INFORME

Instalación y uso de Swagger API

Ricardo Baloira Armas

Daniel Barroso Rocío

ÍNDICE

Introducción

Swagger es un framework creado en el año 2010 con el objetivo de permitir a su usuario describir una API usando un lenguaje que cualquier desarrollador pueda entender. Esto se alcanza gracias a la interfaz de Swagger “Swagger UI”, la cual permite al usuario documentar y organizar los métodos y operaciones de la API de tal manera que tanto máquinas como desarrolladores sean capaces de leerla.

De esta manera se solventa la problemática de que las APIs, al no contar con un estándar de diseño común ni de patrones para ser documentadas, no sean comprensibles para las personas ajenas a su desarrollo. Swagger aporta las reglas y herramientas necesarias para documentar correctamente la API y, de esta forma, conseguir que sea entendida por toda persona con la que queramos compartirla. No obstante, para poder documentar nuestras APIs primero deberemos conocer los pasos a tomar para implementar Swagger en nuestros proyectos y, a continuación, navegar en su interfaz, los cuales procederemos a describir en los siguientes apartados.

Instalación

Para comentar nuestra API haremos uso de SpringFox, que es un conjunto de librerías que nos permite generar automáticamente la documentación de nuestro proyecto en formato Swagger.

En primer lugar necesitaremos incluir las dependencias pertinentes en nuestra API:

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

A continuación debemos por medio del código activar Swagger en el proyecto. Para ello crearemos una clase llamada “SpringFoxConfig” que contendrá las siguientes líneas de código:

```
@Configuration
@EnableSwagger2
public class SpringFoxConfig {

    @Primary
    @Bean
    public LinkDiscoverers discoverers() {
        List<LinkDiscoverer> plugins = new ArrayList<>();
        plugins.add(new CollectionJsonLinkDiscoverer());
        return new LinkDiscoverers(SimplePluginRegistry.create(plugins));

    }
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
          .select()
          .apis(RequestHandlerSelectors.any())
          .paths(PathSelectors.any())
          .build();
    }

}
```

Con la anotación “@Configuration” informamos a nuestra API de que esta se trata de una clase de configuración, mientras que con “@EnableSwagger2” habilitamos Swagger2, cuyas dependencias ya añadimos previamente al archivo “pom.xml” del proyecto y que será la aplicación que nos permitirá documentar la API.

“Docket” es una clase de SpringFox y el constructor designado como interfaz principal dentro del framework “swagger-springmvc”. Con la anotación “@Bean” indicamos que este constructor será por tanto encargado de encapsular información y que implementa las características que por convención poseen los JavaBean, siendo por consiguiente “Docket” un constructor público sin argumentos que aplica Serializable y cuyas propiedades son privadas.

Como se puede observar en la imagen, dentro del constructor definimos “.select()” seguido de “.apis” para que, de esta forma, se seleccione a través de la clase “RequestHandlerSelectors” el paquete base para nuestras clases API REST. En este ejemplo en lugar de especificarse un paquete concreto escribimos “.any()” a continuación de “RequestHandlerSelectors” para así generar documentación para todos los paquetes del proyecto. Esto último se aplica también a “PathSelectors”, pues gracias al uso de “.any()” se define que queremos crear documentación para todas las rutas de nuestra API.

Cabe mencionar que el constructor “discoverers” tal cual es expuesto en el ejemplo es requerido en esta clase a fin de evitar que un error nos impida la ejecución de la API a causa del método “linkDiscoverers”, situado en “org.springframework.hateoas.config.HateoasConfiguration”. Al marcar además este bean con la anotación @Primary se le indica al programa qué bean debe ser consumida, previniendo así que aparezca

el error antes mencionado en el que se indica que “linkDiscoverers” precisa de un solo bean.

