros, respuestas).	Permite definir un objeto Post una sola vez y reutilizarlo.

5.2. Estructura de un *Path* (Endpoint)

Dentro de la sección paths, cada ruta se define con sus respectivos métodos HTTP y sus detalles operativos.

paths:

```
/posts/{id}:
  get:            # <--- Método HTTP
    summary: Obtener un post por ID
    parameters:     # <--- Parámetros necesarios (path, query, header)
    responses:      # <--- Respuestas que puede devolver (200, 404, 500)
```

Uso de components/schemas y Referencias ($ref)

Para evitar la repetición de código y asegurar la consistencia, OAS utiliza **schemas** (modelos de datos) y **referencias**.

- **schemas**: Se definen en la sección components y describen la estructura y tipos de datos de los objetos que se envían o reciben (ej. el objeto Post o User).
- **$ref (Referencia)**: Permite reutilizar un esquema en múltiples lugares de la documentación.

Ejemplo de $ref:

```
# En la respuesta de /posts
schema:
  $ref: '#/components/schemas/Post' # Se referencia el modelo Post
```

5.3. Tipos de Parámetros

Los parámetros son los datos que el cliente debe enviar al servidor. Se definen bajo la sección parameters y deben especificar su ubicación (in):

Ubicación (in)	Descripción	Ejemplo en la Petición
path	Parte de la URL, obligatorios para identificar un recurso.	/posts/{id}
query	Parámetros opcionales agregados al final de la URL.	/comments?postId=1
header	Datos de control (ej. Authorization, Content-Type).	
cookie	Datos almacenados como cookies.	

5.4. Diferencia entre PUT y PATCH

Los métodos PUT y PATCH se usan para actualizar recursos, pero tienen significados distintos en REST:

- **PUT (Reemplazo Completo):** Se utiliza para **reemplazar completamente** el recurso con los datos proporcionados. Si se omite algún campo obligatorio del esquema, se podría borrar o establecer como nulo en el recurso original.
- **PATCH (Actualización Parcial):** Se utiliza para aplicar **modificaciones parciales**. Solo se envían los campos que se desean cambiar, dejando intactos los demás.

6. Comparación de Herramientas: Bruno vs Swagger

Aunque Bruno fue la herramienta utilizada para el *testing* en clase y Swagger se usa para la documentación de este taller, ambas herramientas son fundamentales y tienen propósitos complementarios.

Aspecto	Bruno (Cliente API de Testing)	Swagger (Plataforma de Documentación)
Propósito Principal	**Testing**, Desarrollo y Automatización de Pruebas.	**Documentación**, Visualización y Gobernanza de APIs.
Formato de Archivo	Archivos .bru (formato de texto plano)	YAML / JSON (Estándar **OpenAPI/OAS**)
Interfaz	Aplicación de escritorio o Extensión de VS Code.	Interfaz web interactiva (Swagger UI).
Integración con Código	Se enfoca en probar y guardar colecciones de peticiones.	Genera código (Codegen) y se integra con la construcción de la API (Código Servidor).
Uso Ideal	**Uso interno** por desarrolladores para *debugging* y pruebas unitarias/de integración.	**Uso externo** para *stakeholders*, clientes y desarrolladores de terceros.

La práctica recomendada es **usar ambas**: Bruno para un flujo de trabajo ágil de desarrollo y pruebas internas, y Swagger para generar la documentación pública y mantener el contrato de la API bajo el estándar OAS.

