

Gobierno de APIs. (API Owner)

Swagger

Índice del capítulo

- ▶ De un vistazo.
- ▶ Especificaciones.
- ▶ Swagger Open Source.
- ▶ Swagger Inspector.
- ▶ SwaggerHub.

De un vistazo

- ▶ Introducción.
- ▶ ¿Qué es Swagger?
- ▶ ¿Para qué se utiliza Swagger?
- ▶ Ventajas.

Introducción

- ▶ En los **proyectos** de desarrollo, la **documentación** **colleva mucho tiempo**, y las personas externas suelen **desconocer los beneficios** que puede **aportar** al mantenimiento y el desarrollo continuo de sistemas.
- ▶ Uno se da cuenta de esto cuando **dichas tareas no son realizadas por el equipo** que ha desarrollado el sistema, lo que tiende a ser **más la norma que la excepción**.
- ▶ Todavía más importante es la **documentación de las APIs**, es decir, de las application programming interfaces o, en español, interfaces de programación de aplicaciones. Estas conectan **aplicaciones entre sí**, y las propias aplicaciones con **fuentes de datos** y otros **sistemas**.
- ▶ En la actualidad, **apenas se dan escenarios** de aplicación en los que un conjunto variado de APIs **no tenga** un **papel importante** en las conexiones de software. Pero, para poder utilizar las interfaces, también **hay que conocer su estructura y funciones**.

¿Qué es Swagger?

- ▶ La **especificación OpenAPI Swagger** hace esto posible, dado que, al ser un formato de **código abierto**, ayuda a mantener el control sobre las diferentes capacidades de las APIs. Con Swagger también es posible que las personas no involucradas directamente en el proceso de desarrollo faciliten APIs: **ven las interfaces utilizadas y pueden documentarlas**.
- ▶ Debido al **amplio abanico de tecnologías** y lenguajes de programación, describir **las APIs** era una **tarea muy compleja en el pasado**.
- ▶ Para el desarrollo de las APIs, actualmente, se utiliza el **paradigma de programación REST**. Sitios web como Google, Amazon y Twitter utilizan APIs RESTful.
- ▶ Anteriormente, se describían las interfaces en el **lenguaje de descripción de servicios web WSDL**.
- ▶ Desde el punto de vista técnico, con **WSDL 2.0** también se pueden describir APIs REST, lo que es muy complicado para los desarrolladores. El lenguaje de descripción de aplicaciones web (WADL) debía solucionar este problema, pero **nunca pudo llegar a estandarizarse** por su **estructura XML**.

¿Qué es Swagger?

- ▶ Por eso, **Swagger** se ha convertido en poco tiempo en **la tecnología más popular para las documentaciones de APIs**. Mejor dicho, en la tecnología más popular para **las APIs REST** utilizadas con más frecuencia.
- ▶ **Swagger** fue desarrollado por **Reverb**, pero actualmente se caracteriza por su neutralidad como código abierto al abrigo de la **Fundación Linux**, llamada **Open API Initiative**.
- ▶ Con el tiempo, **Swagger** se ha renombrado como **especificación OpenAPI**, aunque sigue siendo conocido de manera no oficial con el nombre de Swagger.

¿Qué es Swagger?

- ▶ Dependiendo del ámbito de aplicación, el **elemento principal** de Swagger es un **archivo JSON o YAML**.
- ▶ La interfaz de usuario con la que se pueden **crear documentaciones fácilmente** está basada en HTML y JavaScript.
- ▶ Básicamente, Swagger tiene un **formato de lenguaje neutral** y legible por máquina. A través de la interfaz de usuario no solo se puede gestionar la documentación, sino que Swagger también puede realizar pruebas ad hoc.
- ▶ Otra **ventaja de Swagger** se hace patente en la posibilidad de desarrollo gracias a la compatibilidad con la librería central, **Swagger Core**. Se complementa con la interfaz de usuario llamada **Swagger UI**, el generador de código **Swagger Codegen** y el **Swagger Editor**.

¿Qué es Swagger?

- ▶ Pero, **si por algo destaca Swagger**, al igual que otras opciones de código abierto, es su relación con el **amplio ecosistema de GitHub**. Los desarrolladores encuentran aquí generadores de código para casi todos los lenguajes de programación. Swagger documenta cada interfaz con toda la información.
- ▶ El archivo de la documentación comienza con **la indicación de la versión de especificación** empleada.
- ▶ Después, siguen los **datos generales sobre la API** clasificados de manera clara en la categoría "info".
- ▶ Swagger también disgrega el alojamiento, la ruta y el esquema URL y los especifica individualmente. Ya después vienen el tipo de medio para toda la API.
- ▶ Este procedimiento funciona como un **complejo sistema de fichas**.

¿Qué es Swagger?

- ▶ Una vez finalizada la **categorización**, se presenta **el contenido**: rutas, operadores y parámetros se procesan junto con la descripción correspondiente.
- ▶ Los **tipos de datos** se gestionan **cada uno en su sección**.
- ▶ Con **Swagger UI** se puede visualizar el **conjunto no solo de forma textual**, sino también **gráficamente**. Esto supone una ventaja sobre todo si se tienen que enviar solicitudes de prueba directamente desde el navegador.
- ▶ Esta combinación de **documentación perfecta y posibilidad de envío directo** de solicitudes API es lo que hace a Swagger tan valioso.

¿Para qué se utiliza Swagger?

- ▶ Para el desarrollo de APIs es esencial contar con una **documentación ordenada y comprensible**, pues solo así pueden utilizarse las interfaces de los desarrolladores.
- ▶ Esto es aún más importante para las **APIs públicas**: sin documentación, estas son inservibles para la comunidad, no se pueden divulgar y **no suelen tener éxito**.
- ▶ En estos momentos, **Swagger es la mejor opción para documentar APIs de REST**, porque puede representar casi todos los servicios web y la información en torno a la interfaz.
- ▶ Crece al **mismo ritmo** que el sistema y documenta los cambios automáticamente. El funcionamiento no acarrea problemas, porque **Swagger deposita la documentación de la API de REST directamente en el código**.

¿Para qué se utiliza Swagger?

- ▶ El punto de partida de cualquier desarrollo API es o bien el **código de programa (Code First)** o la **descripción de la interfaz (Design First)**.
- ▶ Si hablamos de un desarrollo que sigue **la máxima Code First**, el punto de partida establecido es el código de programa. A partir de este, **Swagger puede deducir directamente una documentación** independiente del lenguaje de programación utilizado y legible tanto por máquinas como por seres humanos.
- ▶ Por el otro lado, está el **paradigma Design First**. Como ya hemos mencionado, diferentes desarrolladores son responsables de un cliente y un servidor. En el paradigma Design First se finaliza **primero la descripción** y, a continuación, se generan los **códigos fácilmente con Swagger**. Para ello hay generadores para **todos los lenguajes de programación** habituales e incluso plantillas que se pueden adaptar o ampliar.
- ▶ **Swagger** ofrece casi de manera automática un **código API coherente**. Si algo no cuadra en la programación manual, aparecen errores de compilación visibles automáticamente en un sistema de integración continua.

Ventajas

- ▶ Sus **ventajas predominan** de tal manera que **Swagger** puede definirse como **la aplicación estándar** por excelencia para la descripción de interfaces en las API de RESTful. Como otras tantas aplicaciones de código abierto, Swagger disfruta de una **gran divulgación** y, con ello, de **compatibilidad** con muchas herramientas.
- ▶ El gremio de Swagger está formado por grandes de la tecnología como **Microsoft, IBM y Google**, con lo que la especificación OpenAPI cuenta con respaldo incondicional, aunque haya alternativas como **Restful API Modelling Language (RAML)**. Este también se basa en YAML y desarrolla definiciones todavía más complejas que Swagger, pero incluso su creador (Mulesoft) ha entrado a formar parte de la **OpenAPI Initiative**.
- ▶ Una **pequeña desventaja de Swagger** es la **comprensión y legibilidad** de la documentación. Aunque Swagger ofrece un formato relativamente bien elaborado, se requiere un tiempo para familiarizarse con él.

Especificaciones

- ▶ Introducción.
- ▶ ¿Cuáles son las áreas de aplicación de OpenAPI?
- ▶ Ventajas.
- ▶ Versiones disponibles.
- ▶ Trabajando con un editor.
- ▶ Sintaxis.

Introducción

- ▶ **OpenAPI** es un **estándar** para la descripción de las interfaces de programación, o application programming interfaces (API).
- ▶ La **especificación OpenAPI** define un formato de descripción abierto e independiente de los fabricantes para los servicios de API.
- ▶ En particular, OpenAPI puede utilizarse para **describir, desarrollar, probar y documentar** las APIs compatibles con REST.
- ▶ La actual **especificación OpenAPI** surgió del proyecto predecesor Swagger. La empresa de desarrollo SmartBear sometió la especificación existente de Swagger a una licencia abierta y dejó el mantenimiento y desarrollo posterior en manos de la iniciativa OpenAPI. Además de SmartBear, entre los miembros de la iniciativa OpenAPI se encuentran gigantes de la industria como Google, IBM y Microsoft. La Fundación Linux también apoya este proyecto.

Introducción

- ▶ Una cuestión que puede causar **confusión** es la distinción entre **OpenAPI** y **Swagger**.
- ▶ OpenAPI es una **especificación**, es decir, una **descripción abstracta** que no está ligada a una aplicación técnica concreta.
- ▶ **Hasta la versión 2.0**, esta especificación todavía se llamaba **Swagger** y luego fue renombrada como especificación **OpenAPI**. Sin embargo, las herramientas proporcionadas por SmartBear, la empresa que la desarrolló originalmente, siguen existiendo con el nombre de Swagger.
- ▶ Con OpenAPI, una API puede describirse de **manera uniforme**. Esto se conoce como “definición API” y se genera en un formato legible por máquina. En particular, se utilizan dos lenguajes: **YAML** y **JSON**.

¿Cuáles son las áreas de aplicación de OpenAPI?

- ▶ En general, **OpenAPI** se utiliza para describir **APIs REST de manera uniforme**. Como esta descripción, es decir, la definición API, está disponible en un formato legible por máquina, se pueden generar automáticamente diversos artefactos virtuales a partir de ella. En concreto, estos incluyen:
 - ▶ **Creación de documentación API:** La documentación basada en HTML se genera automáticamente a partir de la definición API legible por máquina. Esta sirve como material de consulta para los desarrolladores que acceden a los servicios API. Si la definición API cambia, la documentación se vuelve a generar para que ambas concuerden.
 - ▶ **Creación de conexiones en diferentes lenguajes de programación:** Con las herramientas apropiadas, se puede crear una biblioteca de software adecuada del lado del cliente a partir de la definición API en un lenguaje de programación compatible. Esto permite a los programadores de todo tipo acceder a la API. La biblioteca de software se incorpora de manera convencional. Por lo tanto, el acceso a los servicios de API tiene lugar, por ejemplo, mediante el acceso a las funciones dentro del mismo entorno de programación.
 - ▶ **Elaboración de casos de prueba:** Cada componente de un software debe someterse a diversas pruebas para asegurar su funcionalidad. En concreto, es preciso volver a probar un componente de software cada vez que se cambia el código subyacente. A partir de la definición API, se pueden generar estos casos de prueba automáticamente para poder comprobar la funcionalidad de los componentes del software en todo momento.
- ▶ En última instancia, cabe señalar que **no todas las APIs pueden representarse** utilizando OpenAPI. Sin embargo, las **API REST son compatibles sin duda**.

