Swagger

v1.1.4 2020-01-09

Contenidos

1. Instalacion

El editor se encuentra disponible online en http://editor.swagger.io, pero tambien podemos tenerlo instalado enm nuestra máquina.

Crearemos nuestra documentación utilizando la herramienta Swagger Editor. Para disponer de ella localmente debemos instalar el siguiente software:

- Node.js
- NPM (incluído en la instalación de Node.js)

Para instalar Node.js debemos descargarlo de https://nodejs.org/es/, recomendandose la versión LTS. Durante la instalación aceptaremos las opciones marcadas por defecto.

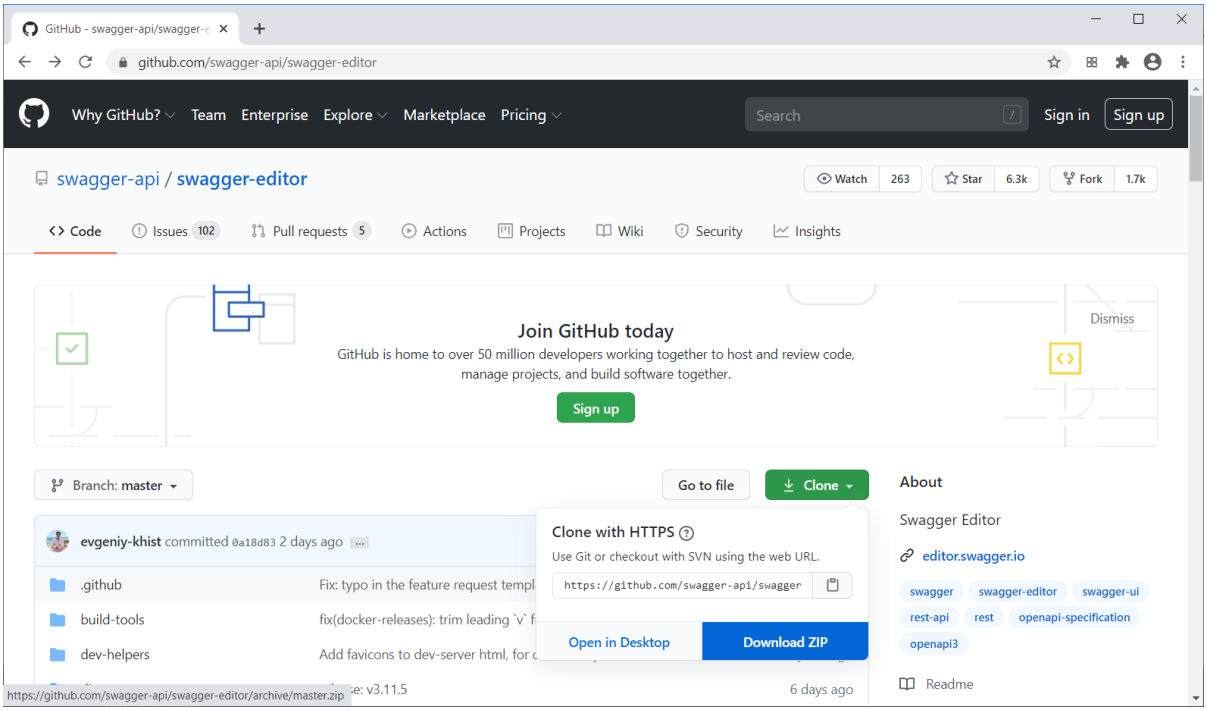
Una vez instalado Node procederemos a obtener el código de Swagger Editor (no es un software que se instale)

- https://github.com/swagger-api/swagger-editor

Podemos clonar el repositorio (debemos tener instalado git)

- Creamos un directorio, por ejemplo 'swagger'
- Entramos y ejecutamos el comando 'git init'
- Clonamos el repositorio con 'git clone https://github.com/swagger-api/swagger-editor.git'