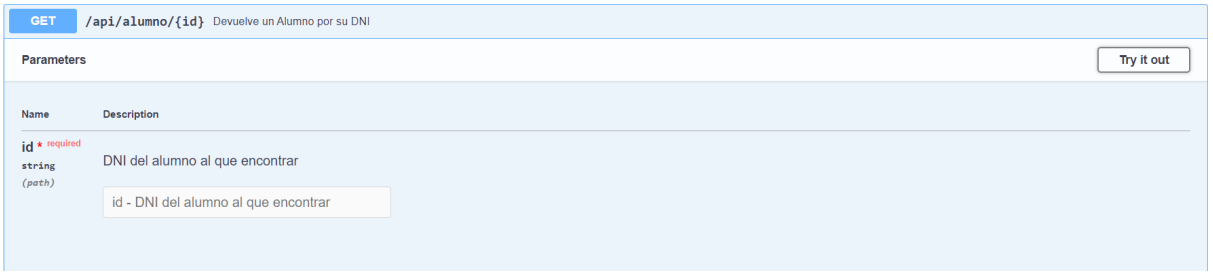
Uso

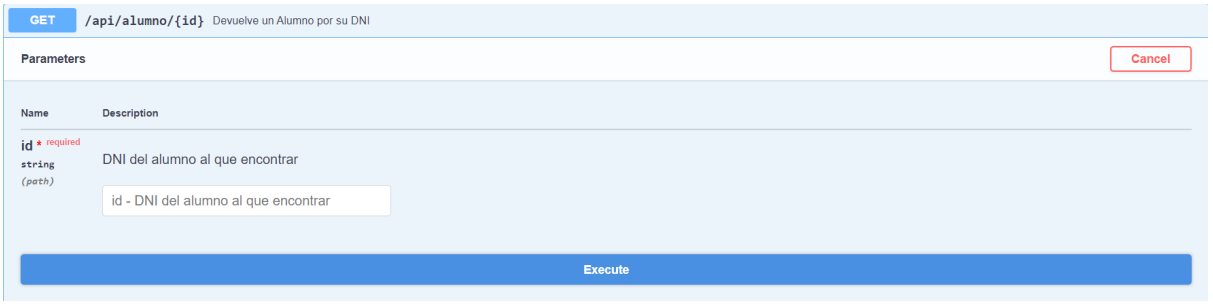
Ya configurado Swagger para su uso podemos, una vez iniciada nuestra API, acceder a la documentación de dicha API mediante el siguiente enlace:

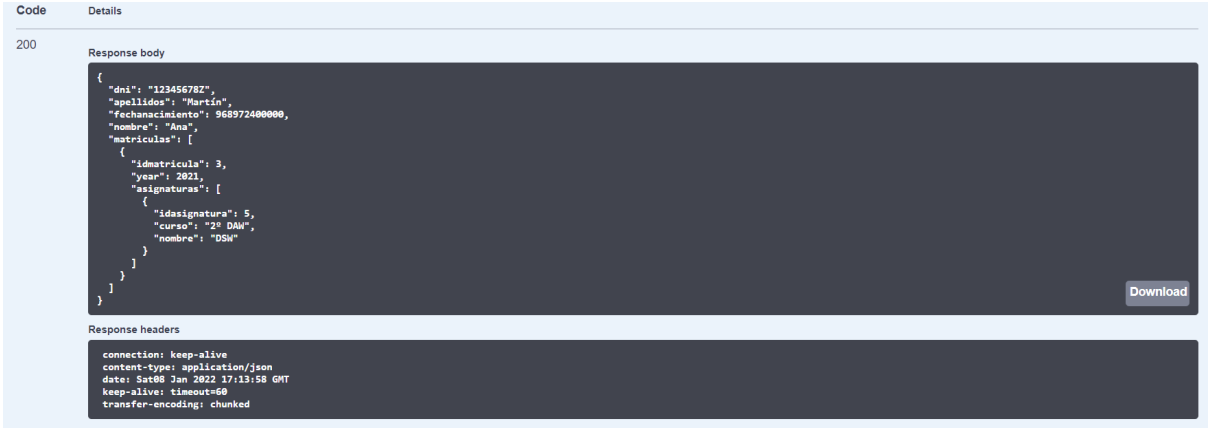
http://localhost:8080/swagger-ui/

Esta documentación se ha generado automáticamente teniendo en cuenta todos los paquetes del proyecto, dada la configuración que hemos seleccionado para Swagger en la clase “SpringFoxConfig”.