Ventajas

- ▶ En general, la **ventaja de OpenAPI** es que la puesta en marcha, documentación y prueba de una API son coherentes y constantes durante el desarrollo y el mantenimiento.
- ▶ Además, el **uso de la especificación OpenAPI** permite una **mejor coordinación** del desarrollo de la API entre los equipos de backend y frontend.
- ▶ En ambos equipos, los **componentes del código** pueden generarse a partir de la **definición API** para que tanto en backend como en frontend puedan desarrollarlos y probarlos sin tener que esperar al otro.
- ▶ Además de estas ventajas generales, OpenAPI se utiliza en particular como estándar de base para el desarrollo de API REST. Esto es atractivo, porque desarrollar una API compatible con REST de forma manual no es una trivialidad. Sin embargo, las API REST ofrecen algunas ventajas. Por ejemplo, REST se ejecuta sobre HTTP/S, y los puertos para ello están abiertos en cualquier cortafuegos.

Ventajas

- ▶ Además, el uso de OpenAPI ofrece las siguientes ventajas:
 - ▶ Definir las API HTTP **independientemente de un lenguaje** de programación específico.
 - ▶ Generar código de servidor para una API definida en OpenAPI.
 - ▶ Generar **bibliotecas del lado del cliente** para una API compatible con OpenAPI en más de **40 lenguajes de programación**.
 - ▶ Procesar una definición OpenAPI con las herramientas apropiadas.
 - ▶ Crear **documentación** interactiva de API.
 - ▶ Permitir que las personas y las máquinas descubran y entiendan las capacidades de un servicio sin tener que mirar el código fuente o la documentación adicional.
 - ▶ Acceder a los servicios de API con un gasto mínimo de puesta en marcha.

Versiones disponibles

Versión	Nombre	Estado
1.0, agosto de 2011	Especificación Swagger	No está en uso
2.0, septiembre de 2014	Especificación Swagger > especificación OpenAPI	Todavía en uso
3.0, julio de 2017	Especificación OpenAPI	Todavía en uso

Versiones disponibles

- ▶ Versión 3.1.0:
 - ▶ <https://swagger.io/specification/>



The screenshot shows the top navigation bar of the Swagger website. It includes the Swagger logo (a green circle with white curly braces), the text "Swagger™ Supported by SMARTBEAR", and a search icon. The navigation menu has three items: "Why Swagger? ▾", "Tools ▾", and "Resources ▾". To the right of the menu are "Sign In" and "Try Free" buttons.

OpenAPI Specification

Version 3.1.0

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [BCP 14](#) [RFC2119](#) [RFC8174](#) when, and only when, they appear in all capitals, as shown here.

This document is licensed under [The Apache License, Version 2.0](#).

Version 3.1.0
Introduction
Table of Contents
Definitions
Specification
Appendix A: Revision History

Versiones disponibles

- ▶ A continuación, se exponen las novedades más importantes del cambio de la versión 2.0 a la 3.0:
- ▶ **Nuevo nombre del objeto raíz:**
 - ▶ Con el lanzamiento de la versión 3.0, se introdujo el objeto OpenAPI, que sustituye el objeto Swagger utilizado anteriormente:

```
# <= 2.0
"swagger": "2.0"
# >= 3.0
"OpenAPI": "3.0.0"
```

Versiones disponibles

- ▶ **Múltiples hosts/servidores:**
 - ▶ A partir de la versión 3.0 de OpenAPI, se puede acceder a una API desde **más de un servidor**. Además, es posible definir partes de la URL del servidor de forma variable.
 - ▶ A continuación, mostramos un ejemplo.

```
"servers": [  
  {  
    "url": "https://{{username}}.example.com:{{port}}/{{basePath}}",  
    "description": "The production API server",  
    "variables": {  
      "username": {  
        "default": "demo",  
        "description": "this value is assigned by the service provider, in this example `example.com`"  
      },  
      "port": {  
        "enum": [  
          "8443",  
          "443"  
        ],  
        "default": "8443"  
      },  
      "basePath": {  
        "default": "v2"  
      }  
    }  
  }  
]
```

Versiones disponibles

- ▶ **Nuevos objetos componente y objetos referencia:**
 - ▶ Los **objetos componente y objetos referencia** añadidos en la versión 3.0 de OpenAPI son una de las principales novedades.
 - ▶ El **objeto componente** permite definir múltiples **objetos reutilizables** dentro de la definición API. Los componentes denominados de esta manera se incluyen dentro de la definición API usando la estructura especial `$ref`.
 - ▶ Mediante los componentes y referencias, es posible elaborar una definición API a partir de varias partes reutilizables. Esto **facilita la lectura y reduce el tamaño** de todo el documento.
 - ▶ A continuación, presentamos un ejemplo de la definición oficial de API de GitHub:
 - ▶ Esquema de un repositorio de código GitHub, definido como un objeto componente.
 - ▶ Definición de licencia, referida a través de un componente definido externamente.

```
"components": {  
    "schemas": {  
        // Esquema Repository  
        "repository": {  
            "title": "Repository",  
            "description": "A git repository",  
            "type": "object",  
            "properties": {  
                "id": {  
                    "description": "Unique identifier of the repository",  
                    "example": 42,  
                    "type": "integer"  
                },  
                "node_id": {  
                    "type": "string",  
                    "example": "MDEwOlJlcG9zaXRvcnkxMjk2MjY5"  
                },  
            }  
        }  
    }  
}
```

```
"name": {  
    "description": "The name of the repository.",  
    "type": "string",  
    "example": "Team Environment"  
},  
"full_name": {  
    "type": "string",  
    "example": "octocat>Hello-World"  
},  
// Definición de licencia  
"license": {  
    "nullable": true,  
    "allOf": [  
        {  
            // Referencia a componente definido externamente  
            "$ref": "#/components/schemas/license-simple"  
        }  
    ]  
},
```

Trabajando con un editor

- Podemos trabajar con el editor que más nos guste, por ejemplo, **Visual Studio Code** tiene un **plugin** específico para trabajar en estos entornos:

Extensión: OpenAPI (Swagger) Editor X



OpenAPI (Swagger) Editor v4.15.16

42Crunch | 489.268 | ★★★★★(30)

OpenAPI extension for Visual Studio Code

Habilitar Desinstalar ⚡ ⚙

El usuario ha deshabilitado esta extensión de forma global.

DETALLES CONTRIBUCIONES DE CARACTERÍSTICAS REGISTRO DE CAMBIOS DEPENDENCIAS

OpenAPI extension for Visual Studio Code

This [Visual Studio Code](#) (VS Code) [extension](#) adds rich support for the [OpenAPI Specification \(OAS\)](#) (formerly known as Swagger Specification) in JSON or YAML format. The features include, for example, SwaggerUI and ReDoc preview, IntelliSense, linting, schema enforcement, code navigation, definition links, snippets, static security analysis, and more!

Categorías

Programming Languages
Snippets Linters

Recursos de extensión

Sintaxis

- ▶ Objeto OpenApi.
- ▶ Componentes.
- ▶ Esquemas.
- ▶ Parámetros.
- ▶ Paths y operaciones.
- ▶ Respuestas.
- ▶ Cuerpo de la petición.
- ▶ Autenticación y autorización.

Objeto OpenApi

- ▶ El **objeto OpenApi** es la **raíz** de nuestro documento, que a su vez contiene campos básicos.
- ▶ Veamos un ejemplo de alguno de estos **campos**:

```
openapi: '3.0.2'  
info:  
  title: API Title  
  version: '1.0'  
servers:  
  - url: https://api.server.test/v1
```

Objeto OpenApi

- ▶ En este bloque podemos observar **varios campos**:
 - ▶ **openapi**: Definirá la **versión** de nuestra **especificación** de openapi utilizada.
 - ▶ **info**: Contendrá un objeto con información de utilidad sobre nuestro api.
 - ▶ **servers**: Array con las urls de los diferentes entornos donde está alojada nuestra API, por ejemplo:

```
servers:  
  - url: 'https://antoniofernandez.com'  
    description: 'Entorno de producción de apis'  
  - url: 'https://development.antoniofernandez.com'  
    description: 'Entorno de desarrollo de apis'
```

Objeto OpenApi

- ▶ (cont.):
 - ▶ **paths**: Rutas de nuestra aplicación.
 - ▶ **components**: Componentes que reutilizaremos a lo largo de nuestro documento.
 - ▶ **security**: Seguridad general aplicada a todos los endpoints de nuestro API.

Componentes

- ▶ Son objetos **reutilizables**.
- ▶ En este apartado definiremos aquellos **componentes que podremos reutilizar** a lo largo de nuestro documento, como por ejemplo, los **modelos** de nuestra implementación.
- ▶ Podemos observar diferentes secciones:

```
components:
```

```
  parameters:
```

```
  responses:
```

```
  schemas:
```

```
  securitySchemes:
```

Esquemas

- ▶ En esta sección definiremos las **entidades utilizadas** en nuestra API, es decir los datos que mostrará o consumirá esta misma.
- ▶ El objetivo es definir la **estructura básica de los datos** con los que trabajamos (podríamos decir que los modelos de base de datos).
- ▶ Una vez que los **hayamos definido en esta sección**, podremos **asignarlos a las operaciones** que los consuman o devuelvan.

schemas:

User:

type: object

required:

- email

- password

- role

properties:

id:

type: string

format: uuid

readOnly: true

username:

type: string

password:

type: string

format: password

Esquemas

- ▶ En este bloque estamos definiendo **un objeto User**, que se corresponderá con nuestro modelo User en nuestra API. A continuación, enumero las propiedades utilizadas:
 - ▶ **required**: Array con los campos que son requeridos a la hora de crear un registro de nuestro modelo
 - ▶ **properties**: Campos del objeto.

Esquemas

- ▶ Cada campo a su vez, se describirá utilizando diferentes propiedades:
 - ▶ **type**: tipo de dato (integer, string, object)
 - ▶ **readOnly/writeOnly**: Indica si es un campo que solo se muestra cuando realizamos peticiones de consulta, pero no se necesita enviar cuando insertamos o actualizamos, o al revés. Normalmente los **campos id** suelen ser un **candidato ideal** para ser readOnly, ya que cuando creamos nuestro modelo, no necesitamos enviarlo, sino que se genera de forma automática.
 - ▶ **format**: Indica diferentes formatos para el tipo string (password, email, uuid, date)
 - ▶ **enum**: Listado de posibles valores que puede tomar la propiedad.
 - ▶ Más adelante podremos ver cómo definir estos modelos en línea (sin necesidad de ser globales), no obstante como la mayoría se reutilizan en varios métodos de nuestro API, la forma más óptima es crearlos dentro del **objeto global components.schemas**.