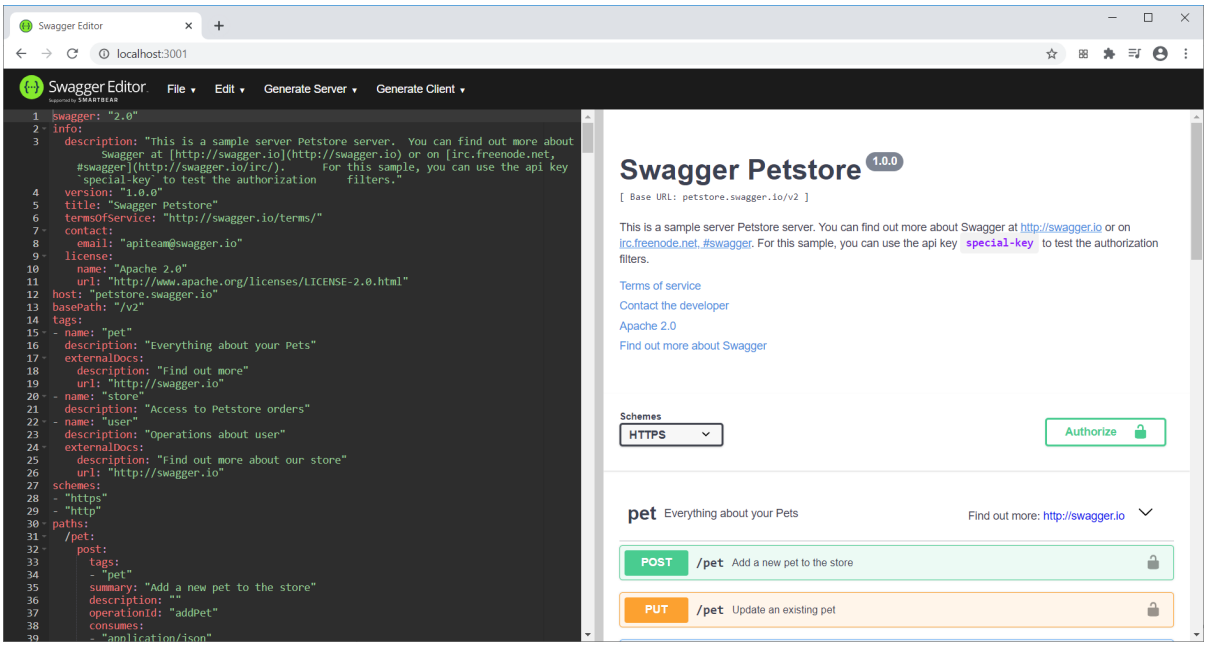
También podemos descargar el repositorio como un zip para decomprimirlo en un directorio a nuestra elección



A continuación abrimos una consola, navegamos hasta el directorio donde está el código y ejecutamos el siguiente comando:

- npm start

Tras unos segundos podemos acceder a la url 'http://localhost:3001' para comprobar que Swagger Editor está disponible (se trata de una aplicación web)



