ACME-Explorer con Swagger

ACME Explorer es una empresa que organiza viajes de aventura alrededor de todo el mundo. El objetivo de este proyecto es desarrollar un sistema de información web que permita a ACME Explorer ejecutar los negocios contemplados en la empresa.

Los recursos principales que se tendrán que exponer para realizar este proyecto son los actores que participan en el negocio. Swagger es una herramienta que permite crear la documentación de APIs de forma intuitiva para que sea realmente útil para las personas que la necesitan.

1 DOCUMENTACIÓN ACME-EXPLORER CON SWAGGER

El primer paso que hay que seguir para implementar una API es tener muy claro el recurso que se quiere implementar, en este caso realizaremos el modelado de la entidad "Actors" que queda definido a continuación:

ACTOR	
Name	String
Surname	String
Email	String
phone	String
address	String
role	Enum

Se usará el módulo de Swagger para NodeJS como herramienta principal para definir nuestro API Rest, y poder probarla sin tener en cuenta inicialmente ciertos detalles de implementación, para ello deberemos instalar Swagger en nuestro sistema de forma global con la siguiente orden:

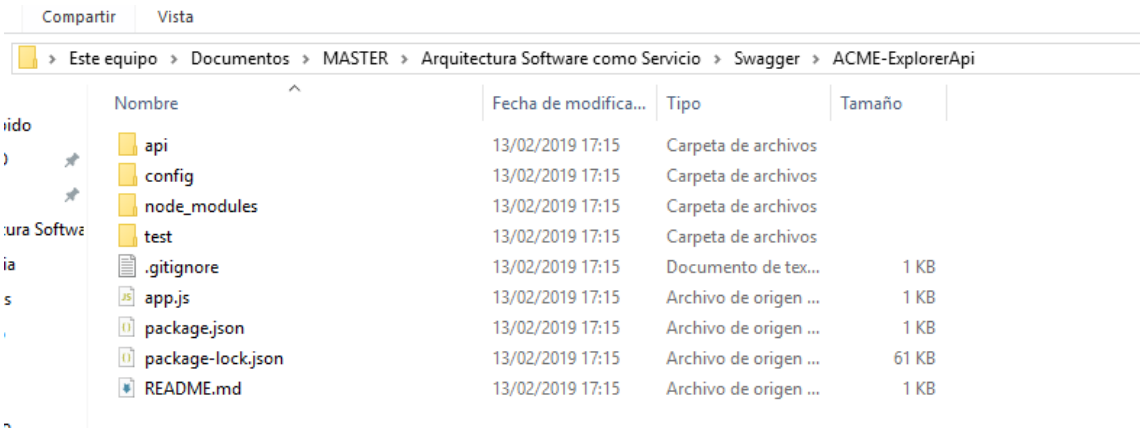
```
$ npm install -g swagger
```

1.1 CREACIÓN DEL PROYECTO

Para crear la carpeta del proyecto y una serie de ficheros iniciales de Swagger, hay que situarse en la carpeta que queremos que contenga nuestro poyecto y escribir en el terminal:

```
$ swagger project create <Nombre de la API>
```

Tras ejecutar esto, se nos creará una estructura de carpetas como se ve en la siguiente imagen:



- **Api:** es donde se implementará toda la lógica de la aplicación
- **Config:** se definirá la configuración específica de la aplicación
- **Node_modules:** carpeta típica de todo proyecto de Node en el que se indican las librerías externas que se utilizan
- **Test:** aquí se introducirán los test unitarios
- **App.js:** fichero típico de todo proyecto Express en el que se indica qué debe hacer la aplicación
- **Api/swagger/swagger.yaml:** es donde se define la documentación de la api.

1.2 ARQUITECTURA ARCHIVO .YAML

Swagger dispone de un editor online Swagger (https://editor.swagger.io/) editor que será el que se utilizará, ya que proporciona una forma muy cómoda de ir modificando el archivo swagger.yaml y ver directamente los resultados que se van obteniendo.