Parámetros

- ▶ Introducción.
- ▶ Atributos del parámetro.
- ▶ Esquema del parámetro.
- ▶ Parámetros en el path.
- ▶ Parámetros en la consulta.
- ▶ Parámetros globales.
- ▶ Más información.

Introducción

- ▶ Los **parámetros** se pueden definir a **nivel de operación** o **path** y también de forma **global** en el objeto `components.parameter`.
- ▶ Ejemplo:

```
/users:  
  get:  
    summary: listar usuario  
    description: Este método lista usuarios  
    parameters:  
      - name: name  
        in: query  
        description: 'Buscar por nombre'  
        schema:  
          type: string
```

Atributos del parámetro

- ▶ Podemos observar varios **atributos importantes**:
 - ▶ **name**: Nombre del parámetro, es decir , el nombre con el que lo capturaremos.
 - ▶ **in**: Indicándonos la localización del parámetro (query, path, header, cookie), en función de si el parámetro está presente en la url, como parámetro del path, en una cabecera o en una cookie.
 - ▶ **description**: Información descriptiva.
 - ▶ **required**: obligatorio o no.
 - ▶ **example**: Podemos indicar un valor de ejemplo de este parámetro.
 - ▶ **schema.type**: Tipo de dato del parámetro.

Esquema del parámetro

- ▶ Se define mediante dos atributos: **type** y **format**.
- ▶ Ejemplo:

```
parameters:  
  - name: petId  
    in: path  
    description: ID of pet to return  
    required: true  
    schema:  
      type: integer  
      format: int64
```

Esquema del parámetro

- La especificación define los tipos asociados a formatos de la siguiente forma:

type	format	Comments
integer	int32	signed 32 bits
integer	int64	signed 64 bits (a.k.a long)
number	float	
number	double	
string	password	A hint to UIs to obscure input.

Esquema del parámetro

- ▶ Se pueden definir parámetros como **arrays**:

```
parameters:  
  - name: tags  
    in: query  
    description: Tags to filter by  
    required: false  
    explode: true  
    schema:  
      type: array  
      items:  
        type: string
```

Esquema del parámetro

- Se pueden definir parámetros como **enumerados** asignando valor por defecto:

```
parameters:  
  - name: status  
    in: query  
    description: Status values that need to be considered for filter  
    required: false  
    explode: true  
    schema:  
      type: string  
      default: available  
      enum:  
        - available  
        - pending  
        - sold
```

Parámetros en el path

- ▶ Ejemplo: atributo **in igual a path**.

```
/pet/{petId}:
  get:
    ...
    summary: Find pet by ID
    description: Returns a single pet
    operationId: getPetById
    parameters:
      - name: petId
        in: path
        description: ID of pet to return
        required: true
    ...
  }
```

Parámetros en la consulta

- ▶ Ejemplo: atributo **in igual a query**.

```
/pet/findByStatus:
```

```
  get:
```

```
    tags:
```

```
      - pet
```

```
    summary: Finds Pets by status
```

```
    description: Multiple status values can be provided with comma separated strings
```

```
    operationId: findPetsByStatus
```

```
    parameters:
```

```
      - name: status
```

```
        in: query
```

```
        description: Status values that need to be considered for filter
```

Parámetros globales

- También podremos definirlos en la **sección parameters de la raíz de nuestro documento**, haciéndolos de esta forma **globales** al mismo y, por tanto, **reutilizables**.
- Algunos ejemplos de parámetros que son el **candidato ideal** para ser definidos de forma global, son aquellos para realizar **paginación u ordenación**, ya que irán **incluidos** en varias de nuestras **peticiones**.
- Al establecer nuestros parámetros como **globales**, podremos reutilizarlos en varias operaciones, referenciándolos mediante **\$ref**. Esto nos **ahorrará mucho trabajo**, evitando que tengamos que **repetir** la definición de los mismos.

parameters:**sortParam:**

name: sort

in: query

description: "Realizar ordenación"

example: "+fecha -nombre"

schema:

type: string

limitParam:

name: limit

in: query

description: "número de resultados"

example: 50

schema:

type: integer

...

Parámetros globales

- Para hacer referencia a un parámetro global tendría que hacer los siguiente:

```
paths:  
/users:  
get:  
  summary: Gets a list of users.  
  parameters:  
    - $ref: '#/components/parameters/offsetParam'  
    - $ref: '#/components/parameters/limitParam'  
responses:  
'200':  
  description: OK
```

Más información

- ▶ <https://swagger.io/specification/>
- ▶ En la sección **Parameter Object Examples**.

Parameter Object Examples

A header parameter with an array of 64 bit integer numbers:

```
1. {
2.   "name": "token",
3.   "in": "header",
4.   "description": "token to be passed as a header",
5.   "required": true,
6.   "schema": {
7.     "type": "array",
8.     "items": {
9.       "type": "integer",
10.      "format": "int64"
```

Más información

- ▶ <https://swagger.io/docs/specification/describing-parameters/>

Describing Parameters

In OpenAPI 3.0, parameters are defined in the `parameters` section of an operation or path. To describe a parameter, you specify its `name`, location (`in`), data type (defined by either `schema` or `content`) and other attributes, such as `description` or `required`. Here is an example:

```
1. paths:
2.   /users/{userId}:
3.     get:
4.       summary: Get a user by ID
5.       parameters:
6.         - in: path
7.           name: userId
8.           schema:
9.             type: integer
10.            required: true
11.            description: Numeric ID of the user to get
```

Paths y operaciones

- ▶ Introducción.
- ▶ Definición de path.
- ▶ Parámetros en el path.
- ▶ Método HTTP.

Introducción

- ▶ En **OpenApi** los **paths** son los **endpoints**, como puede ser:
 - ▶ **/users**
 - ▶ **/users/:userId**
- ▶ Las **operaciones**, los diferentes tipos de métodos http que les aplicamos (GET, POST, PUT, DELETE, etc..).

Introducción

- ▶ A continuación veamos un ejemplo completo de la definición de varios path y operaciones:

```
paths:  
'/users':  
  get:  
    security:  
      - bearerAuth: []  
    tags:  
      - user  
    summary: 'Lista los usuarios del sistema buscando por nombre'  
    parameters:  
      - $ref: "#/components/parameters	sortParam"  
      - name: nombre  
        in: query  
        description: 'Nombre a buscar'
```

Definición de path

- ▶ Se hace de la siguiente manera:

```
paths:
```

```
'/users':
```

```
...
```

```
/invoces:
```

```
...
```

- ▶ Nota: entre comillas o no, es lo mismo.

Parámetros en el path

- ▶ Se pueden definir **parámetros** en la ruta. Esta es una operación muy habitual y nos permite hacer cosas como:
 - ▶ /facturas/003-20021
- ▶ El **parámetro** en cuestión toma el valor **003-20021**.
- ▶ El **path** se define con la siguiente sintaxis:

```
/path/{nombreParametro}:
```

```
...
```

- ▶ El parámetro tiene que estar definido en la sección de parámetros global o específica.

Parámetros en el path

- ▶ Ejemplo:

```
paths:  
...  
/pet/{petId}:  
  get:  
    ...  
    parameters:  
      - name: petId  
        in: path  
        description: ID of pet to return  
    ...
```

Métodos HTTP

- ▶ Se definen dentro del path, por ejemplo:

```
paths:  
...  
/pet/{petId}:  
  get:  
    ...  
  parameters:  
    - name: petId  
      in: path  
      description: ID of pet to return  
    ...
```

Métodos HTTP

- ▶ Sus **atributos más habituales** son:
 - ▶ **security**: En este apartado referenciamos un **security scheme** (que posteriormente definiremos en el apartado de autorización y autenticación).
 - ▶ **tags**: Las **etiquetas** sirven para agrupar nuestras operaciones. Usaremos la tag 'user' y así todas las operaciones se visualizarán agrupadas en nuestra interfaz final.
 - ▶ **parameters**: Array de parameters específicos de esta operación. De igual modo también podemos referenciar a aquellos parámetros definidos como globales haciendo uso de **\$ref**.
 - ▶ **responses**: Es necesario que definamos una respuesta por cada operación. Las respuestas se caracterizan por su **http status code**, el dato returned y su **content type**.

/pet/{petId}:

get:

tags:

- pet

summary: Find pet by ID

description: Returns a single pet

operationId: getPetById

parameters:

- name: petId

in: path

description: ID of pet to return

required: true

schema:

type: integer

format: int64

...

Respuestas

- ▶ Las respuestas se definen a nivel de **operación** y nos indican:
 - ▶ El **código de estado http**,
 - ▶ El **dato returned** y
 - ▶ El **tipo** de contenido del mismo.

Respuestas

- ▶ Ejemplo:

```
/pet/{petId}:
```

```
get:
```

```
...
```

```
responses:
```

```
'200':
```

```
  description: successful operation
```

```
  content:
```

```
    application/json:
```

```
      schema:
```

```
        $ref: '#/components/schemas/Pet'
```

```
    application/xml:
```

```
      schema:
```

```
        $ref: '#/components/schemas/Pet'
```

```
...
```

Respuestas

- ▶ Observemos sus propiedades:
 - ▶ **código de estado:** Definido a través de un número (200, 404, 500, etc).
 - ▶ **description:** Información descriptiva adicional.
 - ▶ **content:** Tipo de contenido, por ejemplo: application/json , application/xml, application/x-www-form-urlencoded, multipart/form-data, text/plain; charset=utf-8, text/html, etc.
 - ▶ **schema:** Aquí haremos referencia a los modelos definidos anteriormente. También podremos definirlos en la propia definición de la response, pero al ser modelos que se reutilizarán en varias operaciones, lo correcto es **definirlos de forma global**.

Lectura recomendada

- ▶ <https://swagger.io/docs/specification/describing-responses/>

SwaggerHub

Swagger Inspector

Open Source Tools

OpenAPI Guide

What Is OpenAPI?

Basic Structure

API Server and Base Path

Media Types

Paths and Operations

Describing Parameters

Parameter Serialization

Describing Request Body

| Describing Responses

Data Models (Schemas)

Adding Examples

...

OAS 3 This guide is for OpenAPI 3.0. If you use OpenAPI 2.0, see our [OpenAPI 2.0 guide](#).

Describing Responses

An API specification needs to specify the `responses` for all API operations. Each operation must have at least one response defined, usually a successful response. A response is defined by its HTTP status code and the data returned in the response body and/or headers. Here is a minimal example:

```
1. paths:
2.   /ping:
3.     get:
4.       responses:
5.         '200':
6.           description: OK
7.           content:
8.             text/plain:
```

Cuerpo de la petición

- ▶ Introducción.
- ▶ La entrada requestBody.
- ▶ requestBody, content y Media Types.
- ▶ anyOf y oneOf.
- ▶ Upload de ficheros.
- ▶ Lectura recomendada.

