EBook Gratis

APRENDIZAJE swagger

Free unaffiliated eBook created from
Stack Overflow contributors.

#swagger

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: swagger

It is an unofficial and free swagger ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official swagger.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con swagger

Observaciones

Esta sección proporciona una descripción general de qué es Swagger y por qué un desarrollador puede querer usarlo.

También debe mencionar los temas grandes dentro de swagger y vincular a los temas relacionados. Dado que la Documentación para swagger es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

Examples

Introducción - Instalación - Configuración (Desarrollo en Node.js)

Introducción:

Swagger es un conjunto de reglas / especificaciones para un formato que describe las API REST. Proporciona un ecosistema de herramientas potente y activamente desarrollado en torno a esta especificación formal, como generadores de códigos y editores. La mejor parte de Swagger es que la documentación de los métodos, los parámetros y los modelos están estrechamente integrados en el código del servidor, lo que permite que las API se mantengan siempre sincronizadas. Aquí hay un enlace que ofrece una breve descripción de lo que es swagger: empezar.

Especificaciones de escritura:

Las especificaciones se pueden escribir en JSON o YAML. Y así hacemos el archivo swagger.json o swagger.yaml en consecuencia. El editor en línea se puede utilizar para crear el archivo. Aquí hay un enlace que describe la sintaxis de las especificaciones: http://swagger.io/specification/

Formas de usar swagger:

1. *Primer enfoque de API (enfoque de arriba a abajo):* use el editor swagger → escriba las definiciones swagger → use swagger-codegen y swagger-ui para generar API
2. *Primer enfoque del servicio (enfoque ascendente):* Desarrolle clases de recursos JAX-RS usando anotaciones de swagger → Use swagger-core para generar automáticamente las definiciones de swagger → Usar swagger-codegen y swagger-ui para generar API y documentación de clientes. Lo anterior se puede hacer durante la compilación maven durante el plugin maven swagger.

Instalación y configuración

En esta sección, instalaremos Swagger, configuraremos la interfaz de usuario de Swagger y generaremos el lado del servidor y el SDK del cliente. Para instalar Swagger utilizando el

administrador de paquetes de Node, ejecute el siguiente comando:

```
npm install -g swagger
```

El uso de la bandera '-g' asegurará que el módulo se instale globalmente. A continuación, crearemos un proyecto usando el siguiente comando:

```
swagger project create <project-name>
```

Esto le pedirá al usuario que seleccione un marco para desarrollar las API REST. Expreso se puede seleccionar para el mismo. Esto creará el directorio del proyecto con los siguientes elementos y un archivo README.md en cada uno de ellos:

- api /
    - controladores /
    - ayudantes /
    - simulacros /
    - pavonearse/
- config /
- prueba/
    - api /
        - controladores /
        - ayudantes
- app.js
- paquete.json

El servidor está básicamente listo ahora y se puede iniciar usando este comando para ejecutarse en la raíz del proyecto:

```
swagger project start
```

Si el servidor host está configurado como `localhost` y el número de puerto no se modifica en el archivo `app.js` , el servidor se inicia en: `http://localhost:10010` Ahora se puede usar la interfaz de usuario Swagger para desarrollar aún más nuestras API REST. Esto se puede iniciar en un nuevo terminal usando:

```
swagger project edit
```

Esto abrirá el editor de Swagger en una pestaña del navegador en un puerto generado aleatoriamente. En el archivo swagger.yaml se puede ver una solicitud de ejemplo GET de saludo. Cualquier cambio adicional en este archivo hará que el servidor se reinicie por sí solo.

En la sección de rutas, el valor utilizado para `x-swagger-router-controller` debe ser el nombre del archivo javascript en la carpeta de controladores. Como muestra, hello_world.js debe estar presente en el directorio de los controladores. Además, el valor para el parámetro `operationId` representa el nombre de la función en el archivo javascript anterior. Aquí es donde se debe escribir la lógica de negocios. Por lo tanto, nuestra configuración de Swagger está completa y puede utilizarse para desarrollar aún más nuestra API.