2. OpenApi

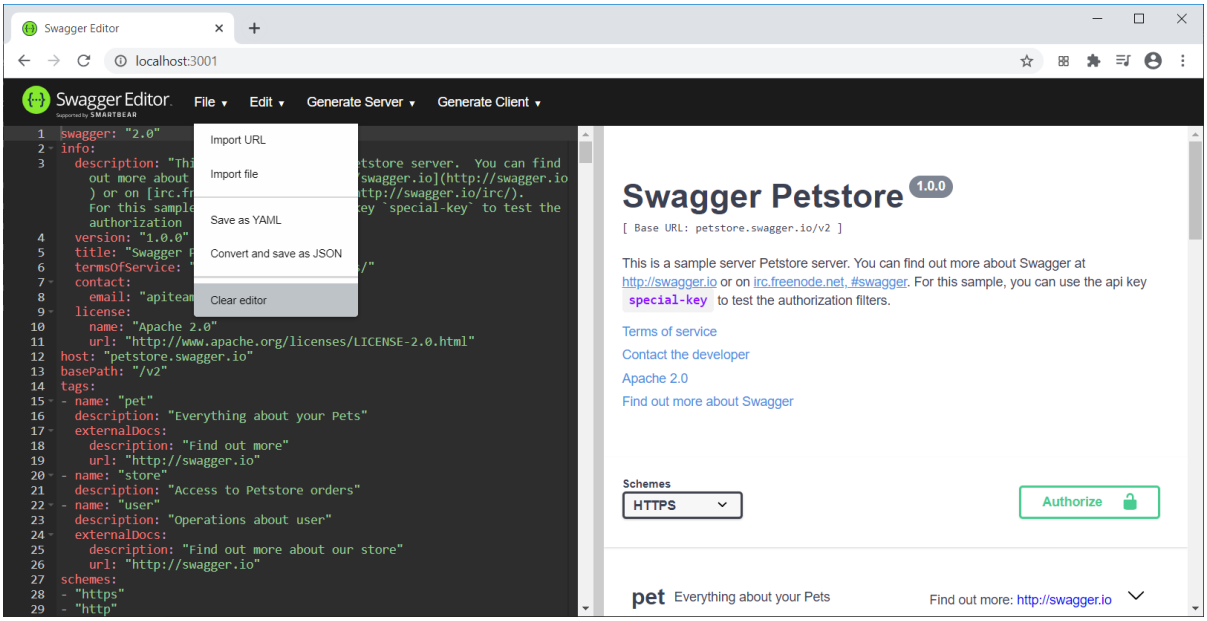
La especificación OpenAPI (OAS) es un formato para describir de una manera entendible por las aplicaciones del api de un servicio REST.

Antíguamente era conocida como 'Swagger Specification'.

Utilizando OAS podremos describir con todo detalle un api rest, indicando que uris incluye, que deben enviar los clientes y qé recibirán como respuesta. Se pueden definir en ficheros .json o .yaml, siendo este último formato el mas recomendable al ser más legible, sucinto y mantenible.

Para iniciar la creación de un documento primero debemos limpiar el editor (por defecto nos muestra un api de ejemplo). Para ello debemos:

- Seleccionar 'File' en el menú.
- Pulsar 'Clear editor'



asciidoctor == Cabecera del fichero

Un fichero OpenAPI comienza con cierta información general.

Versión de OAS:

```
openapi: "3.0.2"
```

Nombre del api, versión e información general:

```
info:
  title: "GestionClientes API"
  description: "Api que permite acceder a los clientes."
  version: "1.0"
  termsOfService: "https://url_a_los_terminos.org"
  contact:
```

```
    name: "Gestionclientes"
    url: "https://gestionclientes.org/api"
    email: "correo@gmail.com"
  license:
    name: "licencia"
    url: "https://url_a_la_licencia.org"
```