Este archivo .yaml básicamente se compone de los siguientes campos:

➔ **Metadata**

Aquí se incluyen la versión de la especificación OpenAPI que se utilizará, el título, la descripción y la versión.

```yaml
swagger: "2.0"
info:
  version: "0.0.1"
  title: ACME-Explorer
```

➔ **Servers**

Se especifica toda la información acerca del servidor, como son los documentos de entrada y de salida.

```yaml
host: localhost:10010
# basePath prefixes all resource paths
basePath: /
#
schemes:
  # tip: remove http to make production-grade
  - http
  - https
# format of bodies a client can send (Content-Type)
consumes:
  - application/json
# format of the responses to the client (Accepts)
produces:
  - application/json
```

➔ **Paths**

Los paths definen los endpoints de nuestra API, así como los métodos HTTP soportados por dichos endpoints.

➔ **Parámetros**

En esta sección se definen los parámetros de entrada para cada una de las distintas operaciones.

➔ **Responses**

Se indican las respuestas a cada una de las llamadas que se han definido previamente.

Además de estos campos hay algunos adicionales que pueden ser consultados en la documentación web de Swagger https://swagger.io/docs/specification/basic-structure/

2 DEFINICIÓN DEL API PARA EL RECURSO ACTOR

Las operaciones principales que se implementarán serán las típicas CRUD, es decir, obtención del recurso, inserción, actualización y borrado, que se corresponden con las típicas GET, POST, PUT y DELETE de http.

Básicamente, en todas las definiciones que se hacen en el archivo .yaml ya que definir los parámetros de entrada de la llamada, así como las respuestas que dará en caso de que funcione o de que vaya mal. También hay que especificar cuál es el nombre del controlador que va a realizar las llamadas a la API mediante el atributo x-swagger-router-controller (En nuestro caso el controlador se llama actorController) y por último, en cada una de las operaciones a la API se indicará también cuál es el nombre de la función concreta dentro del controlador que se encarga de realizar dicha operación.

A continuación se mostrará el código utilizado para cada una de las operaciones y se mostrará una prueba del funcionamiento de la API usando un mock que proporciona Swagger para poder hacer pruebas sin tener la API implementada.

2.1 OBTENCIÓN DE LA LISTA DE ACTORES (GET)

```
## GET
    get:
      description: get the actors list
      operationId: list_all_actors
      responses:
        "200":
          description: Success
          schema:
            $ref: "#/definitions/GetActorsListResponse"
        default:
          description: Error
          schema:
            $ref: "#/definitions/ErrorResponse"
```

Lo que se expone arriba es lo que se muestra en el editor de Swagger, en la interfaz gráfica que se nos proporciona se vería lo siguiente:

ACME-Explorer API 0.0.1

[Base URL: localhost:10010/]

Schemes

HTTP

default

GET /actors

Models

Si levantamos el servidor en modo mock con la orden $ swagger Project start -m observamos que nos da una respuesta con unos datos aleatorios:

Code	Details
200	Response body


```
{
  "actors": [
    {
      "_id": 1,
      "name": "Sample text",
      "surname": "Sample text",
      "email": "Sample text",
      "phone": "Sample text",
      "address": "Sample text",
      "role": "ADMINISTRATOR"
    }
  ]
}
```


Download

Response headers

```
content-type: application/json
```

2.2 Inserción de un nuevo actor (POST)

```yaml
## POST
    post:
      description: add a new actor to the list
      operationId: create_an_actor
      parameters:
        - in: body
          name: name
          description: The actor to create.
          schema:
            $ref: "#/definitions/Actor"
      responses:
        "201":
          description: Success
          schema:
            $ref: "#/definitions/GetActorsResponse"
        default:
          description: Error
          schema:
            $ref: "#/definitions/ErrorResponse"
```

2.3 Eliminar todos los Actores (DELETE)

```yaml
delete:
        description: delete the actors list
        operationId: delete_all_actors
        responses:
          "200":
            description: Success
```