Introducción

- Al igual que se definen las **respuestas**, también tendremos que **definir la estructura de entrada** de datos para aquellas operaciones que lo permitan, como las que usen **PUT Y POST**.

...

requestBody:

required: true

content:

application/json:

schema:

\$ref: '#/components/schemas/User'

Introducción

- ▶ Podemos observar las **siguientes propiedades**:
 - ▶ **required**: Indicando la obligatoriedad del dato.
 - ▶ **content**: Tipo de contenido, por ejemplo: application/json , application/xml, application/x-www-form-urlencoded, multipart/form-data, text/plain; charset=utf-8 ,text/html, etc.
 - ▶ **schema**: Aquí al igual que en las respuestas, referenciaremos a nuestros modelos definidos anteriormente.

requestBody, content y Media Types

```
paths:  
/pets:  
  post:  
    summary: Add a new pet  
    requestBody:  
      description: Optional description in *Markdown*  
      required: true  
    content:  
      application/json:  
        schema:  
          $ref: '#/components/schemas/Pet'  
      application/xml:  
        schema:  
          $ref: '#/components/schemas/Pet'
```

anyOf y oneOf

- ▶ Se pueden definir esquemas alternativos:

```
requestBody:  
  description: A JSON object containing pet information  
  content:  
    application/json:  
      schema:  
        oneOf:  
          - $ref: '#/components/schemas/Cat'  
          - $ref: '#/components/schemas/Dog'  
          - $ref: '#/components/schemas/Hamster'
```

Upload de ficheros

- ▶ <https://swagger.io/docs/specification/describing-request-body/file-upload/>

SwaggerHub

Swagger Inspector

Open Source Tools

OpenAPI Guide

What Is OpenAPI?

Basic Structure

API Server and Base Path

Media Types

Paths and Operations

Describing Parameters

Parameter Serialization

Describing Request Body

| File Upload

Multipart Requests

Describing Responses

Data Models (Schemas)

OAS 3 This guide is for OpenAPI 3.0. If you use OpenAPI 2.0, see our [OpenAPI 2.0 guide](#).

File Upload

In OpenAPI 3.0, you can describe files uploaded directly with the request content and files uploaded with `multipart` requests. Use the `requestBody` keyword to describe the request payload containing a file. Under `content`, specify the request media type (such as `image/png` or `application/octet-stream`). Files use a `type: string` schema with `format: binary` or `format: base64`, depending on how the file contents will be encoded. For example:

```
1. requestBody:  
2.   content:  
3.     image/png:  
4.       schema:  
5.         type: string  
6.         format: binary
```

Lectura recomendada

- ▶ <https://swagger.io/docs/specification/describing-request-body/>

The screenshot shows the Swagger documentation website. At the top, there's a navigation bar with links for "Why Swagger?", "Tools", and "Resources". On the right side of the header are "Sign In" and "Try Free" buttons. Below the header, on the left, is a sidebar with links to various OpenAPI resources like "SwaggerHub", "Swagger Inspector", "Open Source Tools", "OpenAPI Guide", and several detailed sections. The main content area has a large title "Describing Request Body" followed by a detailed paragraph about request bodies. Below that is a section titled "Differences From OpenAPI 2.0" with a summary and a bulleted list of changes.

Swagger
Supported by SMARTBEAR

Why Swagger? ▾ Tools ▾ Resources ▾

Sign In Try Free

SwaggerHub

Swagger Inspector

Open Source Tools

OpenAPI Guide

What Is OpenAPI?

Basic Structure

API Server and Base Path

Media Types

Paths and Operations

Describing Parameters

Parameter Serialization

Describing Request Body

Request bodies are typically used with “create” and “update” operations (POST, PUT, PATCH). For example, when creating a resource using POST or PUT, the request body usually contains the representation of the resource to be created. OpenAPI 3.0 provides the `requestBody` keyword to describe request bodies.

Differences From OpenAPI 2.0

If you used OpenAPI 2.0 before, here is a summary of changes to help you get started with OpenAPI 3.0:

- Body and form parameters are replaced with `requestBody`.
- Operations can now consume both form data and other media types such as JSON.

Autenticación y autorización

- ▶ **OpenAPI** usa el término **security schema** para definir la autenticación y autorización. Los diferentes security schemes serán definidos en el objeto **components.securitySchemes**:

```
securitySchemes:
```

```
basicAuth:
```

```
  type: http
```

```
  scheme: basic
```

```
bearerAuth:
```

```
  type: http
```

```
  scheme: bearer
```

```
  bearerFormat: JWT
```

Autenticación y autorización

- ▶ Podemos observar las siguientes propiedades:
 - ▶ **type**: Puede tomar diferentes valores en función del tipo de autenticación (http, apiKey, oauth2, openidConnect).
 - ▶ **scheme**: Dentro de cada tipo de autenticación podemos elegir entre diferentes subtipos, por ejemplo, bearer y basic dentro de la autenticación http.
 - ▶ **bearerFormat**: Propiedad arbitraria que indica el tipo de token, en este caso JWT (JSON Web Token).
- ▶ Una vez que hemos definidos nuestros **esquemas de seguridad**, podremos aplicarlos en **todas las operaciones** o sólo en determinados paths y operaciones, usando para esto la **propiedad security** de nuestro documento raíz o de nuestra operación.

Autenticación y autorización

- ▶ Ejemplo:

```
openapi: 3.0.0
info:
  description: Especificación de API para proyecto de ejemplo.
  version: 1.0.0
  title: API proyecto de ejemplo
servers:
  - url: 'https://antoniofernandez.com'
security:
  - bearerAuth []
```

Swagger Open Source

- ▶ Introducción.
- ▶ Swagger Codegen.
- ▶ Swagger Editor.
- ▶ Swagger UI.

Introducción

- ▶ **Swagger nos ofrece un conjunto de herramientas Open Source:**

The screenshot shows the official Swagger website. At the top, there's a navigation bar with the logo, a search icon, a 'Sign In' button, and a 'Try Free' button. Below the header, there's a main banner with the text 'API Development for Everyone' and a 'NEW API Exploration Made Easy' badge. To the right of the banner, there are two columns of tools:

- Pro**:
 - SwaggerHub**: Design & document all your REST APIs in one collaborative platform.
 - SwaggerHub Enterprise**: Standardize your APIs with projects, style checks, and reusable domains.
 - SwaggerHub Explore**: Instantly evaluate the functionality of any API.
- Open Source**:
 - Swagger Codegen**: Generate server stubs and client SDKs from OpenAPI Specification definitions.
 - Swagger Editor**: API editor for designing APIs with the OpenAPI Specification.
 - Swagger UI**: Visualize OpenAPI Specification definitions in an interactive UI.

A large green button at the bottom right of the left column says 'Explore all tools >'. To the right of the tools, there's a large graphic of a 3D cube network made of smaller cubes, with one central cube highlighted in green.

Introducción

- ▶ Las herramientas que pasamos a comentar son:
 - ▶ **Swagger Codegen**: puede simplificar el proceso de build de un proyecto al generar stubs de servidor y SDK de cliente para cualquier API definida con la **especificación OpenAPI**. De esta forma su equipo pueda concentrarse en la **implementación de su API**.
 - ▶ **Swagger Editor**: **Diseñe, describa y documente su API** en el primer editor de código abierto compatible con múltiples especificaciones de API y formatos de serialización. El editor de Swagger ofrece una **manera fácil** de comenzar con la **especificación OpenAPI**.
 - ▶ **Swagger UI**: esta herramienta permite que **cualquier persona**, ya sea de su equipo de desarrollo o sus **consumidores finales**, **visualice e interactúe con los recursos de la API** sin tener implementada ninguna lógica. Se genera automáticamente a partir de su especificación OpenAPI, con la documentación visual que facilita la implementación de back-end y el consumo del lado del cliente.

Swagger Codegen

- ▶ Introducción.
- ▶ Descargar una versión local.
- ▶ Instalación.
- ▶ Usando la herramienta.
- ▶ Generadores en línea.

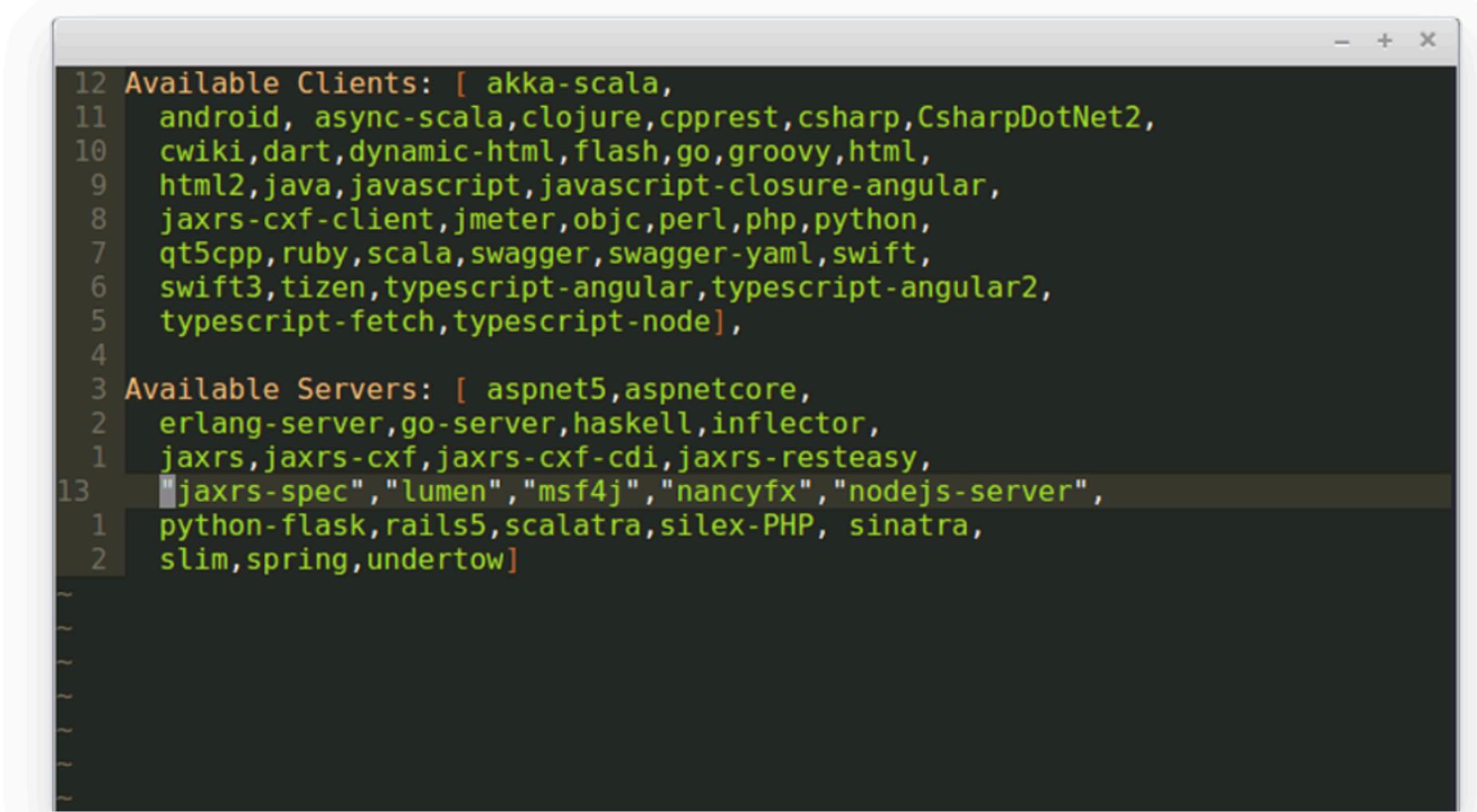
Introducción

- ▶ <https://swagger.io/tools/swagger-codegen/>

Swagger Codegen

Swagger Codegen can simplify your build process by generating server stubs and client SDKs for any API, defined with the OpenAPI (formerly known as Swagger) specification, so your team can focus better on your API's implementation and adoption.