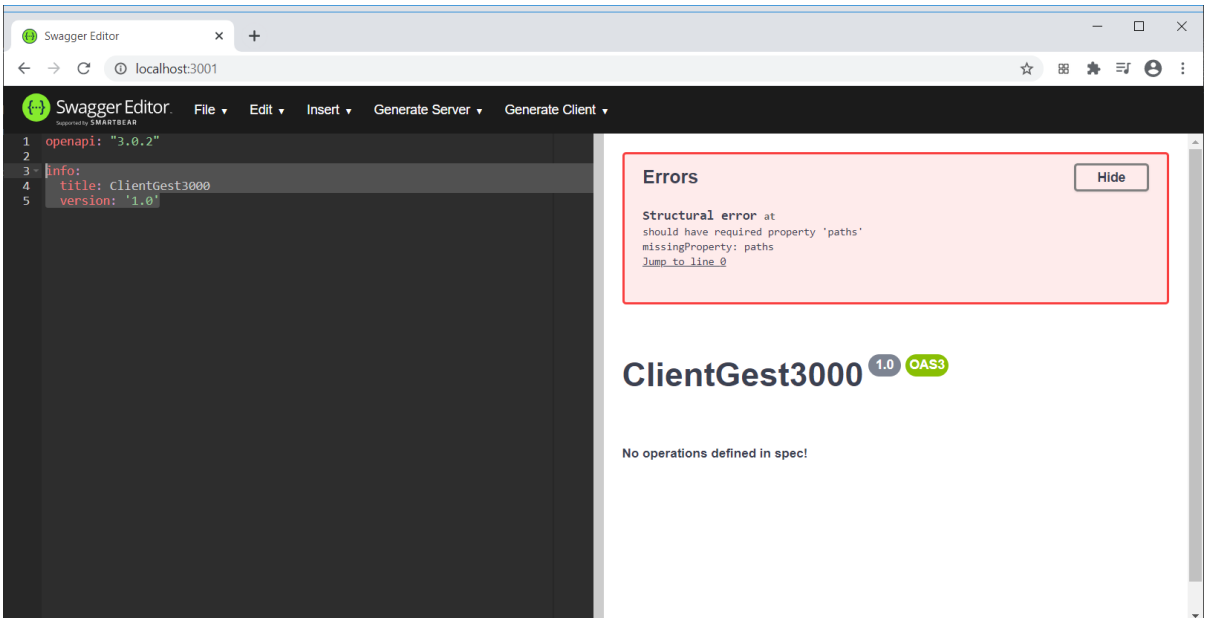
2.1. Server

Indicando el servidor definimos una 'base url' que se utilizará a lo largo de todo el documento

```
servers:
  - url: https://url_produccion
    description: Servidor de producción
  - url: http://url_integracion
    description: Servidor de integración
  - url: http://localhost:8080
    description: Servidor local
```

3. Paths

En este momento Swagger Editor mostrará un error puesto que no estamos indicando ninguna ruta



Los paths serán las distintas uris que publicará nuestro servicio. Las rutas tienen las siguientes propiedades

```
paths:
  /clientes:
    get:
      tags:
      summary:
      description:
      operationId:
      externalDocs:
      parameters:
      responses:
      deprecated: marca este path como en desuso
      security: para indicar si la seguridad de este path es distinta a la definida
globalmente
      servers: para indicar la url base (si es diferente a la referida en la sección
'servers')
      requestBody: para cuando en el body e la petición haya algo
      callbacks:
```

3.1. Parameters

Define los parámetros asociados a la petición

Parámetros de la query, van después de la interrogación:

- get /clientes?estado=Activo

```
paths:
  /clientes:
    get:

      parameters:
        - name: estado
          in: query
          description: "Utilizado para filtrar los clientes por estado. Estados válidos
'Activo' e 'Inactivo'"
          schema:
            type: string
```

Parámetros en la ruta

- get /clientes/5

```
paths:
  /clientes/{id}:
    get:

      parameters:
        - name: id
          in: path
          description: "id del cliente a buscar"
          schema:
            type: integer
          required: true
```

3.2. Tipos en swagger

Cuando indicamos la existencia de un parámetro o de cualquier otro valor que se adjunta a una petición o respuesta debemos indicar el tipo. Los tipos existentes son:

- string
- number
- integer
- boolean
- array
- object

3.3. Responses

Define las respuestas asociadas a las peticiones

```
paths:
  /clientes:
    get:
      responses:
        200:
```

```
        description: Petición procesada correctamente
        content:
          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  nombre:
                    type: string
                    description: Nombre del cliente
                  direccion:
                    type: string
                    description: Dirección del cliente
                  telefono:
                    type: string
                    description: Teléfono del cliente

      400:
        description: Peticion incorrecta
        content:
          text/plain:
            schema:
              title: Peticion incorrecta
              type: string
              example: Criterio de busqueda incorrecto

/clientes/{id}:
  get:
    parameters:
      - name: id
        in: path
        schema:
          type: integer
        required: true

    responses:
      200:
        description: Petición procesada correctamente
        content:
          application/json:
            schema:
              type: object
              properties:
                nombre:
                  type: string
                  description: Nombre del cliente
                direccion:
                  type: string
                  description: Dirección del cliente
                telefono:
                  type: string
                  description: Teléfono del cliente

      404:
        description: El cliente no existe
```

```
          content:
            text/plain:
              schema:
                title: Cliente no encontrado
                type: string
                example: Not found
```