2.4 Obtener un Actor en concreto (GET/{id})

```yaml
get:
      description: get an actor
      operationId: read_an_actor
      parameters:
      - name: id
        type: string
        in: path
        required: true
      responses:
        "200":
          description: Error
          schema:
            $ref: "#/definitions/ErrorResponse"
```

2.5 Modificar un Actor existente (PUT/{id})

```yaml
    put:
      description: update an actor
      operationId: update_an_actor
      #define the parameters
      parameters:
        - name: id
          description: actor id
          type: string
          in: path
          required: true
        - in: body
          name: actor
          description: The actor to update
          schema:
            $ref: "#/definitions/Actor"
      responses:
        "200":
          description: Error
          schema:
            $ref: "#/definitions/ErrorResponse"
```

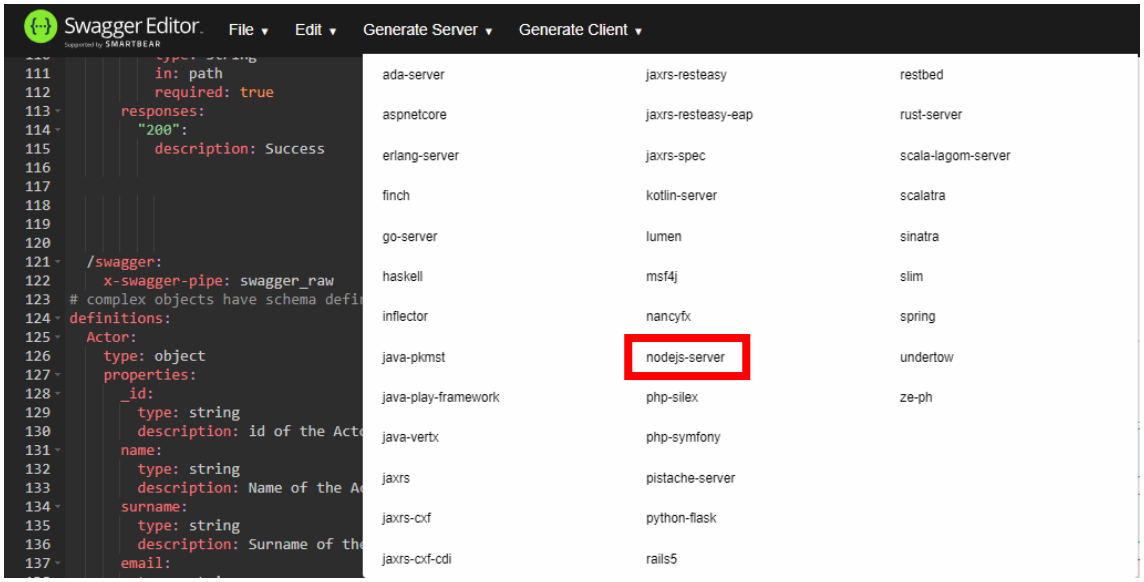
2.6 ELIMINAR UN ACTOR POR ID (DELETE/{ID})

```yaml
delete:
      description: delete an actor
      operationId: delete_an_actor
      parameters:
        - name: id
          description: actor
          type: string
          in: path
          required: true
      responses:
        "200":
          description: Success
```

Se observa en los códigos expuestos anteriormente que hay unos objetos llamados mediante referencias, estos objetos se han definido al final del archivo .yaml y son trozos de código que son reutilizados en algunas de las llamadas a la API por lo que se decidió definirlos de forma independiente para poder reutilizarlos. Se pueden consultar en el archivo .yaml adjunto, pero en definitiva son las definiciones de las respuestas a las llamadas a la API, así como el modelo de datos que sigue el recurso Actor.

3 CREAR API CON SWAGGER

Una vez que la documentación está finalizada, Swagger dispone de una herramienta que genera con un solo click el servidor definido en el .yaml. Para ello solo hay que elegir la opción “Generate Server” que aparece en la interfaz y elegir qué tipo de servidor queremos, en nuestro caso nodejs-server:



Tras hacer click en esta opción se nos descargará una carpeta que contiene la implementación de la API.