Setting Up Swagger 2 with a Spring REST API

Last modified: November 1, 2017

1. Overview

When creating a REST API, good documentation is instrumental.

Moreover, every change in the API should be simultaneously described in the reference documentation. Accomplishing this manually is a tedious exercise, so automation of the process was inevitable.

In this tutorial, we will look at **Swagger 2 for a Spring REST web service.** For this article, we will use the **Springfox** implementation of the Swagger 2 specification.

If you are not familiar with Swagger, you should visit its web page (http://swagger.io/) to learn more before continuing with this article.

2. Target Project

The creation of the REST service we will use in our examples is not within the scope of this article. If you already have a suitable project, use it. If not, the following links are a good place to start:

- **Build a REST API with Spring 4 and Java Config article (/2011/10/25/building-a-restful-web-service-with-spring-3-1-and-java-based-configuration-part-2/)**
- **Building a RESTful Web Service (https://spring.io/guides/gs/rest-service/)**

3. Adding the Maven Dependency

As mentioned above, we will use the Springfox implementation of the Swagger specification.

To add it to our Maven project, we need a dependency in the *pom.xml* file.

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.7.0</version>
</dependency>
```

4. Integrating Swagger 2 into the Project

4.1. Java Configuration

The configuration of Swagger mainly centers around the ***Docket*** bean.

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
          .select()
          .apis(RequestHandlerSelectors.any())
          .paths(PathSelectors.any())
          .build();
    }
}
```

Swagger 2 is enabled through the ***@EnableSwagger2*** annotation.

After the *Docket* bean is defined, its ***select()*** method returns an instance of ***ApiSelectorBuilder***, which provides a way to control the endpoints exposed by Swagger.

Predicates for selection of *RequestHandler*s can be configured with the help of ***RequestHandlerSelectors*** and ***PathSelectors***. Using *any()* for both will make documentation for your entire API available through Swagger.

This configuration is enough to integrate Swagger 2 into existing Spring Boot project. For other Spring projects, some additional tuning is required.

4.2. Configuration Without Spring Boot

Without Spring Boot, you don't have the luxury of auto-configuration of your resource handlers. Swagger UI adds a set of resources which you must configure as part of a class that extends *WebMvcConfigurerAdapter,* and is annotated with *@EnableWebMvc.*

```
@Override
public void addResourceHandlers(ResourceHandlerRegistry registry) {
    registry.addResourceHandler("swagger-ui.html")
      .addResourceLocations("classpath:/META-INF/resources/");

    registry.addResourceHandler("/webjars/**")
      .addResourceLocations("classpath:/META-INF/resources/webjars/");
}
```

4.3. Verification

To verify that Springfox is working, you can visit the following URL in your browser:

http://localhost:8080/spring-security-rest/api/v2/api-docs

The result is a JSON response with a large number of key-value pairs, which is not very human-readable. Fortunately, Swagger provides **Swagger UI** for this purpose.

5. Swagger UI

Swagger UI is a built-in solution which makes user interaction with the Swagger-generated API documentation much easier.

5.1. Enabling Springfox's Swagger UI

To use Swagger UI, one additional Maven dependency is required:

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.7.0</version>
</dependency>
```

Now you can test it in your browser by visiting *http://localhost:8080/your-app-root/swagger-ui.html*

In our case, by the way, the exact URL will be: *http://localhost:8080/spring-security-rest/api/swagger-ui.html*

The result should look something like this:

(/wp-content/uploads/2016/07/Screenshot_11.png)

5.2. Exploring Swagger Documentation

Within Swagger's response is a **list of all controllers** defined in your application. Clicking on any of them will list the valid HTTP methods (*DELETE*, *GET*, *HEAD*, *OPTIONS*, *PATCH*, *POST*, *PUT*).

Expanding each method provides additional useful data, such as response status, content-type, and a list of parameters. It is also possible to try each method using the UI.

Swagger's ability to be synchronized with your code base is crucial. To demonstrate this, you can add a new controller to your application.

```
@RestController
public class CustomController {

    @RequestMapping(value = "/custom", method = RequestMethod.POST)
    public String custom() {
        return "custom";
    }
}
```

Now, if you refresh the Swagger documentation, you will see **custom-controller** in the list of controllers. As you know, there is only one method (*POST*) shown in Swagger's response.

6. Advanced Configuration

The *Docket* bean of your application can be configured to give you more control over the API documentation generation process.

6.1. Filtering API for Swagger's Response

It is not always desirable to expose the documentation for your entire API. You can restrict Swagger's response by passing parameters to the ***apis()*** and ***paths()*** methods of the *Docket* class.

As seen above, *RequestHandlerSelectors* allows using the *any* or *none* predicates, but can also be used to filter the API according to the base package, class annotation, and method annotations.

PathSelectors provides additional filtering with predicates which scan the request paths of your application. You can use *any()*, *none(), regex()*, or *ant()*.

In the example below, we will instruct Swagger to include only controllers from a particular package, with specific paths, using the *ant()* predicate.

