Engineer

OpenAPI

descripción de una API HTTP legible por máquinas

Qué es

OpenAPI es un formato estándar para recoger todo lo que necesita saber quien llama a una API HTTP —direcciones, operaciones, entradas, salidas, errores y seguridad— en un único archivo JSON o YAML que pueden leer tanto personas como programas. El propio estándar es la OpenAPI Specification, a menudo abreviada como OAS, y un archivo escrito conforme a ella es una descripción OpenAPI. El formato empezó como Swagger Specification, y ese nombre anterior sobrevive en herramientas construidas a su alrededor, como Swagger UI.

La especificación dice que define una «descripción de interfaz estándar e independiente del lenguaje de programación para API HTTP». Su objetivo declarado es que una persona o un programa pueda saber qué hace un servicio sin su código fuente, documentación adicional ni una captura de su tráfico de red. Se recomienda llamar al documento de entrada `openapi.json` u `openapi.yaml`, y las descripciones grandes suelen repartirse entre varios archivos unidos por referencias.

Swagger 2.0 se publicó en septiembre de 2014 y se donó a la OpenAPI Initiative el 31 de diciembre de 2015. La versión 3.0.0 llegó en julio de 2017 con el nuevo nombre, la 3.1.0 en febrero de 2021 y la 3.2.0 en septiembre de 2025.

Una descripción tiene unas pocas partes de nivel superior. `info` da el nombre y la versión de la API; `servers` enumera dónde se ejecuta; `paths` contiene cada dirección y, bajo ella, una operación por método HTTP, con sus parámetros, cuerpo de petición y posibles respuestas. `components` reúne los elementos reutilizados: esquemas de datos, parámetros, respuestas y esquemas de seguridad que indican cómo se autentica quien llama. Desde la versión 3.1, una sección `webhooks` también puede describir las peticiones que la API envía a sus consumidores. Un fragmento mínimo:

```
openapi: 3.1.0
info:
    title: Orders
    version: 1.4.0
paths:
    /orders/{id}:
        get:
            operationId: getOrder
            parameters:
                - name: id
                  in: path
                  required: true
                  schema:
                      type: string
            responses:
                "200":
                   description: One order
```

Las estructuras de datos se escriben en un lenguaje de esquemas. Desde la versión 3.1, el objeto Schema es un superconjunto de JSON Schema Draft 2020-12; la 3.0 utilizaba un subconjunto ampliado de un borrador anterior, con añadidos propios. JSON Schema describe un elemento de datos, mientras que OpenAPI describe toda una API HTTP y toma prestado JSON Schema para sus partes. Sus vecinos describen otros tipos de interfaz: AsyncAPI cubre sistemas basados en mensajes, los servicios gRPC se declaran en archivos `.proto` de Protocol Buffers y un servidor GraphQL publica su propio esquema.

Utilice el concepto en una decisión

Aplique «¿Qué es OpenAPI?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. La parte que el usuario no ve: dónde se guardan los datos, qué reglas se aplican y cómo pueden comunicarse otros sistemas con ella.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Diseño y operación de 6 flujos ETL/ELT — 6 flujos ETL en tres motores de bases de datos

Diseño y operación de 6 flujos ETL/ELT Póngase en contacto →

Por qué importa

Una descripción en un formato estándar es un archivo que muchas herramientas pueden leer, y cada una ahorra trabajo que un equipo haría a mano. La documentación de referencia se genera a partir de ella. También se generan bibliotecas cliente en varios lenguajes, de modo que un frontend llama a una función tipada en vez de montar una URL. Los servidores de simulación

responden a partir de ella, así que puede construirse un frontend antes de que exista el backend. Las puertas de enlace importan sus rutas, las baterías de pruebas validan el tráfico contra ella y los agentes de IA que llaman a API la leen para saber qué se ofrece.

Ese alcance también es el riesgo. Todas esas herramientas confían en el archivo. Una descripción incorrecta es peor que ninguna, porque el cliente generado compila, la documentación parece una referencia autorizada y el error aparece durante la ejecución en el producto de otra persona.

Por tanto, la decisión principal es de dónde sale la descripción. En el enfoque que parte del diseño, las personas escriben la descripción, la revisan como contrato y después construyen el servidor para ajustarse a ella. La forma de la API se debate antes de que exista código, lo que encaja con API públicas y equipos que trabajan en paralelo. El coste es que la descripción y el servidor son dos cosas que pueden discrepar. En el enfoque que parte del código, la descripción se genera a partir de los tipos y las anotaciones del propio servidor. No puede describir una ruta que el servidor no tenga, pero solo puede ser tan precisa como esos tipos, y el debate de diseño ocurre en la revisión de código en vez de en un documento. En ambos casos, la descripción solo se mantiene fiel si una compilación la comprueba.

El paso de la versión 3.0 a la 3.1 cambió la forma de escribir esquemas, porque la 3.1 adoptó JSON Schema 2020-12. Generadores, validadores y puertas de enlace admiten cada uno su propio intervalo de versiones, así que la versión que escribe un equipo queda limitada por la herramienta más antigua de la cadena.