7. Evidencia Práctica y Documentación de Métodos

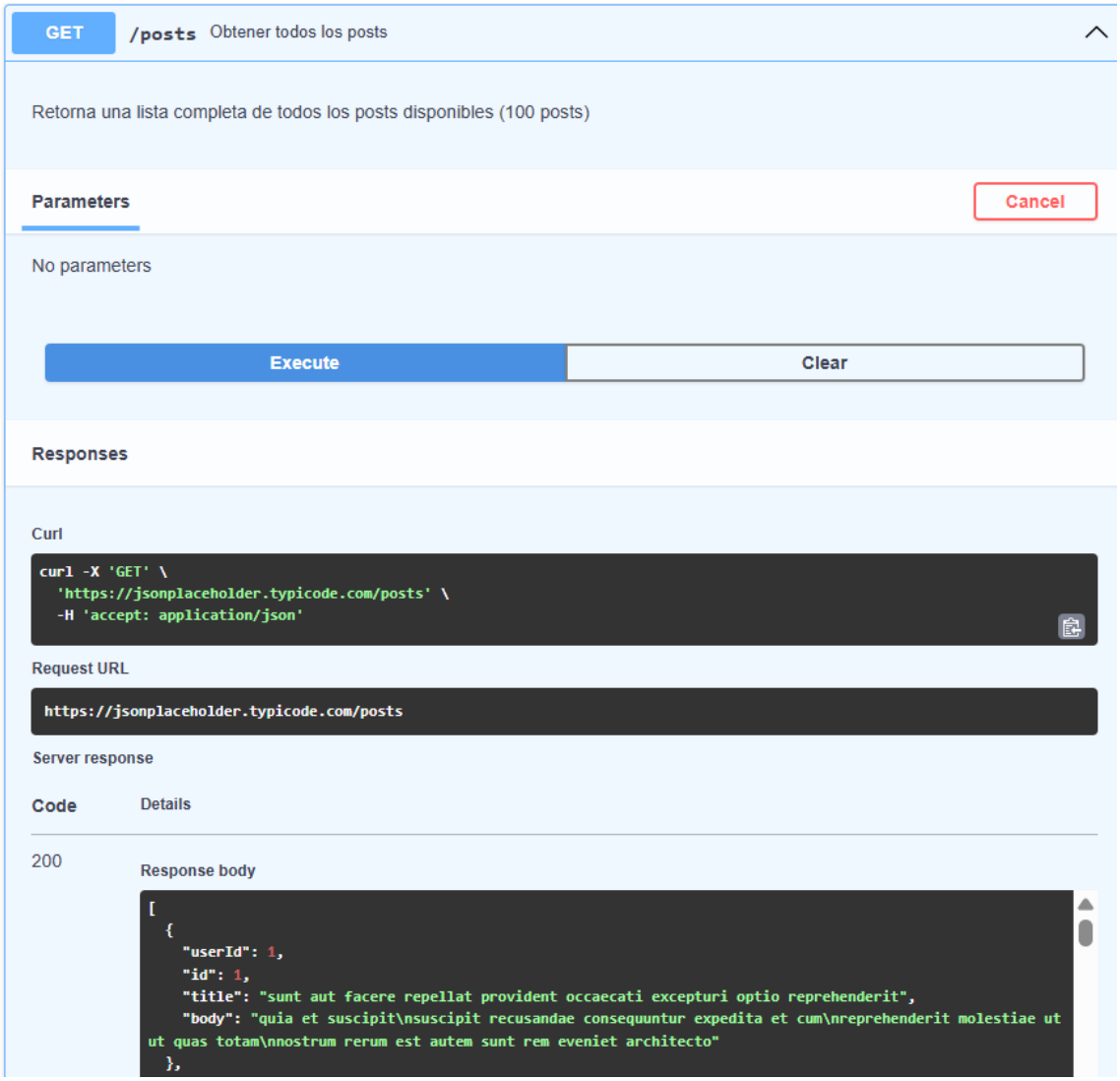
Esta sección documenta la ejecución de las pruebas de los principales métodos REST utilizando las herramientas de testing, validando los conceptos teóricos expuestos anteriormente.