```
@Bean
public Docket api() {
    return new Docket(DocumentationType.SWAGGER_2)
      .select()
      .apis(RequestHandlerSelectors.basePackage("org.baeldung.web.controller"))
      .paths(PathSelectors.ant("/foos/*"))
      .build();
}
```

6.2. Custom Information

Swagger also provides some default values in its response which you can customize, such as "Api Documentation", "Created by Contact Email", "Apache 2.0".

To change these values, you can use the ***apiInfo(ApiInfo apiInfo)*** method. The ***ApiInfo*** class that contains custom information about the API.

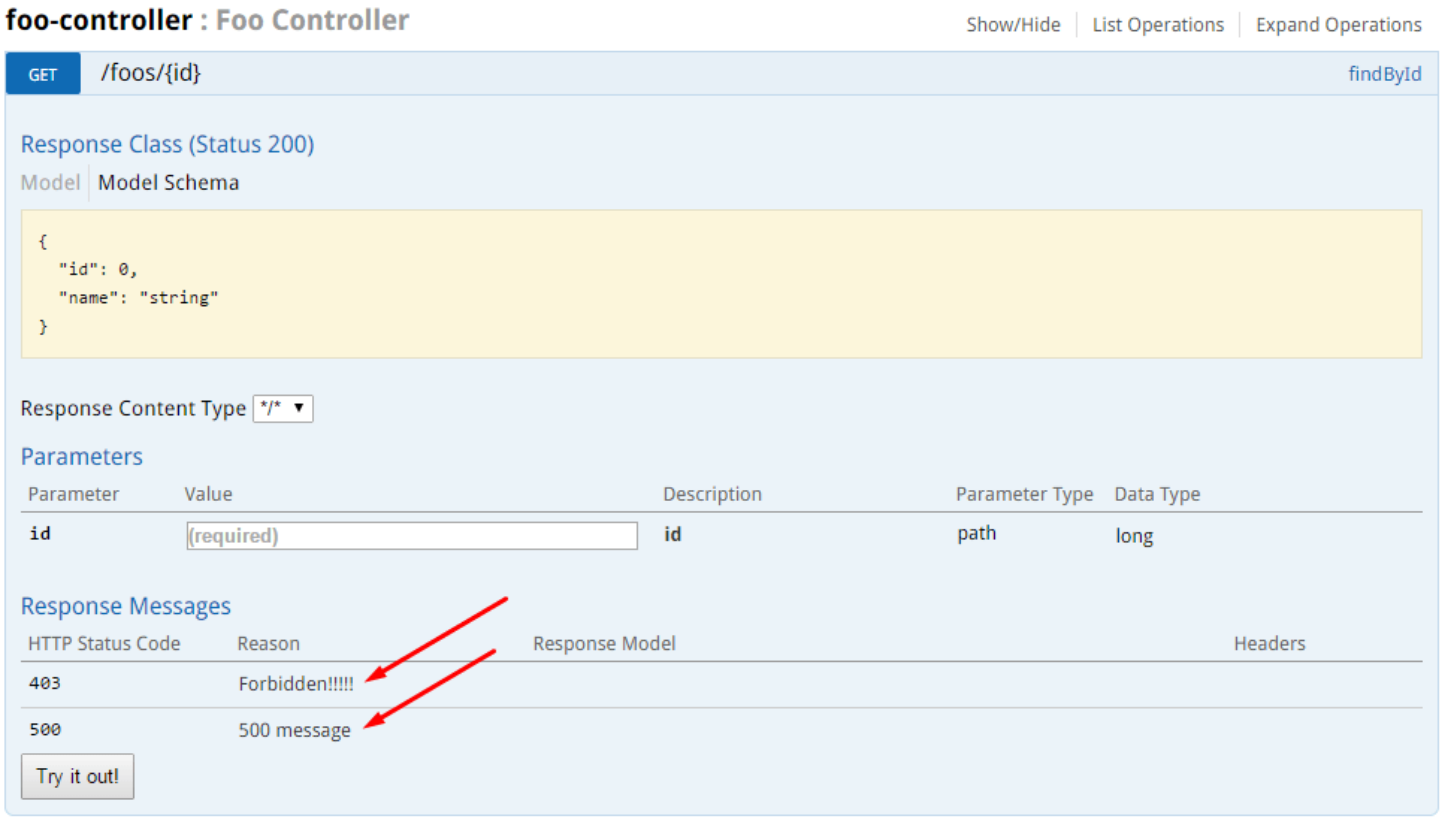
```
@Bean
public Docket api() {
    return new Docket(DocumentationType.SWAGGER_2)
      .select()
      .apis(RequestHandlerSelectors.basePackage("com.example.controller"))
      .paths(PathSelectors.ant("/foos/*"))
      .build()
      .apiInfo(apiInfo());
}

private ApiInfo apiInfo() {
     return new ApiInfo(
       "My REST API",
       "Some custom description of API.",
       "API TOS",
       "Terms of service",
       new Contact("John Doe", "www.example.com", "myeaddress@company.com"),
       "License of API", "API license URL", Collections.emptyList());
}
```

6.3. Custom Methods Response Messages

Swagger allows **globally overriding response messages of HTTP methods** through *Docket*'s ***globalResponseMessage()*** method. First, you must instruct Swagger not to use default response messages.

Suppose you wish to override **500** and **403** response messages for all *GET* methods. To achieve this, some code must be added to the *Docket*'s initialization block (original code is excluded for clarity):

```
.useDefaultResponseMessages(false)
.globalResponseMessage(RequestMethod.GET,
  newArrayList(new ResponseMessageBuilder()
    .code(500)
    .message("500 message")
    .responseModel(new ModelRef("Error"))
    .build(),
    new ResponseMessageBuilder()
      .code(403)
      .message("Forbidden!")
      .build()));
```



(/wp-content/uploads/2016/07/Screenshot_2.png)

7. Conclusion

In this tutorial, we set up Swagger 2 to generate documentation for a Spring REST API. We also have explored ways to visualize and customize Swagger's output.

The **full implementation** of this tutorial can be found in the Github project (https://github.com/eugenp/tutorials/tree/master/spring-security-rest) – this is an Eclipse based project, so it should be easy to import and run as it is.

And, if you're a student of REST With Spring (http://www.baeldung.com/rest-with-spring-course#master-class), go to Lesson 1 from Module 7 for a deep-dive into setting up Swagger with Spring and Spring Boot.