Lea Empezando con swagger en línea: https://riptutorial.com/es/swagger/topic/5361/empezando-

con-swagger

Capítulo 2: springfox

Examples

Anular mensajes de respuesta predeterminados

Springfox define un conjunto de mensajes de respuesta predeterminados que se aplican a todos los controladores API de forma predeterminada. Esto incluye, por ejemplo, `201 - Created` y `204 - No Content` , así como varias respuestas de `40x` . Puede haber casos en los que los mensajes de respuesta predeterminados no se apliquen a su API. Tienes que incorporar posibilidades para lidiar con esto:

- Puede desactivar el mensaje de respuesta predeterminado y definir el suyo propio mediante la anotación `@ApiResponses` .
- Puedes definir tus propios mensajes de respuesta globalmente.

Turno de mensajes de respuesta por defecto.

```
docket.useDefaultResponseMessages(false);
```

Ahora puede configurar sus mensajes de respuesta individuales en un nivel por controlador. P.ej

```
 @ApiResponses(value = {
        @ApiResponse(code=400, message = "This is a bad request, please stick to the API
description", response = RestApiExceptionModel.class),
        @ApiResponse(code=401, message = "Your request cannot be authorized.", response =
RestApiExceptionModel.class)
 })
```

Configure sus propios mensajes de respuesta predeterminados

```
ModelRef errorModel = new ModelRef("RestApiExceptionModel");
List<ResponseMessage> responseMessages = Arrays.asList(
        new
ResponseMessageBuilder().code(401).message("Unauthorized").responseModel(errorModel).build(),
        new
ResponseMessageBuilder().code(403).message("Forbidden").responseModel(errorModel).build(),
        new
ResponseMessageBuilder().code(404).message("NotFound").responseModel(errorModel).build());

docket.globalResponseMessage(RequestMethod.POST, responseMessages)
        .globalResponseMessage(RequestMethod.PUT, responseMessages)
        .globalResponseMessage(RequestMethod.GET, responseMessages)
        .globalResponseMessage(RequestMethod.DELETE, responseMessages);
```

Configura springfox usando swagger-ui en spring-boot

1. Obtenga springfox en su aplicación usando Maven o Gradle
2. Crea un nuevo bean Docket en tu aplicación y configúralo
3. Documentar su API de acuerdo a sus necesidades
4. Inicie su aplicación y vea los resultados obtenidos.

1 Obteniendo springfox con Maven

Agregue las dependencias para swagger2 y swagger-ui en su pom.xml

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.6.0</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.6.0</version>
</dependency>
```

2 Configura tu aplicación para usar swagger

Agregue la anotación `@EnableSwagger2` a su clase principal anotada `@SpringBootApplication` y cree un bean Docket swagger dentro de esta (o cualquier otra) clase de configuración.

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

Esta configuración generará una documentación de API sobre todos los controladores Spring dentro de su aplicación. Si necesita limitar la generación de documentación de la API a ciertos controladores, puede elegir entre varios RequestHandlerSelectors. Por ejemplo, puede generar su documentación de API en función de la estructura de su paquete utilizando `RequestHandlerSelectors.basePackage("your.package.structure")` o clases específicas basadas que haya anotado utilizando `RequestHandlerSelectors.withClassAnnotation(Api.class)` .

3 documenta tu API

Use las anotaciones como se describe en la documentación para mejorar las clases y los métodos de su controlador con información adicional. Para describir la información general de su api, como el título general, la descripción o la versión, use ApiInfoBuilder () dentro de su bean Docket.

Ejemplo para la definición de metadatos usando ApiInfoBuilder:

```
// Within your configuration class
public static ApiInfo metadata(){
    return new ApiInfoBuilder()
            .title("Your Title")
            .description("Your Description")
            .version("1.x")
            .build();
}