7.1. Prueba de Lectura (GET)

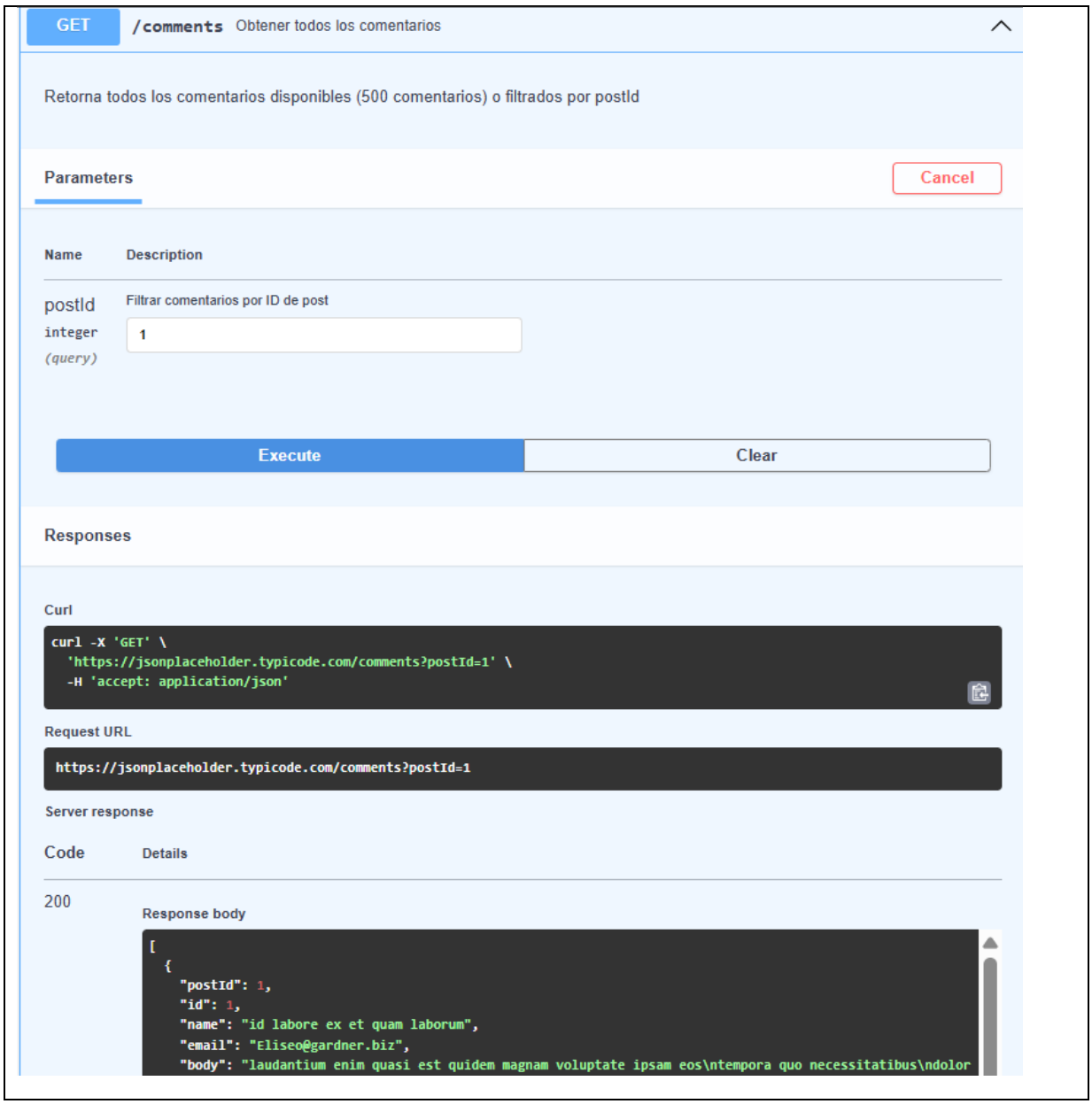
Se prueba la obtención de colecciones y recursos individuales, asegurando el código de respuesta HTTP 200 (OK).

