Isabel González Anzano

ÍNDICE

¿QUÉ ES SWAGGER?

las APIs no cuentan con un estándar de diseño común y ni si quiera existen unos parámetros comunes para documentarlas.

Su objetivo es estandarizar el vocabulario que utilizan las APIs. Es el diccionario API.

Cuando hablamos de Swagger nos referimos a una serie de reglas, especificaciones y herramientas que nos ayudan a documentar nuestras APIs. De esta manera, podemos realizar documentación que sea realmente útil para las personas que la necesitan. Swagger nos ayuda a crear documentación que todo el mundo entienda.

Con Swagger la documentación puede utilizarse directamente para automatizar procesos dependientes de APIs.

SWAGGER UI – LA INTERFAZ DE USUARIO DE SWAGGER

Swagger UI es una de las herramientas atractivas de la plataforma. Para que una documentación sea útil necesitaremos que sea navegable y que esté perfectamente organizada para un fácil acceso. Por esta razón, realizar una buena documentación puede ser realmente tedioso y consumir mucho tiempo a los desarrolladores.

Swagger UI utiliza un documento JSON o YAML existente y lo hace interactivo. Crea una plataforma que ordena cada uno de nuestros métodos (get, put, post, delete) y categoriza nuestras operaciones. Cada uno de los métodos es expandible, y en ellos podemos encontrar un listado completo de los parámetros con sus respectivos ejemplos. Incluso podemos probar valores de llamada.

Esta no es la única herramienta útil con la que cuenta Swagger, “Editor” es otra herramienta muy interesante que nos ayudará a identificar los errores de nuestras documentación en YAML o JSON. Simplemente subiendo nuestra documentación a la plataforma la comparará con las especificaciones Swagger Editor, no solo identificará los errores que hemos cometido, sino que nos planteará sugerencias y nos dará alternativas para que nuestra documentación sea perfecta.

INCLUIR SWAGGER EN TU PROYECTO

CONFIGURACIÓN

Primero hemos de añadir las LIBRERÍAS de swagger al proyecto, en caso de que utilices **Maven**:

```
<!-- SWAGGER -->
		<!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger2 -->
		<dependency>
			<groupId>io.springfox</groupId>
			<artifactId>springfox-swagger2</artifactId>
			<version>2.9.2</version>
		</dependency>
		<!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger-ui -->
		<dependency>
			<groupId>io.springfox</groupId>
			<artifactId>springfox-swagger-ui</artifactId>
			<version>2.9.2</version>
		</dependency>
```

Si por el contrario utilizas **Gradle**:

```
compile group: 'io.springfox', name: 'springfox-swagger2', version: '2.9.2'
compile "io.springfox:springfox-swagger-ui:2.9.2"
dev("org.springframework.boot:spring-boot-devtools")
```

Necesitamos incluir una clase tipo, llamada SWAGGERCONFIGURATION.

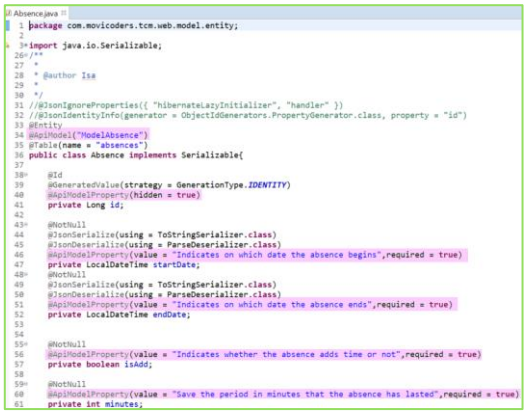
SwaggerConfiguration.java

```java
package com.movicoders.tcm.web.config;


import com.google.common.base.Predicate;
import com.google.common.base.Predicates;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;

import springfox.documentation.service.ApiInfo;

import springfox.documentation.spi.DocumentationType;

import springfox.documentation.spring.web.plugins.Docket;

import springfox.documentation.swagger2.annotations.EnableSwagger2;


import static springfox.documentation.builders.PathSelectors.regex;


@EnableSwagger2
@Configuration
public class SwaggerConfiguration {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
          .select()
          .apis(RequestHandlerSelectors.any())
          .paths(Predicates.not(PathSelectors.regex("/error.*")))
          .build().forCodeGeneration(true);
    }
}
```

