Memoria de uso de Swagger para la generación automática de API Rest

La herramienta swagger nos permite documentar y generar automáticamente una API Rest. Hemos usado swagger para la generación de una parte de la API de Acme-Explorer (todo lo relacionado a la entidad Actor). En este documento explicaremos como lo hemos implementado.

A la hora de implementar una API con swagger tenemos dos opciones:
usar el editor online o instalar swagger en nuestra máquina.

- **Swagger en local:** para trabajar con swagger en local en nuestra máquina, debemos instalarlo con “npm install -g swagger”, luego crearemos el proyecto con “swagger project create acme-explorer”
- **Swagger hub:** podemos usar swagger hub para crear nuestra api online. Para ello nos crearemos una cuenta en https://app.swaggerhub.com y ahí crearemos un projecto. Esta aplicación te permite añadir colaboradores para trabajar con los otros miembros del grupo.

Nosotros hemos optado por la opción de Swagger hub.

Una vez tenemos el proyecto creado, tenemos que definir nuestra api siguiendo la sintaxis de swagger.

En primer lugar definiremos los información basica de nuestra app: título, descripción, versión, contacto, licencia

```
info:
  description: |
    This is Acme-explorer  server. It has been created by Pedro, Manuel
      and Miguel for a university project.
  version: 1.0.0
  title: Acme Explorer
  contact:
    email: invented@acme-explorer.us.es
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html
```

A continuación creamos una etiqueta para cada entidad de nuestra API. Nosotros hemos listado Actor, Trip y Application como ejemplo.

```yaml
tags:
- name: Actor
  description: Operations about the actors
- name: Trip
  description: Everything about the trips of the system
- name: Application
  description: Aplications of the trips
```

A continuación definiremos cada ruta de nuestra api y dentro de cada ruta los métodos disponibles (get, put, post o delete).
Para nuestra entidad Actor, tenemos 4 rutas:

- /v1/actors: GET, POST, DELETE
- /v1/actors/{actorId}: GET, PUT, DELETE
- /v1/actors/{actorId}/ban: GET
- /v1/actors/{actorId}/unban: GET

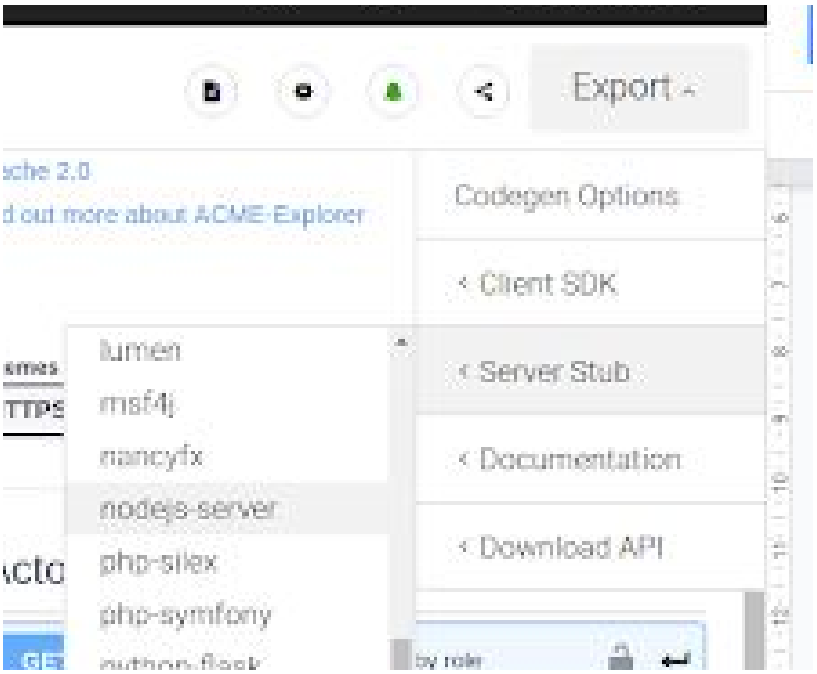
Describimos cada método con los siguientes parámetros:

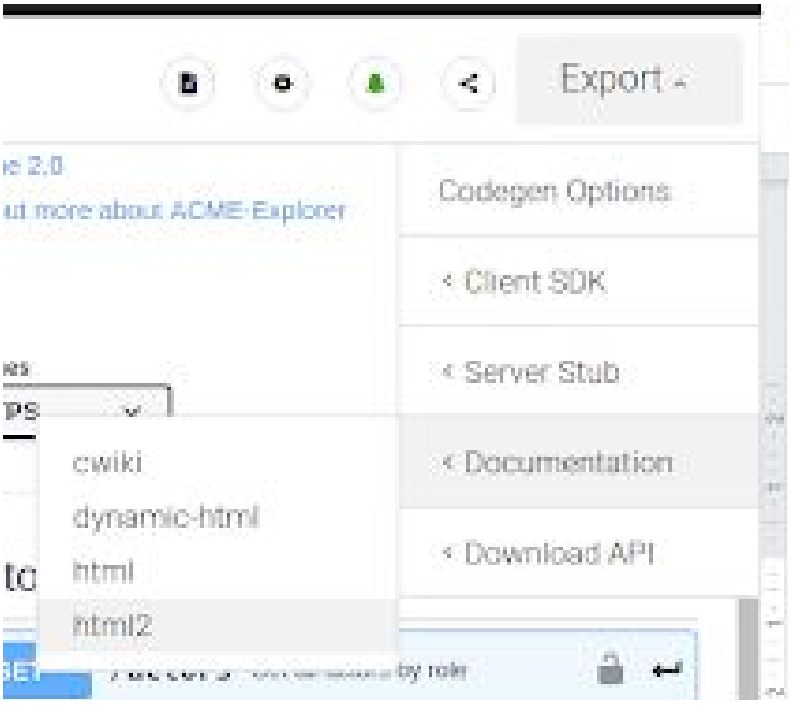
- tags: etiqueta a la que pertenece (por ejemplo Actor).
- summary: resumen del método.
- description: descripción del funcionamiento del método.
- operationId: id de la operación para referenciarla posteriormente.
- produces: formato de la respuesta (ejemplo application/json).
- parameters: parámetros de entrada.
- Response: posibles respuestas del servidor. Se definen con un código http, una descripción y el formato de respuesta.
- Security: permisos de autenticación necesarios.

Para definir la estructura de los objetos que la api consumirá y producirá, se crea un objeto en el apartado definitions. Aquí podemos ver un ejemplo del objeto Actor:

Por último, en securityDefinitions, podremos definir los diferentes roles y permisos que tienen los usuarios que consuman la api.

Una vez que tenemos bien definida nuestra API, podemos exportar el código node-js y la documentación de la api en html.


Export
Codegen Options
Client SDK
Server Stub
Documentation
Download API
lumen
msf4j
nancyfx
nodejs-server
php-silex
php-symfony


Export
Codegen Options
Client SDK
Server Stub
Documentation
Download API
cwiki
dynamic-html
html
html2