Descripción

GET /posts (Obtener lista)



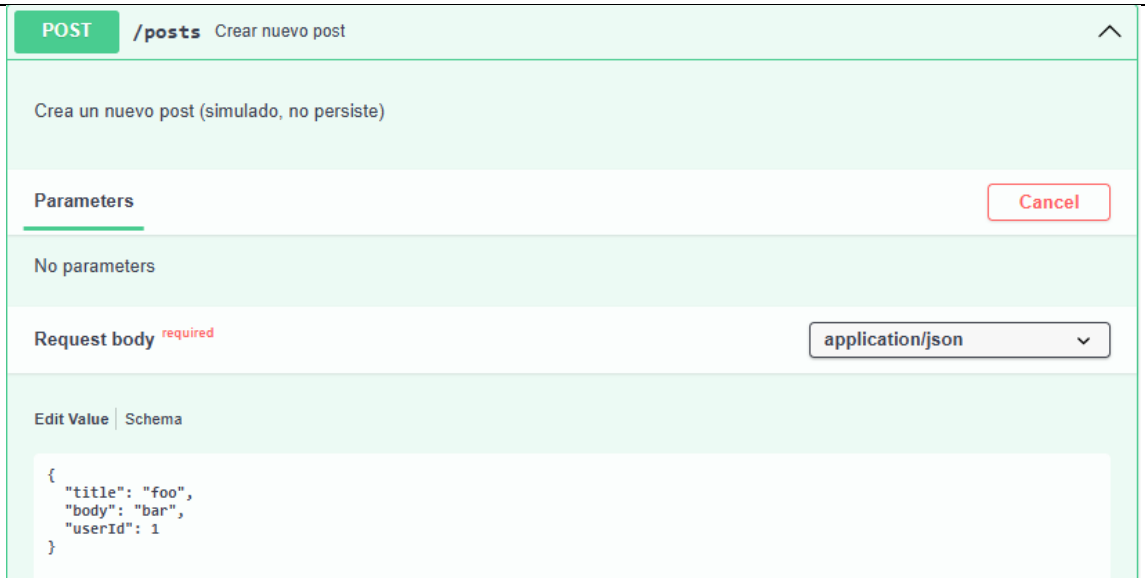
GET /comments?postId=1 (Query Params)



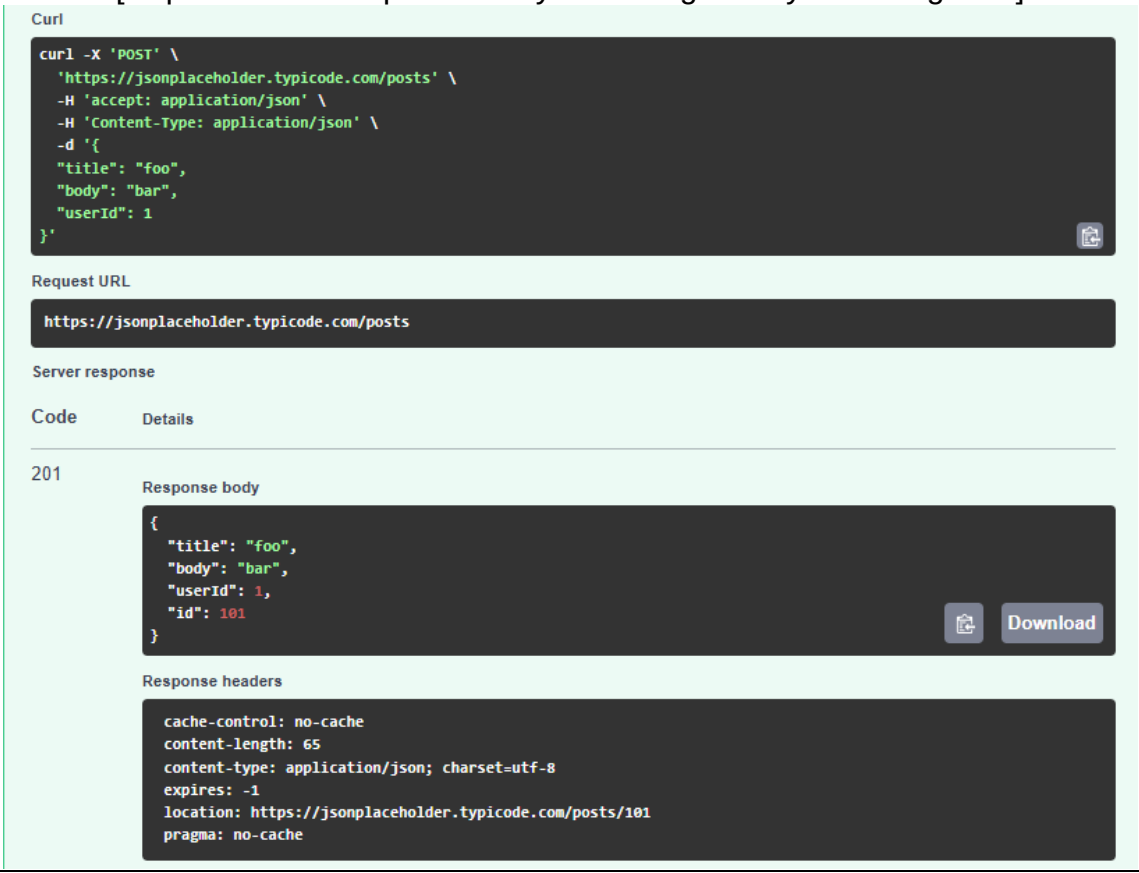
7.2. Prueba de Creación (POST)