 Download



```
12 Available Clients: [ akka-scala,
11 android, async-scala,clojure,cpprest,csharp,CsharpDotNet2,
10 cwiki,dart,dynamic-html,flash,go,groovy,html,
9 html2,java,javascript,javascript-closure-angular,
8 jaxrs-cxf-client,jmeter,objc,perl,php,python,
7 qt5cpp,ruby,scala,swagger,swagger-yaml,swift,
6 swift3,tizen,typescript-angular,typescript-angular2,
5 typescript-fetch,typescript-node],
4
3 Available Servers: [ aspnet5,aspnetcore,
2 erlang-server,go-server,haskell,inflector,
1 jaxrs,jaxrs-cxf,jaxrs-cxf-cdi,jaxrs-resteasy,
13 jaxrs-spec","lumen","msf4j","nancyfx","nodejs-server",
1 python-flask,rails5,scalatra,silex-PHP, sinatra,
2 slim,spring,undertow]
```

Descargar una versión local

- ▶ En la sección de descarga nos encontramos con una comparativa:
 - ▶ <https://swagger.io/tools/swagger-codegen/download/>

	Codegen	SwaggerHub
	Download 	Sign Up Free 
General Features		
Automated Server Stub Generation	✓	✓
Automated SDK Generation	✓	✓
Advanced Editing		
Codeless Mocking		✓
Style Guide Enforcement		✓

Descargar una versión local

- ▶ Si decidimos usar la versión local debemos acceder al siguiente enlace:
 - ▶ <https://github.com/swagger-api/swagger-codegen>

The screenshot shows the GitHub repository page for `swagger-api / swagger-codegen`. The repository is public, has 420 watchers, 5.9k forks, and 15.4k stars. The main navigation tabs include Code (which is selected), Issues (2.9k), Pull requests (383), Discussions, Actions, Projects, Wiki, and a more options menu. A dropdown menu for the master branch is open. Below the navigation is a timeline of recent activity:

- HugoMario Merge pull request #12070 from swagger-api/o... - last week (11,560 reviews)
- .github added gh actions for okhttp4-gson - last week

To the right, there is an "About" section with the following text:

swagger-codegen contains a template-driven engine to generate documentation, API clients and server stubs in different

Instalación

- ▶ <https://github.com/swagger-api/swagger-codegen#prerequisites>

Prerequisites

If you're looking for the latest stable version, you can grab it directly from Maven.org (Java 8 runtime at a minimum):

```
# Download current stable 2.x.x branch (Swagger and OpenAPI version 2)
wget https://repo1.maven.org/maven2/io/swagger/swagger-codegen-cli/2.4.30/swagger-co
java -jar swagger-codegen-cli.jar help

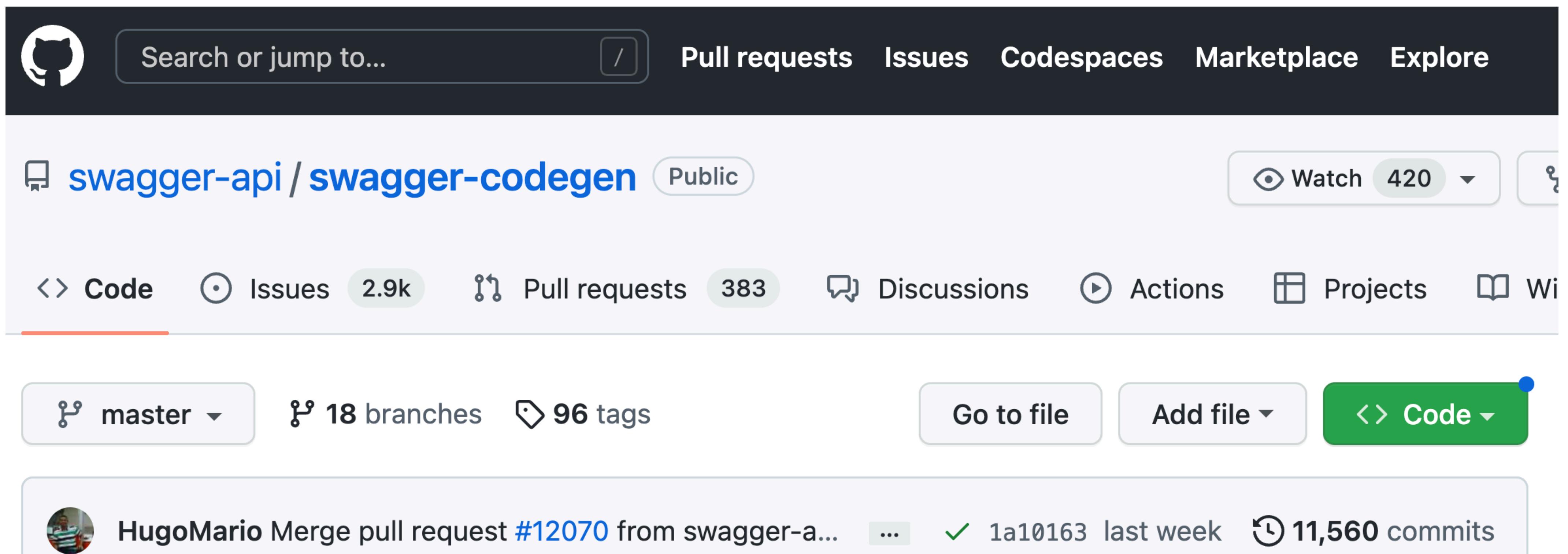
# Download current stable 3.x.x branch (OpenAPI version 3)
wget https://repo1.maven.org/maven2/io/swagger/codegen/v3/swagger-codegen-cli/3.0.41
java -jar swagger-codegen-cli.jar --help
```

Usando la herramienta

- ▶ Introducción.
- ▶ Generando un cliente.
- ▶ Generando stub de servidor.

Introducción

- ▶ Dada una especificación OpenAPI, genera **automáticamente** bibliotecas de cliente para una API (SDK), stubs de servidor y documentación.
- ▶ <https://github.com/swagger-api/swagger-codegen>



Generando un cliente

- ▶ <https://github.com/swagger-api/swagger-codegen#getting-started>

Getting Started

To generate a PHP client for <https://petstore.swagger.io/v2/swagger.json>, please run the following

```
git clone https://github.com/swagger-api/swagger-codegen
cd swagger-codegen
mvn clean package
java -jar modules/swagger-codegen-cli/target/swagger-codegen-cli.jar generate \
-i https://petstore.swagger.io/v2/swagger.json \
-l php \
-o /var/tmp/php_api_client
```

Generando un cliente

- ▶ Puede generar el cliente desde un fichero local:
 - ▶ <https://github.com/swagger-api/swagger-codegen#generating-a-client-from-local-files>

Generating a client from local files

If you don't want to call your server, you can save the OpenAPI Spec files into a directory and pass an argument to the code generator like this:

```
-i ./modules/swagger-codegen/src/test/resources/2_0/petstore.json
```

Great for creating libraries on your ci server, from the [Swagger Editor](#)... or while coding on an airplane.

Generando stub de servidor

- ▶ <https://github.com/swagger-api/swagger-codegen/wiki/Server-stub-generator-HOWTO>

Server stub generator HOWTO

Romain LOUVET edited this page on Nov 9, 2020 · 44 revisions

Here is the documentation to generate a server stub for a couple different frameworks. If you want to contribute to make it better, please click on the "Edit" button to update the documentation.

- [ASP.NET Core 1.0](#)
- [Erlang](#)
- [Go Server](#)
- [Haskell Servant](#)
- [Java JAX-RS \(Apache CXF 2 / 3\)](#)
- [Java JAX-RS \(Apache CXF framework on Java EE server supporting CDI\)](#)
- [Java JAX-RS \(Java JAX-RS \(Jersey\)\)](#)
- [Java JAX-RS \(Resteasy\)](#)

▶ Pages 18

Navigation

Codegen Usage

- [Workflow Integration](#)

Clone this wiki locally

<https://github.com/swagger-api/>



Generando stub de servidor

- ▶ Por ejemplo, trabajando con Spring Boot:
 - ▶ <https://github.com/swagger-api/swagger-codegen/wiki/Server-stub-generator-HOWTO#java-springboot>

Java SpringBoot

spring generator uses `SpringBoot` as the default library

```
git clone https://github.com/swagger-api/swagger-codegen
cd swagger-codegen
java -jar modules/swagger-codegen-cli/target/swagger-codegen-cli.jar generate \
-i https://petstore.swagger.io/v2/swagger.json \
-l spring \
-o samples/server/petstore/springboot
```

Generadores en línea

- ▶ También se puede generar un cliente o servidor API utilizando **los generadores en línea**:
 - ▶ <https://github.com/swagger-api/swagger-codegen#online-generators>

Online generators

One can also generate API client or server using the online generators
(<https://generator.swagger.io>)

For example, to generate Ruby API client, simply send the following HTTP request using curl:

```
curl -X POST -H "content-type:application/json" -d '{"swaggerUrl":"https://petstore.
```

Then you will receive a JSON response with the URL to download the zipped code.

Swagger Generator 2.4.30

[Base URL: generator.swagger.io/api]

<https://generator.swagger.io/api/swagger.json>

This is an online swagger codegen server. You can find out more at <https://github.com/swagger-api/swagger-codegen> or on [irc.freenode.net, #swagger](#).

[Terms of service](#)

[Apache 2.0](#)

Schemes

HTTPS ▾

gen ^

clients ^

GET /gen/download/{fileId} Downloads a pre-generated file ▾

GET /gen/clients/{language} Returns options for a client library ▾

POST /gen/clients/{language} Generates a client library ▾

GET /gen/clients Gets languages supported by the client generator ▾

Swagger Editor

- ▶ Introducción.
- ▶ Usar editor en línea.
- ▶ Descargar una versión local.
- ▶ Generando cliente y servidor.