NOTACIONES

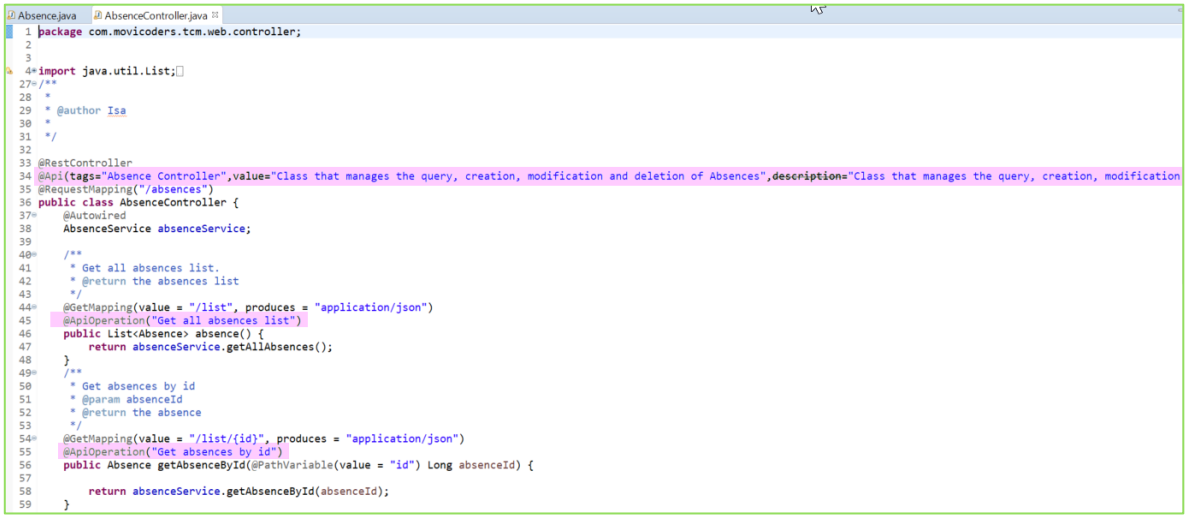
Para cada `@ENTITY`, es decir, para cada modelo de nuestra aplicación deberemos añadir **@ApiModel("Model + Nombre de la Entity")** y para sus atributos **@ApiModelProperty** (tiene varias opciones para ajustarlo a las necesidades de cada tipo).

Por ejemplo:



```java
package com.movicoders.tcm.web.model.entity;

import java.io.Serializable;
/**
 *
 * @author Isa
 *
 */
//@JsonIgnoreProperties({ "hibernateLazyInitializer", "handler" })
//@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "id")
@Entity
@ApiModel("ModelAbsence")
@Table(name = "absences")
public class Absence implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @ApiModelProperty(hidden = true)
    private Long id;

    @NotNull
    @JsonSerialize(using = ToStringSerializer.class)
    @JsonDeserialize(using = ParseDeserializer.class)
    @ApiModelProperty(value = "Indicates on which date the absence begins",required = true)
    private LocalDateTime startDate;
    @NotNull
    @JsonSerialize(using = ToStringSerializer.class)
    @JsonDeserialize(using = ParseDeserializer.class)
    @ApiModelProperty(value = "Indicates on which date the absence ends",required = true)
    private LocalDateTime endDate;


    @NotNull
    @ApiModelProperty(value = "Indicates whether the absence adds time or not",required = true)
    private boolean isAdd;

    @NotNull
    @ApiModelProperty(value = "Save the period in minutes that the absence has lasted",required = true)
    private int minutes;
```

Para cada `@CONTROLLER` habrá que marcarlo con **@Api(tags="Nombre del controlador", value="descripcion",description="descripcion")** y para sus métodos **@ApiOperation("describir lo que hace el método")**



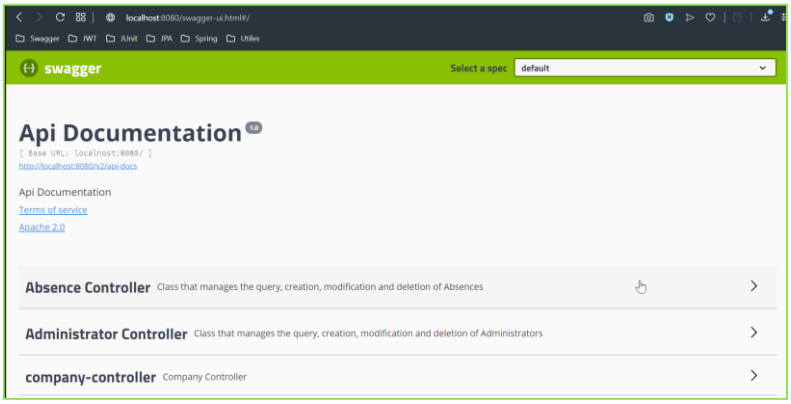
```java
package com.movicoders.tcm.web.controller;


import java.util.List;
/**
 *
 * @author Isa
 *
 */

@RestController
@Api(tags="Absence Controller",value="Class that manages the query, creation, modification and deletion of Absences",description="Class that manages the query, creation, modification
@RequestMapping("/absences")
public class AbsenceController {
    @Autowired
    AbsenceService absenceService;

    /**
     * Get all absences list.
     * @return the absences list
     */
    @GetMapping(value = "/list", produces = "application/json")
    @ApiOperation("Get all absences list")
    public List<Absence> absence() {
        return absenceService.getAllAbsences();
    }
    /**
     * Get absences by id
     * @param absenceId
     * @return the absence
     */
    @GetMapping(value = "/list/{id}", produces = "application/json")
    @ApiOperation("Get absences by id")
    public Absence getAbsenceById(@PathVariable(value = "id") Long absenceId) {

        return absenceService.getAbsenceById(absenceId);
    }
```