Se verifica la creación de un nuevo recurso, validando que la respuesta sea 201 (Created) y que el recurso devuelto contenga el nuevo ID asignado por el servidor.

Descripción
[Captura del Body de la Request (datos a enviar)]

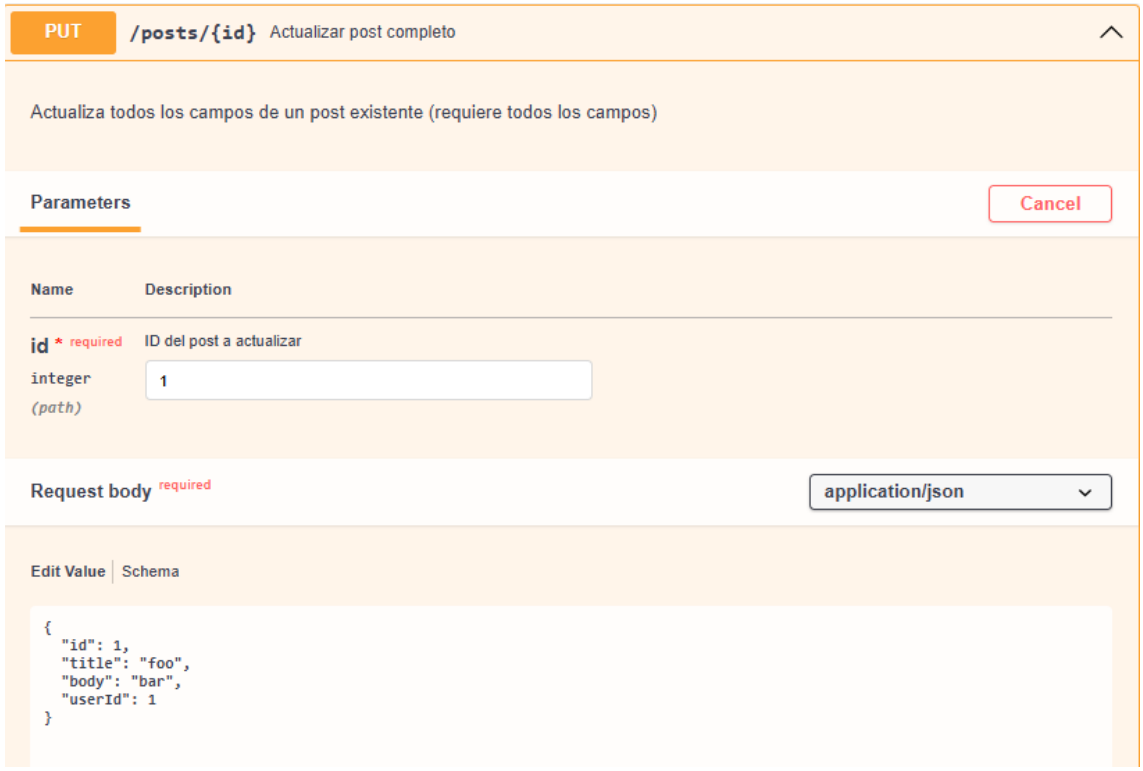


[Captura de la Response Body con código 201 y el ID asignado]



7.3. Prueba de Actualización (PUT y PATCH)

Se documenta la diferencia en el uso de los métodos de actualización: PUT para reemplazo completo y PATCH para actualización parcial. Ambos deben devolver 200 OK.

Descripción
PUT /posts/1 (Reemplazo Total) 

Responses

Curl

```
curl -X 'PUT' \
  'https://jsonplaceholder.typicode.com/posts/1' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": 1,
  "title": "foo",
  "body": "bar",
  "userId": 1
}'
```

Request URL

```
https://jsonplaceholder.typicode.com/posts/1
```

Server response

Code	Details
200	Response body

```
{
  "id": 1,
  "title": "foo",
  "body": "bar",
  "userId": 1
}
```

Download

Response headers

```
cache-control: no-cache
content-type: application/json; charset=utf-8
expires: -1
pragma: no-cache
```

PATCH /posts/1 (Actualización Parcial)