Introducción

- ▶ <https://swagger.io/tools/swagger-editor/>

Swagger Editor

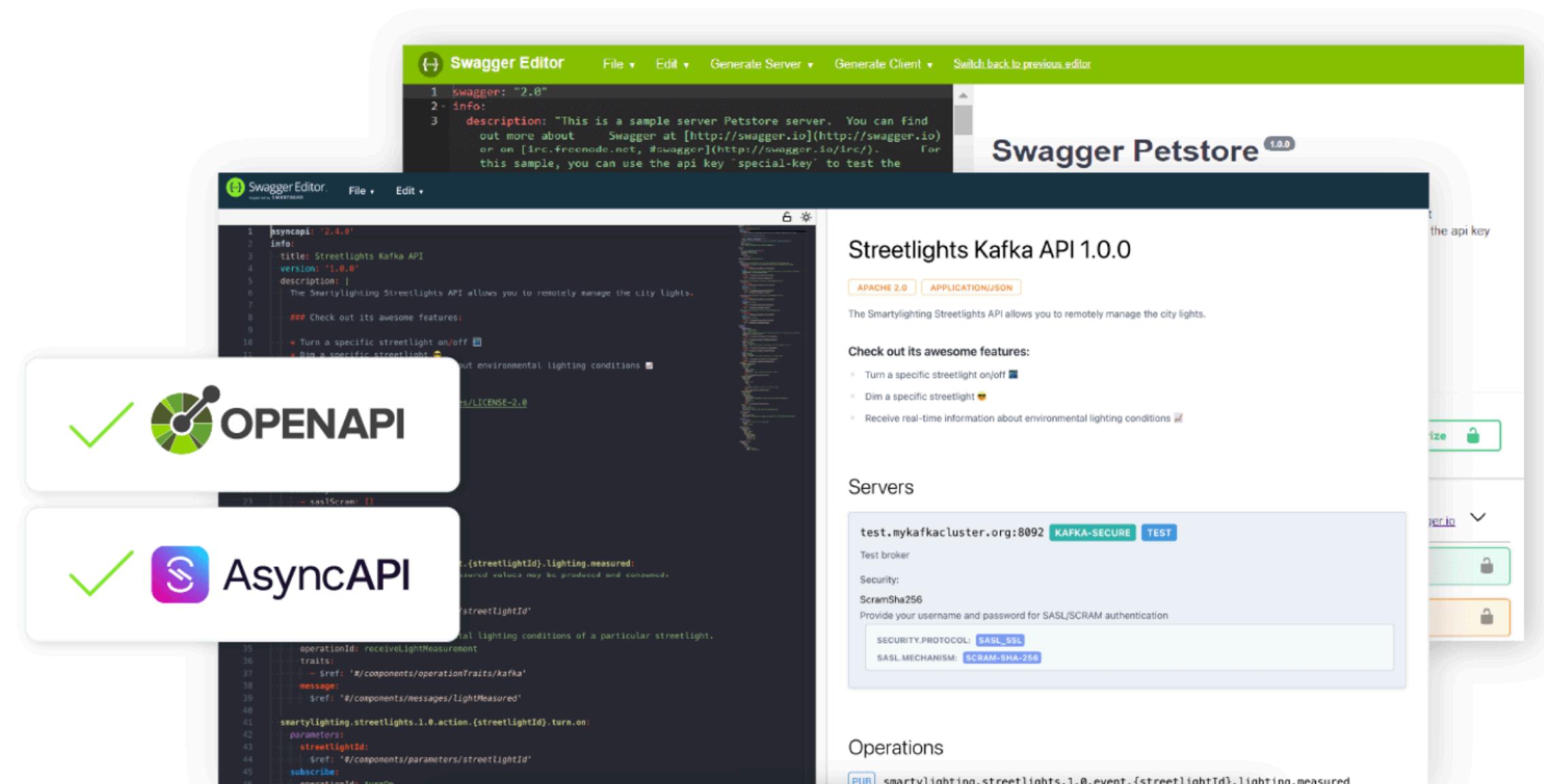
Design, describe, and document your API on the first open source editor supporting multiple API specifications and serialization formats. The Swagger Editor offers an easy way to get started with the OpenAPI Specification (formerly known as Swagger) as well as the AsyncAPI specification, with support for Swagger 2.0, OpenAPI 3.0, and AsyncAPI 2.* versions.

Try Swagger Editor ↗

Try Swagger Editor Next (beta) ↗

 [Download Swagger Editor](#)

Compare Editor Versions >

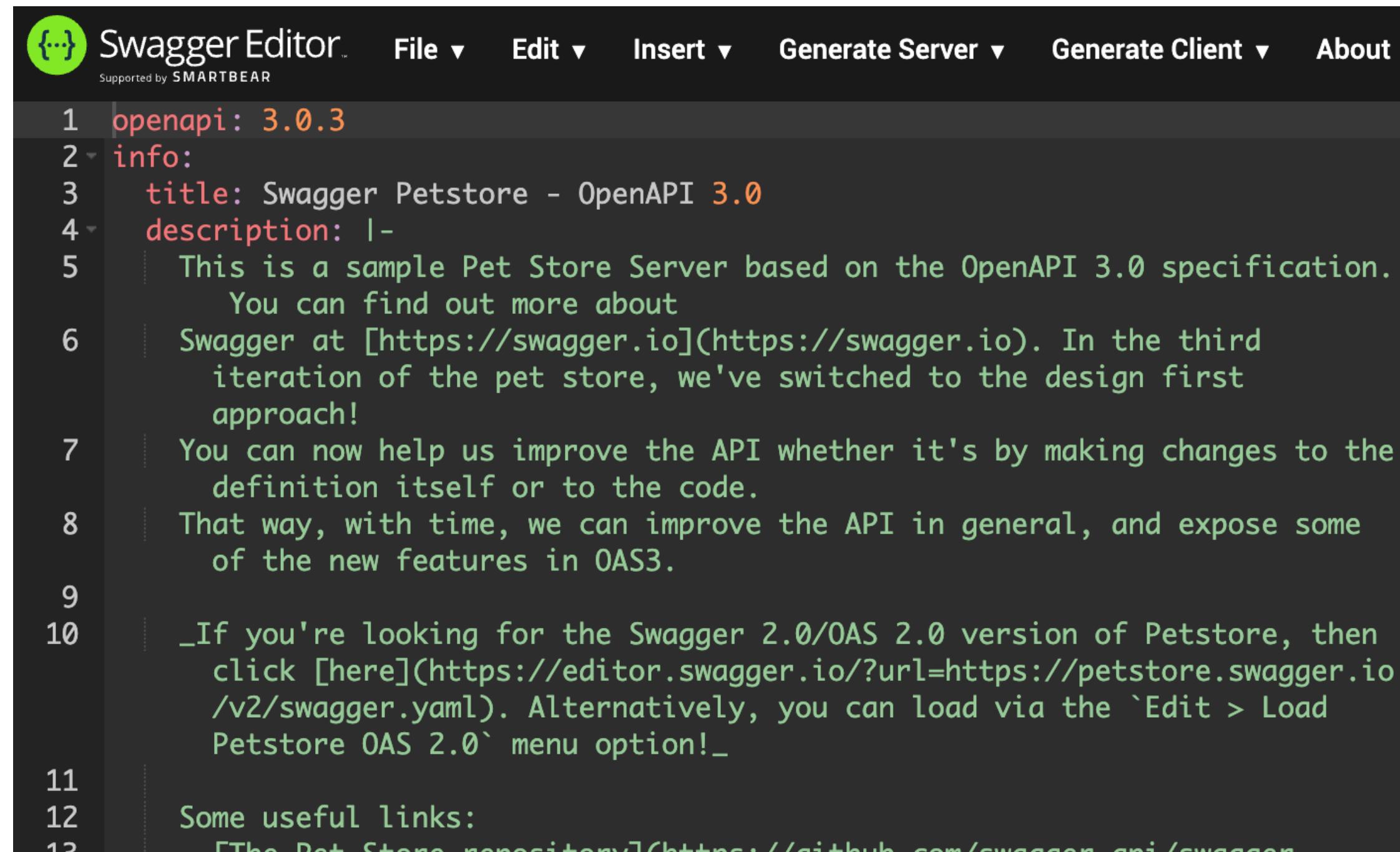


Introducción

- ▶ Tenemos tres opciones para trabajar con esta herramienta:
 - ▶ Usar el editor en línea.
 - ▶ Descargar una versión local.
 - ▶ Probar su versión en la nube.

Usar editor en línea

- ▶ <https://editor.swagger.io/>



The screenshot shows the Swagger Editor interface. At the top, there's a navigation bar with links for File, Edit, Insert, Generate Server, Generate Client, and About. A green button on the right says "Try our new Editor". The main area displays the OpenAPI 3.0 specification for the Petstore API. The code is as follows:

```

1 openapi: 3.0.3
2 info:
3   title: Swagger Petstore - OpenAPI 3.0
4   description: |
5     This is a sample Pet Store Server based on the OpenAPI 3.0 specification.
6     You can find out more about
7     Swagger at [https://swagger.io](https://swagger.io). In the third
8     iteration of the pet store, we've switched to the design first
9     approach!
10    You can now help us improve the API whether it's by making changes to the
11    definition itself or to the code.
12    That way, with time, we can improve the API in general, and expose some
13    of the new features in OAS3.
14
15    If you're looking for the Swagger 2.0/OAS 2.0 version of Petstore, then click here. Alternatively, you can load
16    via the Edit > Load Petstore OAS 2.0 menu option!
17
18  Some useful links:
19    [The Pet Store repository](https://github.com/swagger-api/swagger)

```

On the right side of the editor, there's a preview area titled "Swagger Petstore - OpenAPI 3.0" with version 1.0.11 and OAS3. It contains a summary of the API and a note about the transition to the design first approach. Below the summary, there's a message about the Swagger 2.0/OAS 2.0 version and a "Edit > Load Petstore OAS 2.0" menu option. There are also links to the Pet Store repository and the source API definition.

Descargar una versión local

- ▶ En la sección de descarga nos encontramos con una comparativa:
 - ▶ <https://swagger.io/tools/swagger-editor/download/>
- ▶ Si queremos descargar el editor en local:

	Swagger Editor	Swagger Editor Next (beta)	SwaggerHub
	Download	Download	Sign Up Free
General Features			
YAML & JSON editor with semantic validation & highlighting	✓	✓	✓
OpenAPI 2.0 specification support	✓		✓
OpenAPI 3.0 specification support	✓	✓	✓
OpenAPI 3.1 specification support		partial	
AsyncAPI specification support		✓	✓
Real-Time Validation	✓	✓	✓

Descargar una versión local

- ▶ Esto nos lleva a la siguiente url:
 - ▶ <https://github.com/swagger-api/swagger-editor#docker>

Docker

Running the image from DockerHub

There is a docker image published in [DockerHub](#).