// Within your method that definies the Docket bean...
docket.apiInfo(metadata());
```

Lea springfox en línea: https://riptutorial.com/es/swagger/topic/7637/springfox

Capítulo 3: swagger-ui

Observaciones

Es bueno si tienes una buena idea sobre los siguientes componentes:

Conceptos de REST WS (JAX-RS) Uso de las anotaciones Aplicación web Los estándares API REST Maven y su dependencia

Examples

swagger-ui con jersey REST WS

Como dice el sitio web oficial de Swagger :

Swagger es definir una interfaz estándar, independiente del idioma para las API REST, que permite que tanto los humanos como las computadoras descubran y comprendan las capacidades del servicio sin acceso al código fuente, la documentación o la inspección de tráfico de la red. Cuando se define correctamente a través de Swagger, un consumidor puede comprender e interactuar con el servicio remoto con una cantidad mínima de lógica de implementación. Al igual que las interfaces que han hecho para la programación de nivel inferior, Swagger elimina las conjeturas al llamar al servicio.

Suponiendo que use Maven y jersey, deberá agregar la siguiente dependencia: Maven Swagger Dependency junto con las dependencias de JAX-RS. Ahora tiene que crear "aplicación web maven" y debajo de la aplicación web necesita pegar los contenidos presentes en esta URL . La estructura de carpetas de la aplicación web debe tener el siguiente aspecto después de pegar esos contenidos en su proyecto:

Después de eso siga estos pasos:

1. Cree un archivo java con cualquier nombre (en nuestro caso es " `ApiOriginFilter.java` ") similar a continuación:

```
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;

import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class ApiOriginFilter implements javax.servlet.Filter {
    /**
     * doFilter
     */
    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        HttpServletResponse res = (HttpServletResponse) response;
        res.addHeader("Access-Control-Allow-Origin", "*");
        res.addHeader("Access-Control-Allow-Methods", "GET, POST, DELETE, PUT");
        res.addHeader("Access-Control-Allow-Headers", "Content-Type, api_key, Authorization");
        chain.doFilter(request, response);
    }

    /**
     * destroy
     */
    @Override
    public void destroy() {
    }

    /**
     * init
     */
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }
}
```

Este archivo se asegurará de filtrar las solicitudes entrantes según lo dispuesto en la clase.

2. Cree un archivo java con cualquier nombre, supongamos que en nuestro caso su nombre es " `SwaggerJaxrsConfig.java` " como sigue:

```
import org.eclipse.persistence.jaxb.MarshallerProperties;
import org.eclipse.persistence.jaxb.BeanValidationMode;
import org.glassfish.jersey.moxy.json.MoxyJsonConfig;
import org.glassfish.jersey.moxy.json.MoxyJsonFeature;
import org.glassfish.jersey.moxy.xml.MoxyXmlFeature;
import org.glassfish.jersey.server.ResourceConfig;

import org.glassfish.jersey.server.ServerProperties;
import org.glassfish.jersey.server.filter.RolesAllowedDynamicFeature;
import io.swagger.jaxrs.config.BeanConfig;

public class SwaggerJaxrsConfig extends ResourceConfig {

public SwaggerJaxrsConfig() {
    BeanConfig beanConfig = new BeanConfig();
    beanConfig.setTitle("Swagger API Title");
```

```
    beanConfig.setVersion("1.0.0");
    beanConfig.setSchemes(new String[] { "http" });
    beanConfig.setHost("localhost:8080/swagger-ui");
    beanConfig.setBasePath("/rest");
    beanConfig.setResourcePackage(
            "your.restws.package");
    beanConfig.setScan(true);
    property(ServerProperties.BV_SEND_ERROR_IN_RESPONSE, Boolean.TRUE);
    packages("your.restws.package");
    packages("io.swagger.jaxrs.listing");
    register(MoxyJsonFeature.class);
    // register(BadRequestExceptionMapper.class);
    register(new MoxyJsonConfig().setFormattedOutput(true)
    // Turn off BV otherwise the entities on server would be validated by MOXy as well.
    .property(MarshallerProperties.BEAN_VALIDATION_MODE, BeanValidationMode.NONE).resolver());