PATCH `/posts/{id}` Actualizar post parcialmente

Actualiza solo los campos especificados de un post

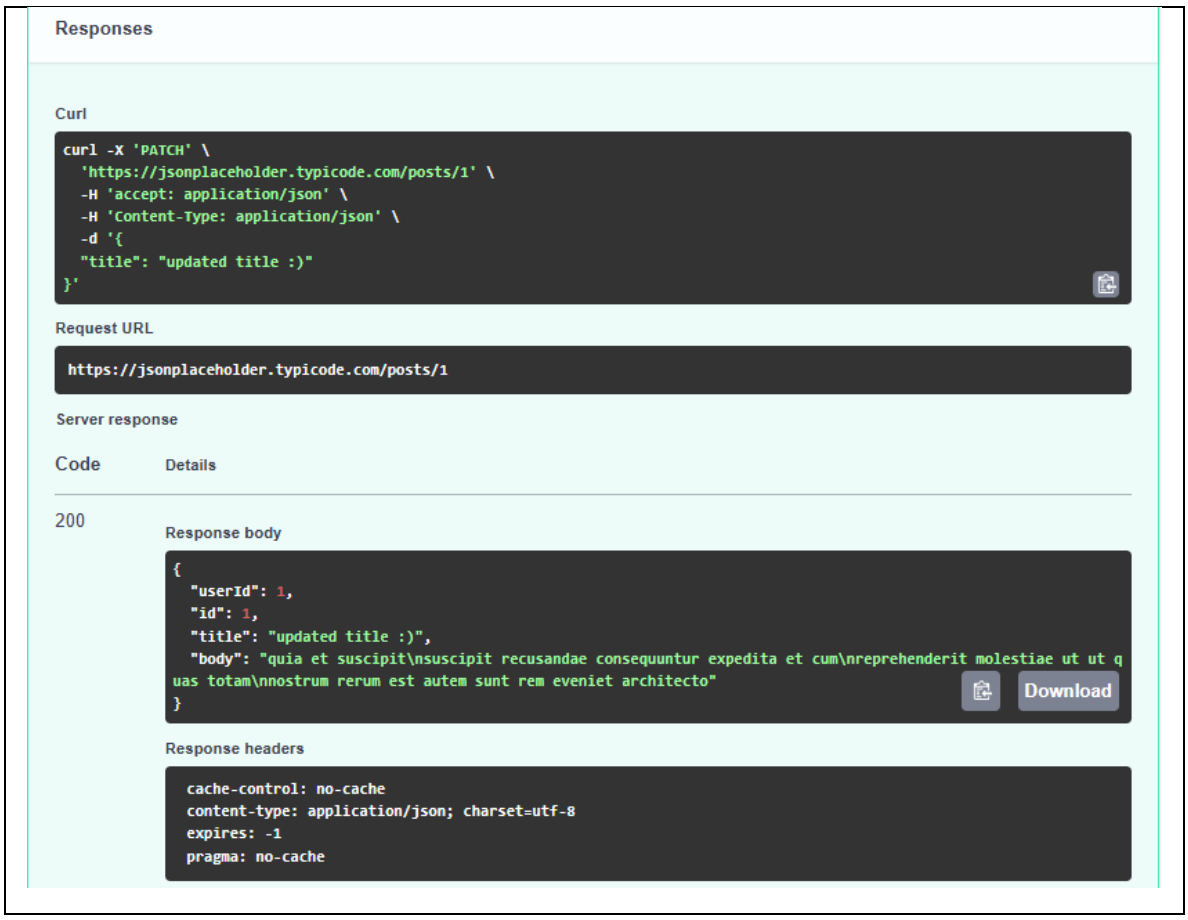
Parameters | Cancel | Reset

Name	Description