To use this, run the following:

```
docker pull swaggerapi/swagger-editor
docker run -d -p 80:8080 swaggerapi/swagger-editor
```

Generando cliente y servidor

- Una vez creada la definición del servicio REST podemos descargar **clientes y servidores** para diferentes lenguajes:

The screenshot shows the Swagger Editor interface with the following details:

- Header:** Swagger Editor, Supported by SMARTBEAR.
- Menu Bar:** File ▾, Edit ▾, Insert ▾, Generate Server ▾, Generate Client ▾, About ▾.
- Left Panel (Code View):** Displays the OpenAPI 3.0 specification for the Petstore API, including sections for info, components, paths, and security schemes.
- Right Panel (Client Generation Options):** A dropdown menu titled "Generate Client" is open, listing various client generation options grouped by server technology:
 - aspnetcore**:
 - go-server
 - inflector
 - jaxrs-cxf
 - jaxrs-cxf-cdi
 - jaxrs-di
 - jaxrs-jersey
 - jaxrs-resteasy
 - jaxrs-resteasy-eap
 - jaxrs-spec
 - kotlin-server**:
 - micronaut
 - nodejs-server
 - python-flask
 - scala-akka-http-server
 - spring
- Bottom Right Content:** A snippet of the generated code for the Petstore API, starting with "store - Op".

Swagger UI

- ▶ Introducción.
- ▶ Usar una demo en línea.
- ▶ Descargar una versión local.

Introducción

- ▶ <https://swagger.io/tools/swagger-ui/>

The screenshot shows the Swagger Open Source homepage. At the top, there's a navigation bar with the Swagger logo (a green circle with three dots), the text "Swagger Supported by SMARTBEAR", and dropdown menus for "Why Swagger?", "Tools", and "Resources". To the right are a search icon and a "Sign In" button. Below the navigation is a main menu with tabs: "Swagger Open Source" (which is active and highlighted in blue), "Editor", "Codegen", and "UI". On the far right of this menu is a green "Try SwaggerHub" button. The background features a large, faint image of a person working at a computer.

Swagger UI

Swagger UI allows anyone — be it your development team or your end consumers — to visualize and interact with the API's resources without having any of the implementation logic in place. It's automatically generated from your OpenAPI (formerly known as Swagger) Specification, with the visual documentation making it easy for back end implementation and client side consumption.

The screenshot shows the Swagger UI interface. At the top, there's a dropdown for "Schemes" set to "HTTP" and an "Authorize" button. The main area displays the "pet" resource with various operations: POST /pet (Add a new pet to the store), PUT /pet (Update an existing pet), GET /pet/findByStatus (Finds Pets by status), GET /pet/findByTags (Finds Pets by tags), GET /pet/{petId} (Find pet by ID), POST /pet/{petId} (Updates a pet in the store with form data), DELETE /pet/{petId} (Deletes a pet), and POST /pet/{petId}/uploadImage (uploads an image). Below the "pet" resource is a collapsed section for the "store" resource. At the bottom of the UI, there are links for "Live Demo" (with an upward arrow icon), "Download Swagger UI" (with a download icon), and "Try it in the cloud" (with a cloud icon).

Introducción

- ▶ Tenemos tres opciones para trabajar con esta herramienta:
 - ▶ Usar una demo en línea.
 - ▶ Descargar una versión local.
 - ▶ Probar su versión en la nube.

Usar una demo en línea

- ▶ <https://petstore.swagger.io/>

The screenshot shows the Swagger Petstore API documentation. At the top, there's a navigation bar with the Swagger logo, the URL <https://petstore.swagger.io/v2/swagger.json>, and a green "Explore" button. Below the header, the title "Swagger Petstore" is displayed with a version "1.0.6" badge. A note indicates the base URL is petstore.swagger.io/v2. The main content area contains a brief introduction, links to "Terms of service", "Contact the developer", "Apache 2.0", and "Find out more about Swagger". At the bottom, there's a "Schemes" dropdown set to "HTTPS" and an "Authorize" button with a lock icon. The main content area shows the "pet" resource, which includes two POST methods: "/pet/{petId}/uploadImage" and "/pet". Each method has a "Find out more" link and a collapse/expand arrow.

Swagger Petstore 1.0.6

[Base URL: petstore.swagger.io/v2]
<https://petstore.swagger.io/v2/swagger.json>

This is a sample server Petstore server. You can find out more about Swagger at <http://swagger.io> or on [irc.freenode.net, #swagger](#). For this sample, you can use the api key `special-key` to test the authorization filters.

[Terms of service](#)
[Contact the developer](#)
[Apache 2.0](#)
[Find out more about Swagger](#)

Schemes

HTTPS ▾

Authorize

pet Everything about your Pets [Find out more](#) ^

POST /pet/{petId}/uploadImage uploads an image

POST /pet Add a new pet to the store

Descargar una versión local

- ▶ <https://swagger.io/tools/swagger-ui/download/>

	Swagger UI	SwaggerHub
	Download 	Sign Up Free 
General Features		
Swagger 2.0, OpenAPI 3.0 specification support	✓	✓
Auto-Generated Interactive API Documentation	✓	✓
Advanced Editing		
YAML editor with basic style validation		✓
Real-Time Validation		✓
Smart Auto-Completion		✓
Codeless Mocking		✓
Style Guide Enforcement		✓
Reusable Components		✓

Descargar una versión local

- ▶ Si decidimos usar la versión local debemos acceder al siguiente enlace:
 - ▶ <https://github.com/swagger-api/swagger-ui>

The screenshot shows the GitHub repository page for `swagger-api/swagger-ui`. The repository is public, has 656 watchers, 8.6k forks, and 23.5k stars. The `Code` tab is selected. The commit history shows the following recent changes:

Author	Commit Message	Time Ago
swagger-bot	chore(release): cut the v4.16.1 release	yesterday
	ci(ga): add next branch to build and dependab...	last week
	chore(deps-dev): update husky to 7.0.2 version	2 years ago
	fix(jsdom): cumulative Jest updates with supp...	7 months ago
	fix(oauth2): parse params properly for casdoor...	6 months ago

The `About` section on the right provides a brief description of Swagger UI:

Swagger UI is a collection of HTML, JavaScript, and CSS assets that dynamically generate beautiful documentation from a Swagger-compliant API.

Associated links include [swagger.io](#) and tags `rest`, `rest-api`, and `swagger`.

Descargar una versión local

- Para instalar la herramienta en local debemos seguir las indicaciones en la siguiente url:
 - <https://github.com/swagger-api/swagger-ui/blob/master/docs/usage/installation.md>

Docker

You can pull a pre-built docker image of the swagger-ui directly from Docker Hub:

```
docker pull swaggerapi/swagger-ui
docker run -p 80:8080 swaggerapi/swagger-ui
```

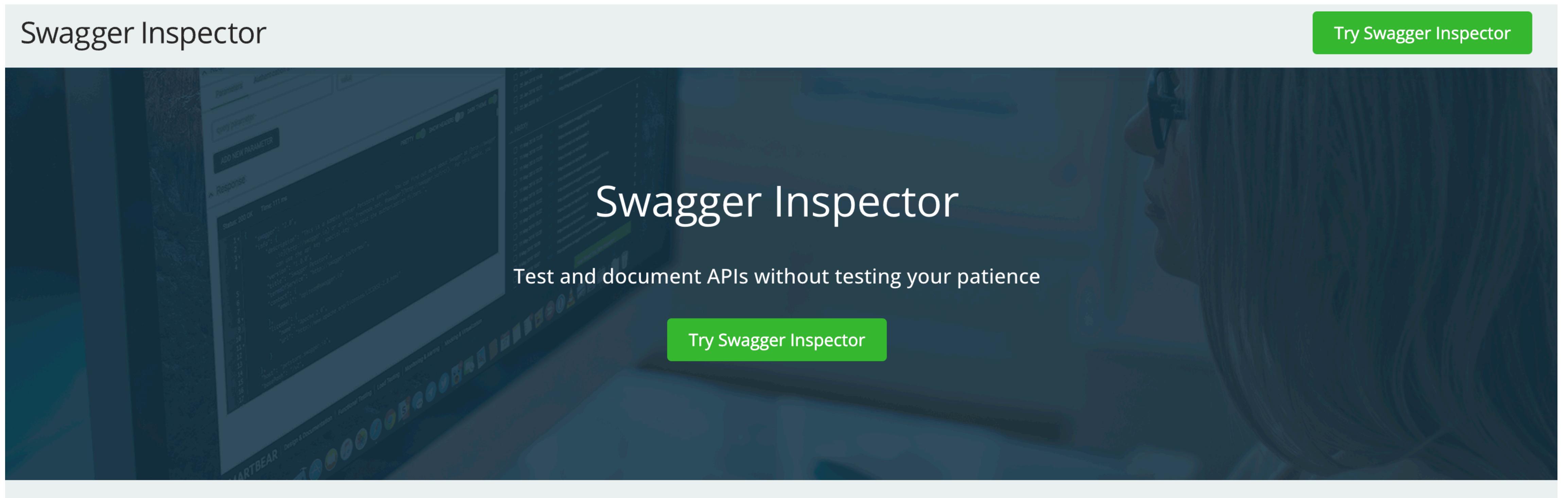
Will start nginx with Swagger UI on port 80.

Or you can provide your own swagger.json on your host

```
docker run -p 80:8080 -e SWAGGER_JSON=/foo/swagger.json -v /bar:/foo swaggerapi/swagger-ui
```

Swagger Inspector

- ▶ Herramienta para **testear y generar definiciones API** en el navegador en cuestión de segundos.
- ▶ <https://swagger.io/tools/swagger-inspector/>
- ▶ Podemos probarla haciendo click en el botón:



Swagger Inspector

The screenshot shows the Swagger Inspector interface. At the top, there's a navigation bar with the SmartBear logo, a search bar containing 'Swagger Inspector', and buttons for '?', 'SIGN UP', and 'LOG IN'.

The main area has tabs for 'DEFINITION' (selected), 'POST' (method), and the URL 'https://petstore.swagger.io/v2/store/order'. A 'SEND' button is also present.

On the left, under 'Request', there are sections for 'Parameters', 'Authentication & Headers', and 'Body' (which is selected). A 'CLEAR REQUEST' button is located at the bottom of this section. A 'DARK THEME' toggle switch is also present.

The 'Body' section contains the following JSON code:

```

1 "id": 0,
2 "petId": 0,
3 "quantity": 0,
4 "shipDate": "2015-07-20T15:49:04-07:00",
5 "status": "placed"

```

Under 'Response', it shows 'Status: 200' and 'Time: 338 ms'. There are toggles for 'PRETTY', 'SHOW HEADERS', and another 'DARK THEME' switch. The 'content-type: application/json' section displays the following JSON response:

```

1 {
2   "id": 8908918220980320000,
3   "petId": 0,
4   "quantity": 0,
5   "shipDate": "2015-07-20T22:49:04.000+0000",
6   "status": "placed",
7   "complete": true
8 }

```

On the right side, there are tabs for 'History' (selected) and 'Collections'. Under 'History', it shows a single entry: '28 Feb 2023 21:55' with a 'POST' method and the same URL. It also indicates 'No pinned items'.