Cómo se aplica en la práctica

Al principio, Fiber alojaba una API con validación escrita a mano, endpoint por endpoint, y sin un documento único que describiera su estructura. La capa HTTP se migró a Huma v2, sobre Fiber, de modo que cada endpoint se declara con estructuras de entrada y salida y la descripción OpenAPI se genera a partir de esos tipos; la documentación sigue al código en vez de envejecer en una wiki. Los dos endpoints WebSocket permanecen en Fiber directamente, fuera del modelo de petición y respuesta de Huma. La escala explica la importancia de esa descripción: con 760 operaciones repartidas entre 649 rutas, es en la práctica la única forma de que alguien pueda leer la API, por lo que saber que está completa importa más que cuando tenía cincuenta. Una comprobación de CI mantiene la paridad de rutas en el 100 %.

En otro proyecto, la descripción es el eslabón intermedio de una cadena que parte de la base de datos hacia fuera. El esquema de la base de datos y sus funciones dan forma a los tipos Go; Huma emite la descripción y lo que utiliza el frontend se genera a partir de ella: un cliente TypeScript tipado de 44.076 líneas, 61 manejadores de simulación para las pruebas del frontend y los límites que aplica cada formulario. Aborda la divergencia entre frontend y backend, que suele manifestarse como un 422 en el navegador una vez publicado el desajuste. Una comprobación del contrato en CI hace fallar la compilación si hay divergencia, con una prueba dirigida a los campos opcionales del cuerpo, donde se confunden «ausente» y «null», y a las enumeraciones de parámetros de consulta. El coste es un paso de generación en medio de todo y una cadena tan fiable como su eslabón menos protegido.

La página de la empresa sobre preparación para agentes de IA aplica el mismo razonamiento a los agentes. Una aplicación que puedan utilizar ofrece una API descrita: un documento OpenAPI, un servidor MCP si se recomienda que los agentes la llamen directamente y metadatos OAuth para que un agente actúe por un usuario sin recibir su contraseña. La página clasifica cada recomendación en lugar de obedecerla sin más y, como el propio sitio de la empresa no tiene API, el catálogo de API y los metadatos OAuth son dos de las cinco que descartó para ese sitio.

Dónde surgen los problemas

La descripción promete algo que el servidor no hace. Una descripción mantenida a mano se queda atrás: un parámetro de consulta nuevo, un campo renombrado o un código de error que solo conoce el servidor. Los clientes generados compilan entonces contra una ficción. Genere la descripción desde el código o manténgala separada y valide peticiones y respuestas reales contra ella en la batería de pruebas, y compare las rutas que registra el servidor con las que enumera la descripción.

Cada regeneración del cliente cambia el nombre de la mitad de los métodos. Sin `operationId`, los generadores inventan nombres de métodos a partir de la ruta y del método HTTP, de modo que mover una ruta cambia el nombre de la función que utilizan todos sus consumidores. La especificación exige que cada `operationId` sea único y distingue mayúsculas de minúsculas. Asigne uno a cada operación, elíjalo con tanto cuidado como el nombre de una función pública y haga que un linter impida compilar si falta alguno.

Los campos que admiten null empiezan a rechazarlo tras una actualización. OpenAPI 3.0 los marcaba con `nullable: true`. En la versión 3.1 esa palabra clave desaparece y el tipo pasa a ser una lista; además, un validador suele ignorar una palabra clave que no conoce. Por tanto, un esquema 3.0 simplemente etiquetado como 3.1 rechaza null donde su autor quería permitirlo. Convierta los esquemas con la guía de actualización de la OpenAPI Initiative, que también explica cómo `exclusiveMinimum` pasa a ser un número:

```
# OpenAPI 3.0
type: string
nullable: true
---
# OpenAPI 3.1
type: [string, "null"]
```

Los ejemplos incumplen su propio esquema. La especificación dice que un ejemplo debería ajustarse a su esquema y deja la comprobación a las herramientas. Un lector copia el ejemplo de la documentación y obtiene un 422; un servidor de simulación lo sirve y el frontend se construye contra datos que la API real rechazará. Valide cada ejemplo en CI con el mismo rigor que el tráfico real.

La descripción pública revela la API de administración. Una única descripción generada desde todas las rutas y servida en una dirección pública enumera las operaciones internas y sus esquemas a cualquiera que la pida. Publique una descripción por público destinatario, generada con las operaciones internas excluidas. La especificación permite dejar objetos vacíos o eliminarlos precisamente para este tipo de control de acceso.

Lecturas adicionales

- Especificación OpenAPI — versión más reciente del estándar, con la definición de cada objeto y campo y el historial de cada publicación.
- JSON Schema Draft 2020-12 — lenguaje de esquemas que OpenAPI 3.1 y posteriores utilizan para cada cuerpo de petición y respuesta.
- Actualización de OpenAPI 3.0 a 3.1 — lista de cambios de esquemas de la propia OpenAPI Initiative, entre ellos `nullable`.
- Huma: generación de OpenAPI — cómo el marco Go utilizado en los proyectos anteriores deriva una descripción de los tipos de los manejadores.

Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es OpenAPI?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/openapi/