    register(MoxyXmlFeature.class);
    register(RolesAllowedDynamicFeature.class);
}
}
```

Como puede ver, en la clase anterior, proporcionamos todos los detalles necesarios para usar la interfaz de usuario swagger como Host en el que se alojará este proyecto swagger junto con la función de establecer la ruta base de su elección y todos los protocolos http compatibles como " `http` " o " `https` " e incluso más detalles según su requerimiento. También hay una disposición para que Swagger conozca toda la ubicación de su REST WS que puede configurarse usando `setResourcePackage` junto con paquetes. Esta clase nos permite usar la interfaz de usuario de Swagger con nuestra personalización.

Ahora ingrese los 2 archivos anteriores en web.xml para usarlos después de implementar nuestra aplicación en el servidor, como a continuación:

```
<servlet>
    <servlet-name>jersey-servlet</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
    <init-param>
        <param-name>javax.ws.rs.Application</param-name>
        <param-value>your.package.SwaggerJaxrsConfig</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>jersey-servlet</servlet-name>
    <url-pattern>/rest/*</url-pattern>
</servlet-mapping>

<filter>
    <filter-name>ApiOriginFilter</filter-name>
    <filter-class>your.package.ApiOriginFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>ApiOriginFilter</filter-name>
    <url-pattern>/rest/*</url-pattern>
</filter-mapping>
```

Ahora, nos moveremos al código real donde usaremos las anotaciones provistas por Swagger:

Cree un Employee.java de frijoles como se muestra a continuación:

```
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

@ApiModel("Employee bean")
public class Employee {

    private String name;
    private String id;
    private String dept;

    @ApiModelProperty(value = "Name of employee", example = "Test Employee")
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @ApiModelProperty(value = "Id of employee", example = "123456")
    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    @ApiModelProperty(value = "Department of employee", example = "IT Division",
allowableValues = "IT, Sales, Admin")
    public String getDept() {
        return dept;
    }

    public void setDept(String dept) {
        this.dept = dept;
    }
}
```

Los componentes de Swagger que usamos aquí son:

1. `@ApiModel("Employee bean")` : esto decidirá el nombre de la clase Bean que se debe mostrar en la interfaz de usuario de Swagger.
2. `@ApiModelProperty(value ="ABC", example="DeptName")` - Esto proporcionará información de cada campo utilizado en el bean. el valor proporciona una descripción del campo y el ejemplo proporciona un valor de muestra de ese campo.

Ahora crearemos un Controlador REST de la siguiente manera para crear un servicio web GET de la siguiente manera:

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.GenericEntity;
import javax.ws.rs.core.MediaType;
```

```
import javax.ws.rs.core.Response;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.swagger.ws.bean.Employee;
import org.swagger.ws.service.EmployeeService;

import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiResponse;
import io.swagger.annotations.ApiResponses;

@Path("/employee")
@Api(tags = {"Employee"})
public class EmployeeController {
    private static final Logger LOGGER = LoggerFactory.getLogger(EmployeeController.class);

@GET
@Produces(MediaType.APPLICATION_JSON + ";charset=utf-8")
@ApiOperation(produces="application/json", value = "Fetch employee details", httpMethod="GET",
notes = "<br>This service fetches Employee details", response = Employee.class)
@ApiResponses(value = { @ApiResponse(code = 200,response = Employee.class, message =
"Successful operation"),@ApiResponse(code = 400, message = "Bad Request", response =
Error.class), @ApiResponse(code = 422, message = "Invalid data", response = Error.class),
@ApiResponse(code = 500, message = "Internal Server Error", response = Error.class) })
public Response getEmployee()  {
        EmployeeService employeeService = new EmployeeService();
        Employee empDetails = employeeService.getEmployee();
        Response response;
        LOGGER.debug("Fetching employee");
        GenericEntity<Employee> entity = new GenericEntity<Employee>(empDetails){};
        response = Response.status(200).entity(entity).build();
        return response;
    }
}
```