Swagger Inspector

- Desde la interfaz podemos cargar **ficheros de definición** para inspeccionar una API:

The screenshot shows the SMARTBEAR Swagger Inspector interface. At the top, there's a navigation bar with the logo, a search bar for 'DEFINITION' and 'POST https://petstore.swagger.io/v2/store/order', and buttons for 'SIGN UP' and 'LOG IN'. Below the header, a message says 'Supports Swagger/OAS 2, OAS 3 or WSDL file'. A link to 'https://petstore.swagger.io/v2/swagger.json' is shown with a 'PARSED' status. On the left, the 'Swagger Petstore' tree view shows the 'store' endpoint with methods: POST /store/order, GET /store/order/{orderId}, DELETE /store/order/{orderId}, and GET /store/inventory. The main area has a 'CLEAR REQUEST' button and a 'DARK THEME' toggle. On the right, there are tabs for 'History' (selected) and 'Collections', and a 'CREATE API DEFINITION' dropdown. Under 'History', it shows a single entry: a POST request to https://petstore.swagger.io/v2/store/order on 28 Feb 2023 at 21:55.

Swagger Inspector

- Se pueden definir peticiones sobre los recursos que deseemos e inspeccionar la respuesta:

The screenshot shows the SMARTBEAR Swagger Inspector interface. At the top, there are navigation links for 'DEFINITION' (selected), 'SIGN UP', and 'LOG IN'. Below this, a search bar shows 'POST https://petstore.swagger.io/v2/store/order' and a 'SEND' button.

The main area is divided into sections: 'Request' and 'Response'. Under 'Request', tabs for 'Parameters', 'Authentication & Headers', and 'Body' are visible. The 'Body' tab is selected, showing a JSON editor with the following code:

```
1 {  
2   "id": 0,  
3   "petId": 0,  
4   "quantity": 0,  
5   "shipDate": "2015-07-20T15:49:04-07:00",  
6   "status": "placed",  
7   "complete": true  
8 }
```

A 'CLEAR REQUEST' button is located at the bottom of the Request section. To the right, there's a 'History' tab (selected) and a 'Collections' tab. The 'History' tab shows a list of recent requests:

- 28 Feb 2023 22:07 POST https://petstore.swagger.io/v2/store/order
- 28 Feb 2023 21:55 POST https://petstore.swagger.io/v2/store/order

At the bottom of the Request section, there are filters for 'Status: 200' and 'Time: 728 ms', and buttons for 'PRETTY', 'SHOW HEADERS', and 'DARK THEME'.

SwaggerHub

- ▶ Introducción.
- ▶ Precios.
- ▶ Integraciones.
- ▶ Crear una cuenta.
- ▶ Dashboard.

Introducción

- ▶ SwaggerHub es una **plataforma integrada de diseño y documentación de API**, creada para que los **equipos impulsen la coherencia y la disciplina** en todo el flujo de trabajo de desarrollo de API.
- ▶ **Diseño API más rápido y estandarizado:**
 - ▶ Acelere el proceso de diseño de su equipo sin pérdida de calidad o consistencia en el estilo organizacional con un **potente editor** que cumple totalmente con los **últimos estándares** de Swagger (OpenAPI).
 - ▶ Un **potente editor** equipado con **retroalimentación de error inteligente** y **autocompletado** de sintaxis.
 - ▶ **Validadores de estilo** para garantizar la **coherencia del diseño** en varias API.
 - ▶ **API Mocking** para virtualizar operaciones sin ningún código.
 - ▶ **Dominios** para almacenar, reutilizar y hacer referencia a la sintaxis común de OAS en múltiples API.

Precios

- ▶ <https://swagger.io/tools/swaggerhub/pricing/>

Explore SwaggerHub Plans

The screenshot shows the SwaggerHub pricing page with three main plans displayed as cards:

- Free**: €0 per month for 1 Designer. Includes API Editor, Hosted Docs & Mocking. Call-to-action button: Create Free Account.
- Team**: €84 per month for 3 Designers / 6 Consumers. Billed Annually by default. Includes Collaboration, Private APIs, & API Lifecycle Integrations. Call-to-action button: Start Free Trial.
- Enterprise**: Starts at 15 Designers / 30 Consumers. Includes SaaS or On-Premise, Style Guide Enforcement, Priority Support & Enterprise SSO. Call-to-action button: Contact Us.

Integraciones

- ▶ **SwaggerHub** está diseñado para **complementar la pila de herramientas** de su equipo. Con una gran cantidad de integraciones nativas, SwaggerHub ayuda a los diseñadores a mantenerse conectados durante el desarrollo de la API.
- ▶ <https://swagger.io/tools/swaggerhub/integrations/>



Deploy to API Gateways



If you're leveraging SwaggerHub to generate server templates and client SDKs, you can also sync your API definitions with source control repositories in GitHub, Bitbucket, GitLab, Azure DevOps Services, and Azure DevOps Server.

The synchronization is made every time you save the API in SwaggerHub.

You can fully control which files will be added, updated or ignored in the target repository.

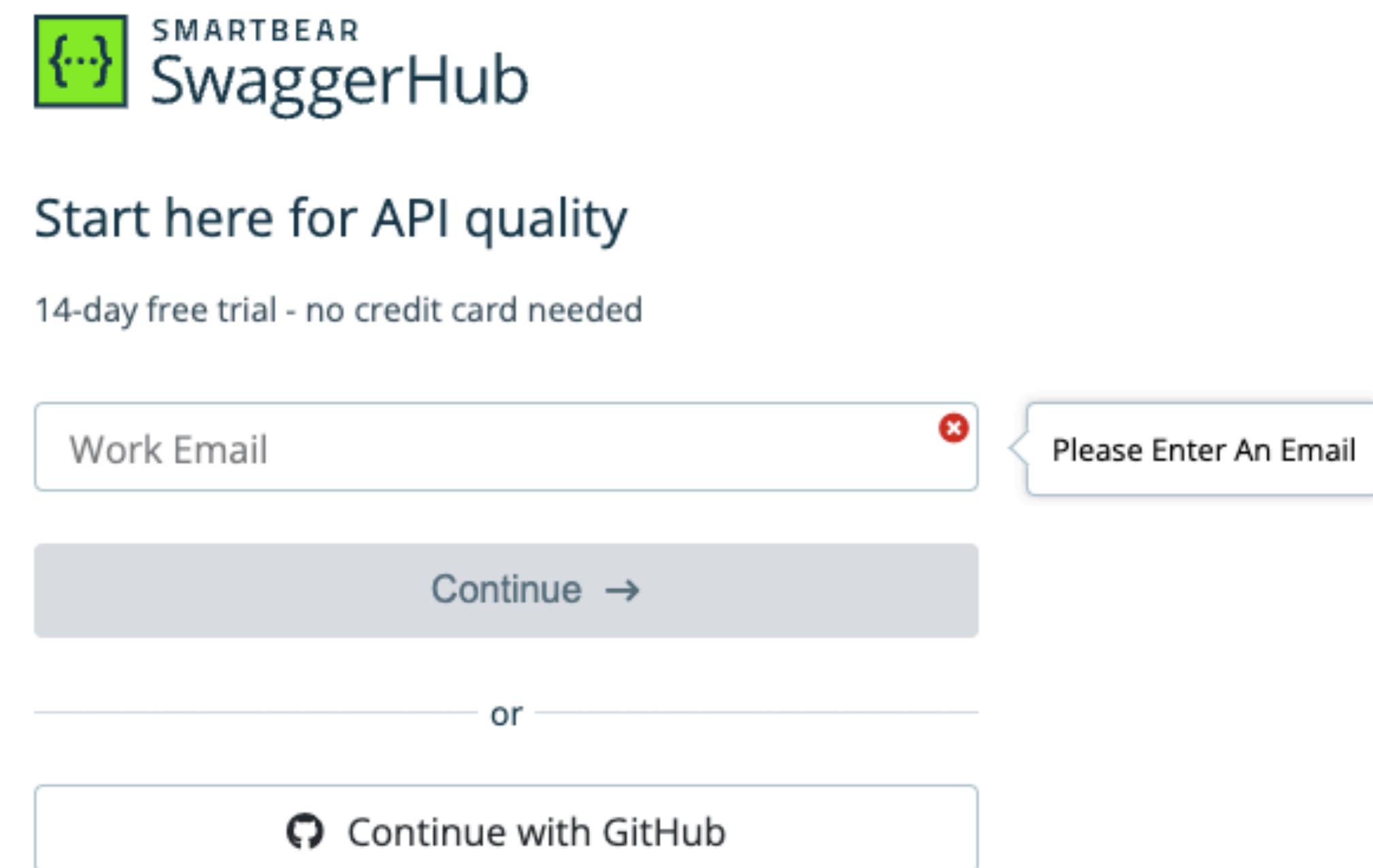
Crear una cuenta

- ▶ <https://swagger.io/tools/swaggerhub/>

The screenshot shows the SwaggerHub website. At the top, there is a navigation bar with links for "Why Swagger?", "Tools", and "Resources". On the right side of the header are a search icon and a "Sign In" button. Below the header, there is a main menu with "SwaggerHub.", "Features", "Pricing", "Enterprise", and "Integrations". To the right of the menu is a green "Create Free Account" button. The main content area features a banner with the text "NEW API Exploration Made Easy" and "The Single Source of Truth". A modal window is open in the bottom right corner, displaying the "SMARTBEAR SwaggerHub" logo and a list of items: "1. SwaggerHub_customer: Adidas" and "2.". To the right of the modal, there is a small image of a person's head.

Crear una cuenta

- ▶ Si hacemos click en **Create Free Account**:



Dashboard

- ▶ Nos proporciona acceso a toda la funcionalidad:

The screenshot shows the SwaggerHub dashboard interface. At the top, there's a header with the SmartBear logo, the text "SwaggerHub™", a message about an "Enterprise Trial" (jsalinas has 14 days left), an "Upgrade" button, and a user profile for "jesus.salinas". Below the header, the main area is titled "MY hub". On the left, there's a sidebar with navigation links like "Create New", "MY hub" (which is selected and highlighted in blue), "Search", and user profiles for "cleformacion" and "jsalinas". The main content area displays two API entries. The first entry is for "jsalinas" with the name "api-1", described as a "Simple Inventory API" that is "PRIVATE | UNPUBLISHED". The second entry is for "cleformacion" with the name "curso-1", also described as a "Simple Inventory API" that is "PUBLIC | UNPUBLISHED". Both entries have "API" and "OAS3" buttons and small circular icons for notifications, sharing, and deletion.

Owner	Name	Description	Type	Status
jsalinas	api-1	This is a simple API	API	PRIVATE UNPUBLISHED
cleformacion	curso-1	This is a simple API	API	PUBLIC UNPUBLISHED