id * required integer *(path)*	ID del post a actualizar 1

Request body required — application/json

Edit Value | Schema

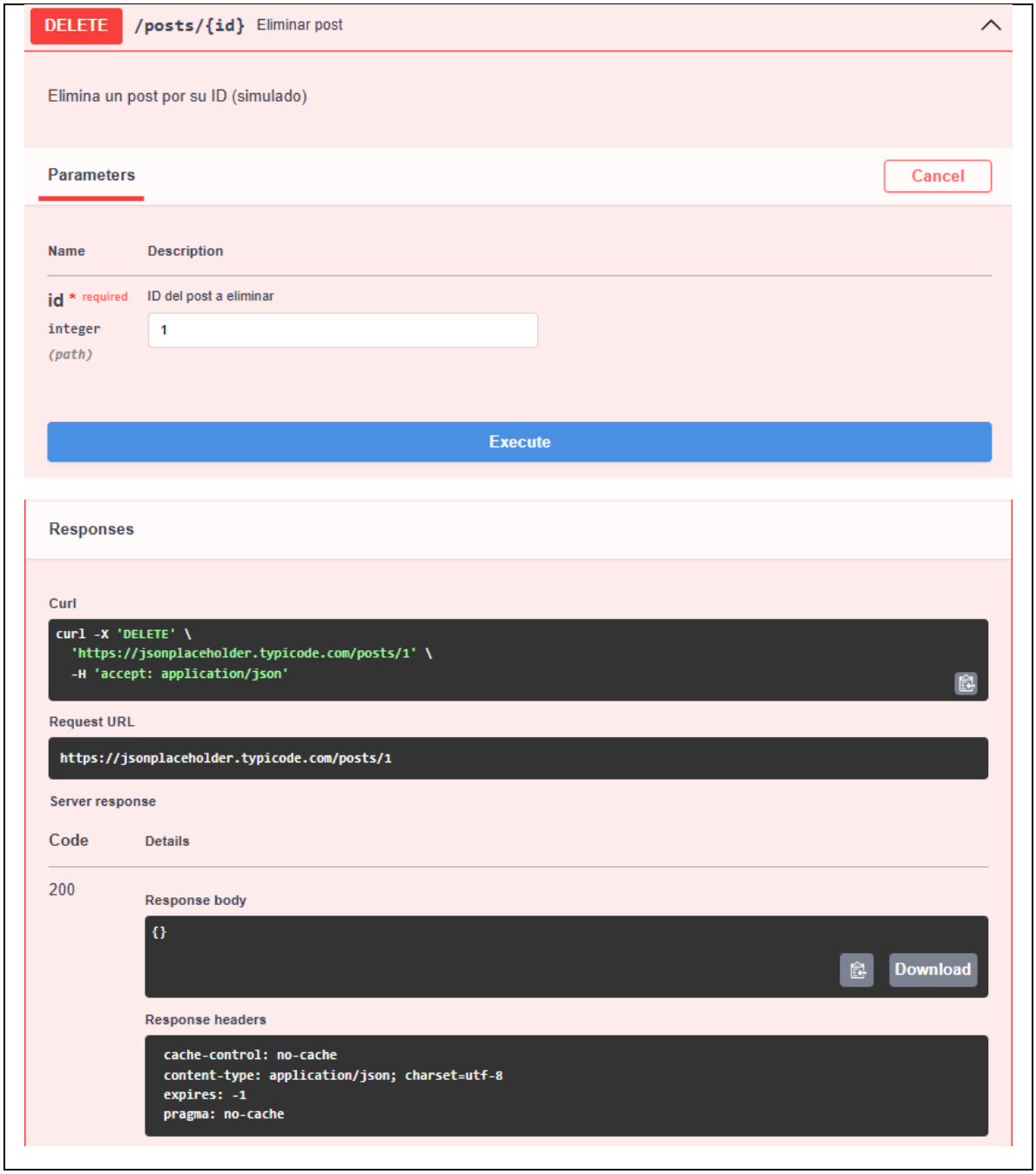
```
{
  "title": "updated title :)"
}
```