Los componentes de Swagger que usamos aquí son:

1. `@Api(tags = {"Employee"})` : esta anotación le dirá a Swagger cuál debe ser el título del servicio web. En este caso el título es " `Employee` ". Tenga en cuenta que algunos estándares al escribir REST WS también son aplicables al escribir Documentación Swagger como el título no debería ser como " `GetEmployee` " o " `DeleteEmployee` ", etc.
2. `@ApiOperation(produces="application/json", value = "Fetch employee details", httpMethod="GET", notes = "<br>This service fetches Employee details", response = Employee.class)` : esta anotación proporciona una breve idea acerca de su servicio web.
    - `produces` describe el formato de respuesta,
    - `value` describe breve idea sobre el servicio web
    - `notes` describen información detallada sobre este servicio web en caso de que existan.
3. `@ApiResponses(value = { @ApiResponse(code = 200,response = Employee.class, message = "Successful operation"),@ApiResponse(code = 400, message = "Bad Request", response = Error.class), @ApiResponse(code = 422, message = "Invalid data", response = Error.class), @ApiResponse(code = 500, message = "Internal Server Error", response = Error.class) })` - Estas anotaciones nos proporcionan una forma de maneja diferentes tipos de códigos de estado `HTTP` que pueden recibirse como respuesta mientras consumes este servicio web. Nos permite establecer el código de error, el mensaje personalizado en su contra e incluso nos permite detectar ese error en una clase de error separada, si es necesario.

Finalmente, tenemos que crear una clase de servicio real que obtenga los detalles de los empleados cuando el servicio web sea consumido por el cliente. Puede implementarlo según sea necesario. A continuación se muestra el servicio de muestra para el propósito de demostración:

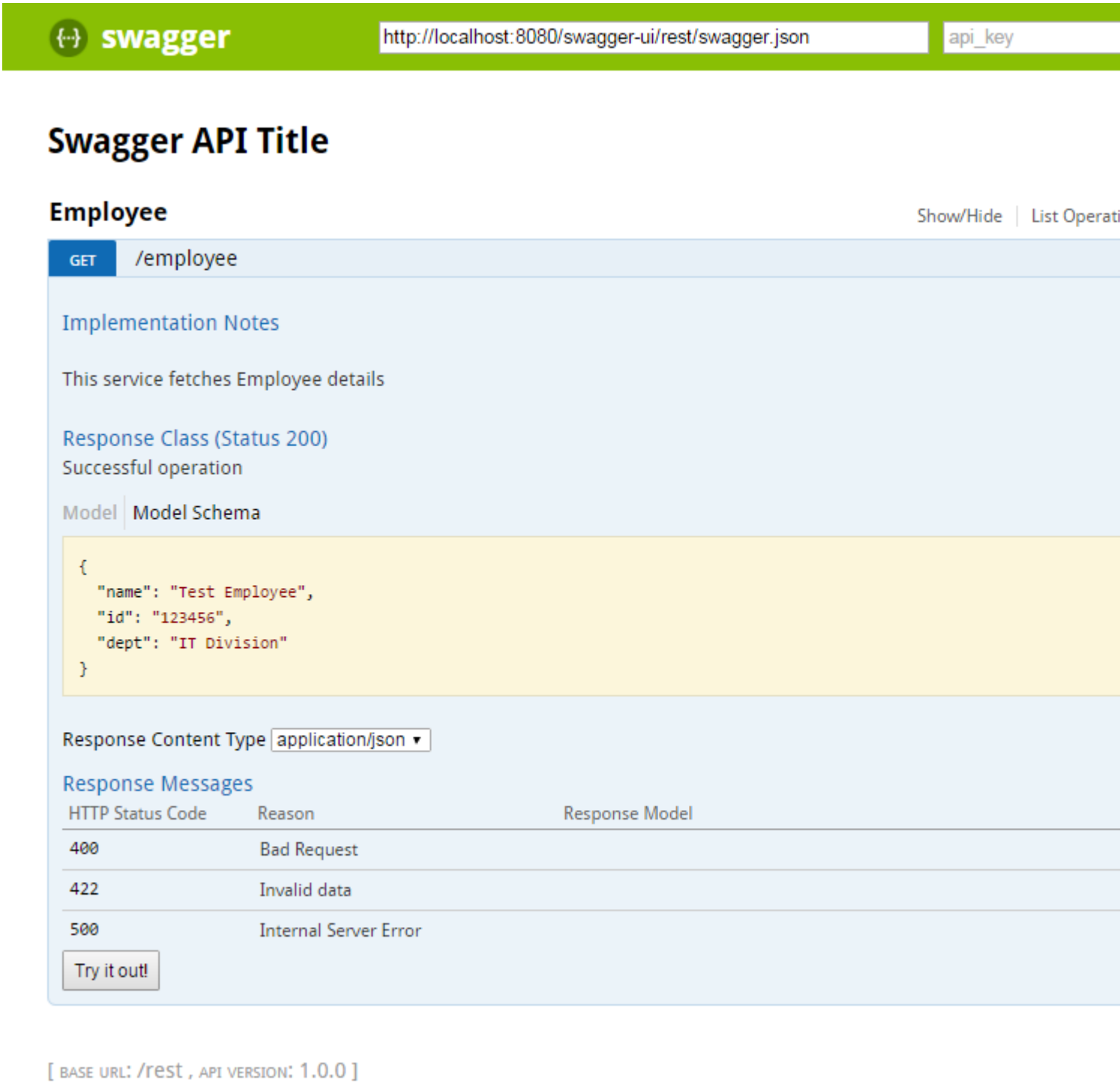
```
public class EmployeeService {

public Employee getEmployee() {
    Employee employee = new Employee();
    employee.setId("1");
    employee.setName("Test");
    employee.setDept("IT");
    return employee;
    }
}
```

Ahora está listo para ver su documentación en la interfaz de usuario de Swagger. Despliegue su proyecto Swagger e inicie el servidor. Ahora ve al navegador y presiona la URL de tu proyecto. En este caso su