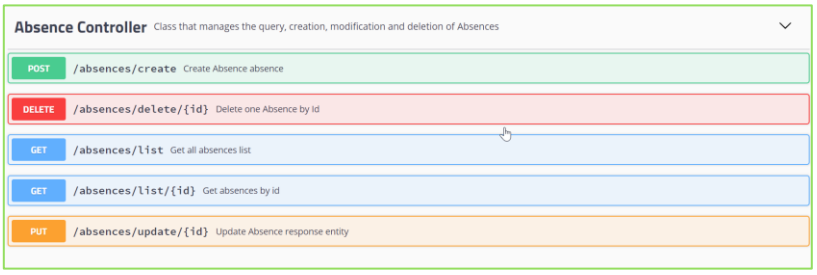
ACCEDER A SWAGGER UI

Una vez hayas anotado cada entidad y cada controlador puedes acceder desde el mismo proyecto al swagger a través de la url:

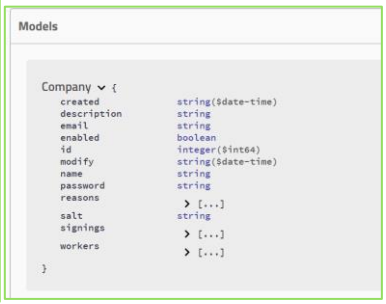
http://localhost:8080/swagger-ui.html

Podrás ver los controladores:





Y los modelos:



SWAGGER Y SPRING SECURITY

Si en tu proyecto se ha implementado Spring Security, es posible que no te permita acceder a la url, aunque la incluyas en la lista de permitidos.

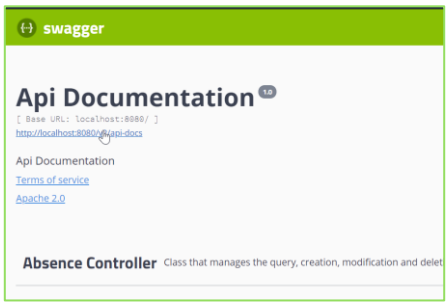
Por lo que una solucion es añadir a tu fichero de Configuración de Spring Security:

Absence.java | AbsenceController.java | SpringSecurityConfig.java

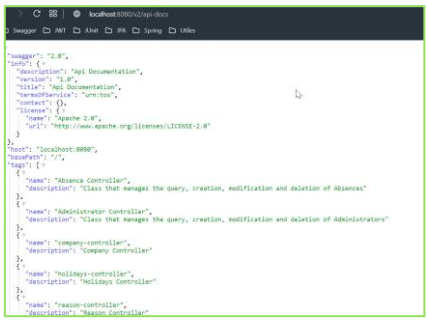
```
@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true)
@Configuration
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private BCryptPasswordEncoder passwordEncoder;

    @Autowired
    private JpaUserDetailsService userDetailsService;

    @Autowired
    private JWTService jwtService;


    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests().antMatchers("/","/css/**","/js/**","/images/**","/api/login").permitAll()
        .anyRequest().authenticated()
        .and()
        .addFilter(new JWTAuthenticationFilter(authenticationManager(),jwtService))
        .addFilter(new JWTAuthorizationFilter(authenticationManager(),jwtService))
        .csrf().disable()
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);  //We mark that the login will be stateless.
    }


    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().antMatchers("/v2/api-docs",
                                   "/configuration/ui",
                                   "/swagger-resources/**",
                                   "/configuration/security",
                                   "/swagger-ui.html",
                                   "/webjars/**");
    }

    //Method to configure and register the users of our system:
    @Autowired
    public void configurerGlobal(AuthenticationManagerBuilder builder) throws Exception{
```

COMPARTIR TU SWAGGER

Tan sólo tienes que acceder a:



Se cargará tu api-doc :



Y ya tan sólo tienes que guardarlo y enviárselo a tus compañeros, los cuales para verlo sólo tienen que acceder a swagger - editor y pegar ahí el doc.