4. Body en la petición

Para los métodos HTTP POST, PUT y PATCH es opcional incluir el body en la petición. Se define así:

```
paths:
  /clientes:
    post:
      summary: Inserta un cliente.
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                nombre:
                  type: string
                  description: Nombre del cliente
                direccion:
                  type: string
                  description: Dirección del cliente
                telefono:
                  type: string
                  description: Teléfono del cliente
      responses:
        201:
          description: Created
```

5. Reutilización de elementos, sección 'Components'

Es probable que distintos paths utilicen los mismos parámetros o los mismos objetos JSON en las peticiones y las respuesta. Podemos definir estos elementos de manera global para su reutilización en la sección 'components'.

5.1. Reutilización de parámetros

Los parámetros se pueden declarar globalmente de la siguiente forma:

```
components:

  parameters:
    estado:
      name: estado
      in: query
      description: "Utilizado para filtrar por estado. Estados válidos 'Activo' e 'Inactivo'"
      schema:
        type: string
```

A partir de ahora podremos referenciarlo así:

```
paths:
  /clientes:
    get:
      parameters:
        - $ref: '#/components/parameters/estado'
```

5.2. Reutilización de esquemas json

Definicion:

```
components:

  schemas:
    Cliente:
      title: Cliente
      type: object
      properties:
        nombre:
          type: string
          description: Nombre del cliente
        direccion:
          type: string
          description: Dirección del cliente
        telefono:
          type: string
          description: Teléfono del cliente
```

Utilización:

```
paths:
  /clientes:
    get:
      parameters:
      - name: estado
        in: query
        description: "Utilizado para filtrar los clientes por estado. Estados válidos 'Activo'
e 'Inactivo'"
        schema:
          type: string

      responses:
        200:
          description: Petición procesada correctametne
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Cliente'

  /clientes/{id}:
    get:
      parameters:
      - name: id
        in: path
        description: "id del cliente a buscar"
        schema:
          type: string
        required: true

      responses:
        200:
          description: Petición procesada correctametne
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Cliente'
```