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

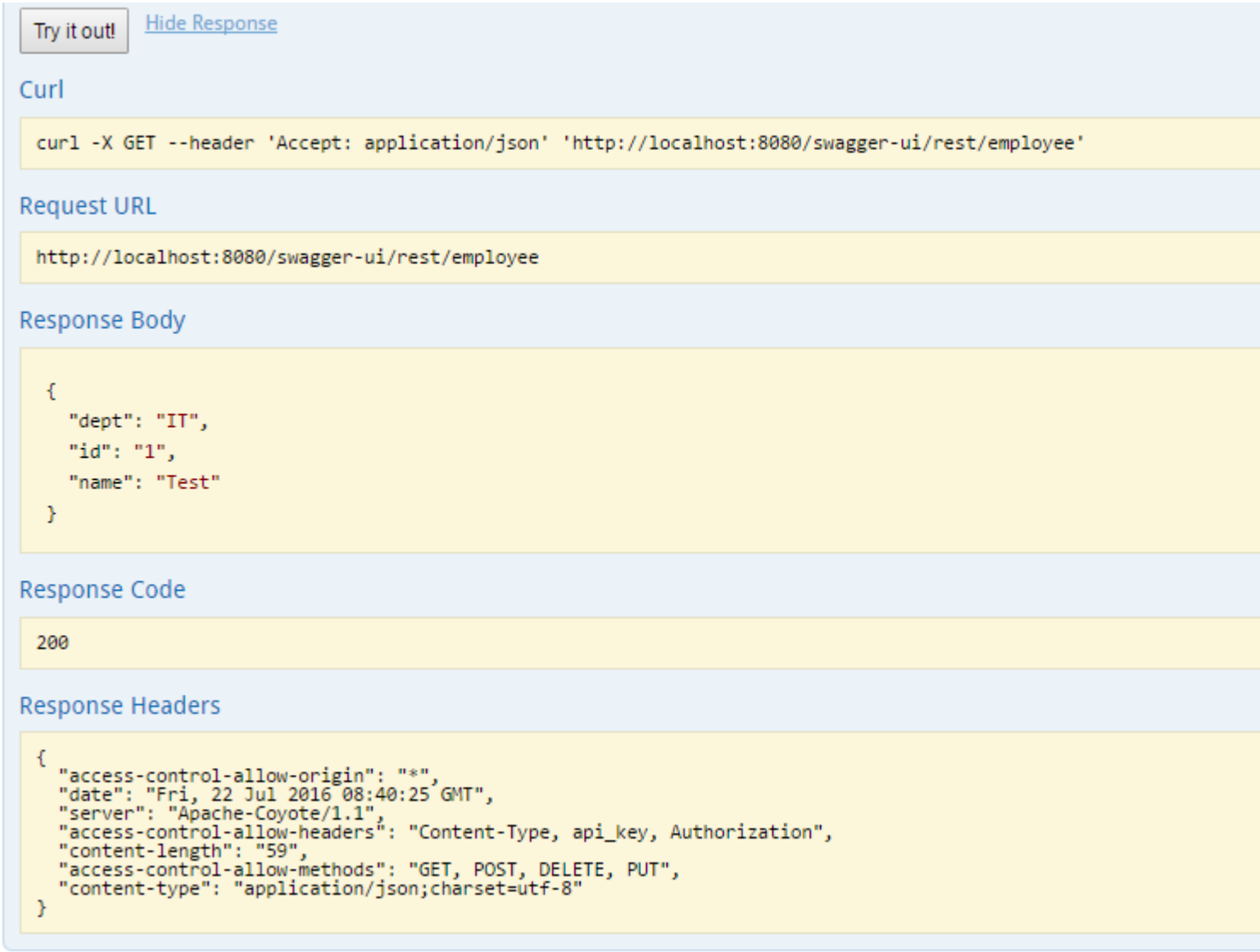
Si swagger está correctamente configurado, debería poder ver la siguiente pantalla:

Ahora, para ver la documentación de su servicio web, proporcione la "base_url / rest / swagger.json" de su proyecto en el cuadro de texto (http://exampl.com/api) que puede ver en la imagen de arriba. Después de proporcionar esa URL en el cuadro de texto, puede ver la siguiente pantalla con la documentación de su servicio web REST:



El cuadro de texto donde se escribe la " `api key` " en la imagen de arriba, puede proporcionar un encabezado específico o un valor clave si su proyecto lo exige como autenticación basada en la identificación del usuario, etc. Para eso, también deberá realizar cambios en index.html bajo la etiqueta que comienza con `<input placeholder="api_key" ......`

Otro beneficio adicional de Swagger-UI es que proporciona un botón "¡ `Try it out!` ". Este botón le permite habilitar para ver cuál podría ser la respuesta de este servicio web. Para este caso, si hacemos clic en ese botón, daremos la siguiente respuesta en la pantalla:



Así que esta es la demostración de muestra para Swagger UI. Hay muchas opciones que aún puede descubrir mientras escribe el controlador REST que lo ayudará a nivel de documentación al usar más anotaciones y sus atributos.

Lea swagger-ui en línea: https://riptutorial.com/es/swagger/topic/6686/swagger-ui

Creditos

S. No	Capítulos	Contributors
1	Empezando con swagger	Community, Daniel Käfer, gonephishing
2	springfox	mika
3	swagger-ui	Suyash