7.4. Prueba de Eliminación (DELETE)

Se prueba la eliminación de un recurso. La respuesta esperada es un código 200 OK o 204 (No Content), indicando que la operación se realizó con éxito sin necesidad de devolver un Body.

Descripción
DELETE /posts/1


DELETE
/posts/{id}
Eliminar post
Elimina un post por su ID (simulado)
Parameters
Cancel
Name
Description
id * required
ID del post a eliminar
integer
(path)
1
Execute
Responses
Curl
curl -X 'DELETE' \
'https://jsonplaceholder.typicode.com/posts/1' \
-H 'accept: application/json'
Request URL
https://jsonplaceholder.typicode.com/posts/1
Server response
Code
Details
200
Response body
{}
Download
Response headers
cache-control: no-cache
content-type: application/json; charset=utf-8
expires: -1
pragma: no-cache

8. Conclusiones

La documentación de APIs con Swagger/OpenAPI no es simplemente una tarea administrativa, sino un **proceso estratégico** que define el éxito de un servicio web.

- ✓ **Swagger como Contrato Único:** El estándar **OpenAPI Specification (OAS)** se ha consolidado como el lenguaje universal para describir APIs REST. El archivo YAML/JSON resultante actúa como el **contrato oficial** de la API, garantizando la interoperabilidad entre el frontend y el backend.

- ✓ **Valor de la Interfaz Interactiva:** Herramientas como **Swagger UI** demuestran el poder de la documentación interactiva, transformando el código YAML estático en una herramienta viva que acelera el proceso de adopción y permite el *testing* inmediato.

- ✓ **Complementariedad de Herramientas:** Aunque **Bruno** es indispensable para el *testing* y el flujo de trabajo interno del desarrollador (debido a su enfoque en la practicidad y el control de versiones con Git), **Swagger** es crucial para la documentación externa y la gobernanza de la API. La combinación de un cliente API robusto y un estándar de documentación líder en la industria define un flujo de trabajo de desarrollo moderno y eficiente.

- ✓ **Impacto en el Mantenimiento:** La definición precisa de *schemas* y *paths* mediante OAS reduce significativamente la ambigüedad, lo que se traduce en menos errores, menor tiempo de depuración y una mayor mantenibilidad del proyecto a largo plazo.