6. Ejemplo completo

```
openapi: "3.0.2"

info:
  title: "GestionClientes API"
  description: "Api que permite acceder a los **clientes**."
  version: "1.0"
  termsOfService: "https://url_a_los_terminos.org"
  contact:
    name: "Gestionclientes"
    url: "https://gestionclientes.org/api"
    email: "correo@gmail.com"
  license:
    name: "licencia"
    url: "https://url_a_la_licencia.org"

servers:
  - url: https://url_produccion
    description: "Servidor de producción"
  - url: http://url_integracion
    description: "Servidor de integración"
  - url: http://localhost:8090
    description: "Servidor local"

components:

  parameters:
    id:
      name: id
      in: path
      description: "id del recurso a buscar"
      schema:
        type: integer
      required: true

  schemas:
    respuesta:
      type: object
      properties:
        status:
          type: string
          description: "status http de la respuesta"
    error:
      type: object
      properties:
        codigoError:
          type: integer
        mensaje:
          type: string
        detalles:
          type: object
    datos:
```

```
      type: object
      properties:
        descripcion:
          type: string
        valor:
          type: object

    cliente:
      title: Cliente
      type: object
      properties:
        id:
          type: integer
          description: "Id cliente"
        nombre:
          type: string
          description: "Nombre del cliente"
        direccion:
          type: string
          description: "Dirección del cliente"
        telefono:
          type: string
          description: "Teléfono del cliente"
    arrayClientes:
      type: array
      items:
        $ref: '#/components/schemas/cliente'

    clienteSinId:
      title: Cliente sin id
      type: object
      properties:
        nombre:
          type: string
          description: "Nombre del cliente"
        direccion:
          type: string
          description: "Dirección del cliente"
        telefono:
          type: string
          description: "Teléfono del cliente"

  responses:
    400_BAD_REQUEST:
      description: Error en la peticion
      content:
        application/json:
          schema:
            allOf:
            - $ref: '#/components/schemas/respuesta'
            - type: object
              properties:
                error:
                  $ref: '#/components/schemas/error'

    404_NOT_FOUND:
```

```
      description: Recurso no encontrado
      content:
        application/json:
          schema:
            allOf:
            - $ref: '#/components/schemas/respuesta'
            - type: object
              properties:
                error:
                  $ref: '#/components/schemas/error'

    500_INTERNAL_SERVER_ERROR:
      description: Error al procesar la petición
      content:
        application/json:
          schema:
            allOf:
            - $ref: '#/components/schemas/respuesta'
            - type: object
              properties:
                error:
                  $ref: '#/components/schemas/error'

paths:
  /clientes:
    get:
      summary: Lista clientes utilizando un criterio para el filtrado de resultados
      responses:
        200:
          description: Petición procesada correctamente
          content:
            application/json:
              schema:
                allOf:
                - $ref: '#/components/schemas/respuesta'
                - type: object
                  properties:
                    datos:
                      allOf:
                        - $ref: '#/components/schemas/datos'
                        - type: object
                          properties:
                            valor:
                              $ref: '#/components/schemas/arrayClientes'
        400:
           $ref: '#/components/responses/400_BAD_REQUEST'
        500:
           $ref: '#/components/responses/500_INTERNAL_SERVER_ERROR'

    post:
      summary: Inserta un cliente.
      requestBody:
        required: true
        content:
          application/json:
            schema:
```

```
            $ref: '#/components/schemas/clienteSinId'
    responses:
      201:
        description: Cliente insertado correctamente
        content:
          application/json:
            schema:
              allOf:
              - $ref: '#/components/schemas/respuesta'
              - type: object
                properties:
                  datos:
                    allOf:
                      - $ref: '#/components/schemas/datos'
                      - type: object
                        properties:
                          valor:
                            $ref: '#/components/schemas/cliente'
      400:
         $ref: '#/components/responses/400_BAD_REQUEST'
      500:
         $ref: '#/components/responses/500_INTERNAL_SERVER_ERROR'

  /clientes/{id}:
    get:
      summary: "Busca un cliente por su identificador (id)"
      parameters:
        - $ref: '#/components/parameters/id'
      responses:
        200:
          description: Cliente encontrado
          content:
            application/json:
              schema:
                allOf:
                - $ref: '#/components/schemas/respuesta'
                - type: object
                  properties:
                    datos:
                      allOf:
                        - $ref: '#/components/schemas/datos'
                        - type: object
                          properties:
                            valor:
                              $ref: '#/components/schemas/cliente'
        404:
           $ref: '#/components/responses/404_NOT_FOUND'
        500:
           $ref: '#/components/responses/500_INTERNAL_SERVER_ERROR'

    delete:
      summary: "Elimina un cliente por su identificador (id)"
      parameters:
        - $ref: '#/components/parameters/id'
      responses:
        200:
```

```
          description: Cliente eliminado
          content:
            application/json:
              schema:
                allOf:
                - $ref: '#/components/schemas/respuesta'
                - type: object
                  properties:
                    datos:
                      allOf:
                        - $ref: '#/components/schemas/datos'
                        - type: object
                          properties:
                            valor:
                              $ref: '#/components/schemas/cliente'
        404:
           $ref: '#/components/responses/404_NOT_FOUND'
        500:
           $ref: '#/components/responses/500_INTERNAL_SERVER_ERROR'
```