La documentación la podemos modificar por medio de anotaciones que podríamos incluir de así desearlo, aparte de probar el funcionamiento de cada comando REST introduciendo diferentes valores y pulsando en el botón “Try it out” para seguidamente pulsar en “Execute”, devolviéndonos Swagger los resultados de dichas pruebas en tiempo real.







Para agregar comentarios personalizados en los parámetros de la API debemos realizar las anotaciones pertinentes en el código del proyecto. Algunos ejemplos son:

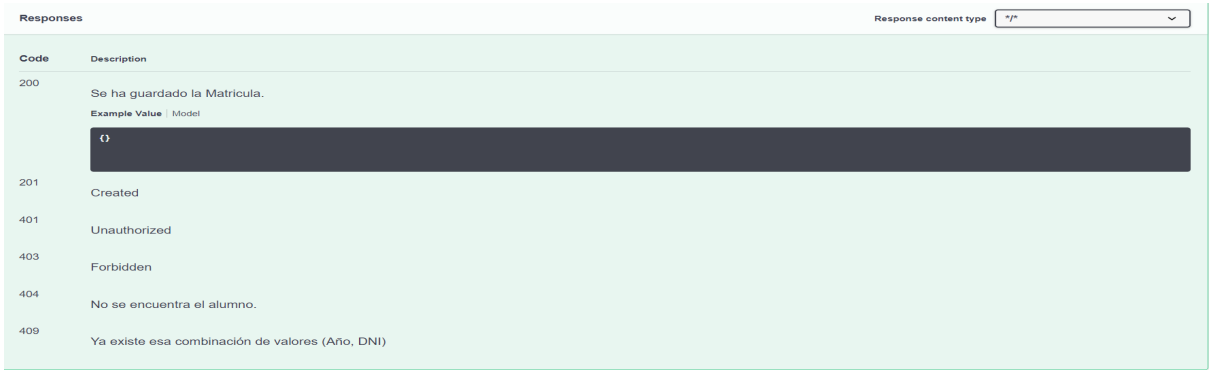
```java
package es.iespuertodelacruz.daniel.matriculasrest.controller;

import java.util.ArrayList;

@Api(value = "AlumnoREST", description = "REST APIs relacionadas con las entidades Alumno y Matricula")
@RestController
@RequestMapping("/api/alumno")
public class AlumnoREST {
```

Usamos la anotación @Api en el controlador para añadir información básica de dicho controlador.

```
@ApiOperation(value = "Guarda una matricula",
        response = Matricula.class,
        notes = "Devolvemos un String que diga si hay algun problema, y la Matricula si la guardamos con éxito")
@ApiResponses(value = {
          @ApiResponse(code = 409, message = "Ya existe esa combinación de valores (Año, DNI)"),
          @ApiResponse(code = 404, message = "No se encuentra el alumno."),
          @ApiResponse(code = 200, message = "Se ha guardado la Matricula.") })
@PostMapping("/{idalu}/matricula")
public ResponseEntity<?> saveMatricula(
        @ApiParam(value = "DNI del alumno dueño de la matrícula a añadir", required = true)
        @PathVariable("idalu") String idAlu,
        @ApiParam(value = "Matricula que queremos guardar en la base de datos", required = true)
        @RequestBody MatriculaDTO matriculaDto) {
    Optional<Alumno> optAlumno = alumnoService.findById(idAlu);
```



Con la anotación @ApiOperation añadimos información de un método concreto de la API, concretando por medio de “notes” aquella información que consideremos importante añadir a la ya dada en “value”. Adicionalmente, podemos usar @ApiResponses para agregar información a cada una de las respuestas posibles del método en cuestión (200, 404…). Como podemos observar en Swagger UI la información relativa a los parámetros y variables del método en cuestión, comentadas mediante las anotaciones @ApiParam, se encuadran en el apartado “Parameters”, en tanto que los comentarios agregados a las diferentes respuestas, realizados por medio de las anotaciones @ApiResponse, se muestran en el apartado “Responses”.