(/)

Setting Up Swagger 2 with a Spring REST API

Last modified: July 27, 2019

by baeldung (https://www.baeldung.com/author/baeldung/)

REST (https://www.baeldung.com/category/rest/)
Security (https://www.baeldung.com/category/security-2/)
Spring (https://www.baeldung.com/category/spring/) +

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.

Further reading:

Generate Spring Boot REST Client with Swagger (https://www.baeldung.com/spring-boot-rest-client-swagger-codegen)

Learn how you can generate a Spring Boot REST client using Swagger Code generator.

Read more (https://www.baeldung.com/spring-boot-rest-client-swagger-codegen) →

Introduction to Spring REST Docs (https://www.baeldung.com/spring-rest-docs)

This article introduces Spring REST Docs, a test-driven mechanism to generate documentation for RESTful services that is both accurate and readable.

Read more (https://www.baeldung.com/spring-rest-docs) →

Introduction to Asciidoctor in Java (https://www.baeldung.com/asciidoctor)

Learn how to generate documents using AsciiDoctor.

Read more (https://www.baeldung.com/asciidoctor) →

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 (/building-a-restful-web-service-with-spring-and-java-based-configuration)**

- **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. The latest version can be found on Maven Central (https://search.maven.org/classic/#search%7Cga%7C1%7C%22Springfox%20Swagger2%22).

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>2.9.2</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 an 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>2.9.2</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.

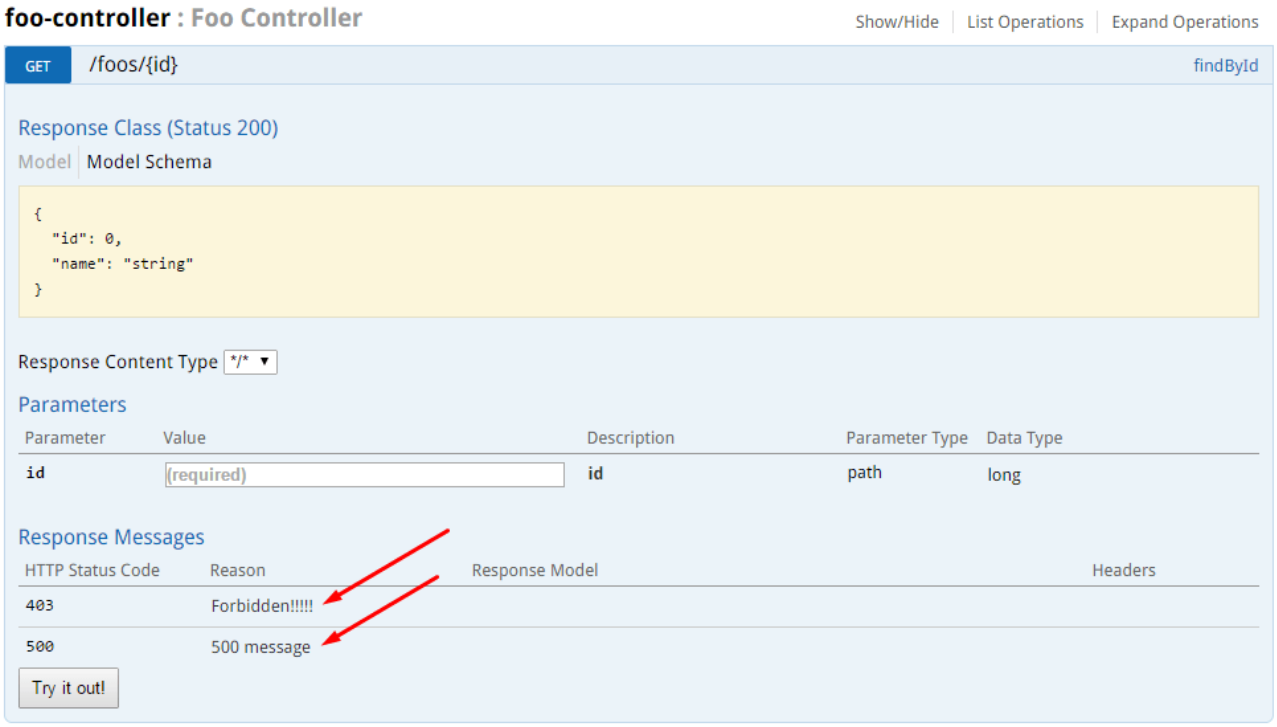
```
@Bean
public Docket api() {
    return new Docket(DocumentationType.SWAGGER_2)
      .select()
      .apis(RequestHandlerSelectors.basePackage("com.example.controller"))
      .paths(PathSelectors.ant("/foos/*"))
      .build()
      .apiInfo(apiInfo());
}

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. Swagger UI with an OAuth-secured API

The Swagger UI provides a number of very useful features – which we've covered well so far here. But we can't really use most of these if our API is secured and not accessible.

Let's see how we can allow Swagger to access an OAuth-secured API – using the Authorization Code grant type in this example.

We'll configure Swagger to access our secured API using the *SecurityScheme* and *SecurityContext* support:

```
@Bean
public Docket api() {
    return new Docket(DocumentationType.SWAGGER_2).select()
        .apis(RequestHandlerSelectors.any())
        .paths(PathSelectors.any())
        .build()
        .securitySchemes(Arrays.asList(securityScheme()))
        .securityContexts(Arrays.asList(securityContext()));
}
```

7.1. Security Configuration

We'll define a *SecurityConfiguration* bean in our Swagger configuration – and set some defaults:

```
@Bean
public SecurityConfiguration security() {
    return SecurityConfigurationBuilder.builder()
        .clientId(CLIENT_ID)
        .clientSecret(CLIENT_SECRET)
        .scopeSeparator(" ")
        .useBasicAuthenticationWithAccessCodeGrant(true)
        .build();
}
```

7.2. *SecurityScheme*

Next, we'll define our *SecurityScheme*; this is used to describe how our API is secured (Basic Authentication, OAuth2, …).

In our case here, we'll define an OAuth scheme used to secure our Resource Server (/rest-api-spring-oauth2-angularjs):


```
private SecurityScheme securityScheme() {
    GrantType grantType = new AuthorizationCodeGrantBuilder()
        .tokenEndpoint(new TokenEndpoint(AUTH_SERVER + "/token", "oauthtoken"))
        .tokenRequestEndpoint(
          new TokenRequestEndpoint(AUTH_SERVER + "/authorize", CLIENT_ID, CLIENT_SECRET))
        .build();

    SecurityScheme oauth = new OAuthBuilder().name("spring_oauth")
        .grantTypes(Arrays.asList(grantType))
        .scopes(Arrays.asList(scopes()))
        .build();
    return oauth;
}
```

Note that we used the Authorization Code grant type – for which we need to provide a token endpoint and the authorization URL of our OAuth2 Authorization Server.

And here are the scopes we need to have defined:

```
private AuthorizationScope[] scopes() {
    AuthorizationScope[] scopes = {
      new AuthorizationScope("read", "for read operations"),
      new AuthorizationScope("write", "for write operations"),
      new AuthorizationScope("foo", "Access foo API") };
    return scopes;
}
```

These sync up with the scopes we actually have defined in our application, for the */foos* API.

7.3. Security Context

Finally, we need to define a security context for our example API:

```
private SecurityContext securityContext() {
    return SecurityContext.builder()
      .securityReferences(
        Arrays.asList(new SecurityReference("spring_oauth", scopes())))
      .forPaths(PathSelectors.regex("/foos.*"))
      .build();
}
```

Note how the name we used here, in the reference – *spring_oauth* – syncs up with the name we used previously, in the *SecurityScheme*.

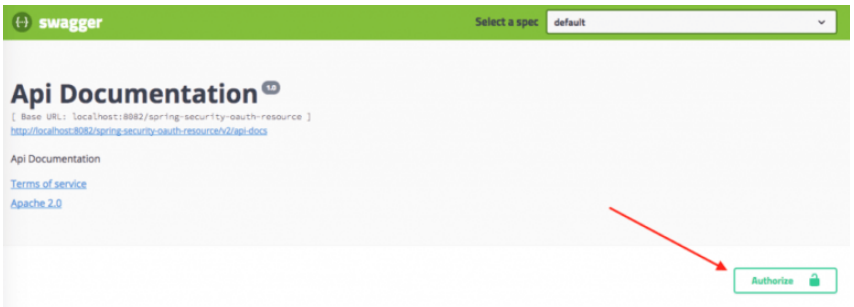
7.4. Test

Alright, now that we have everything set up and ready to go, let's take a look at our Swagger UI and try access the Foo API:

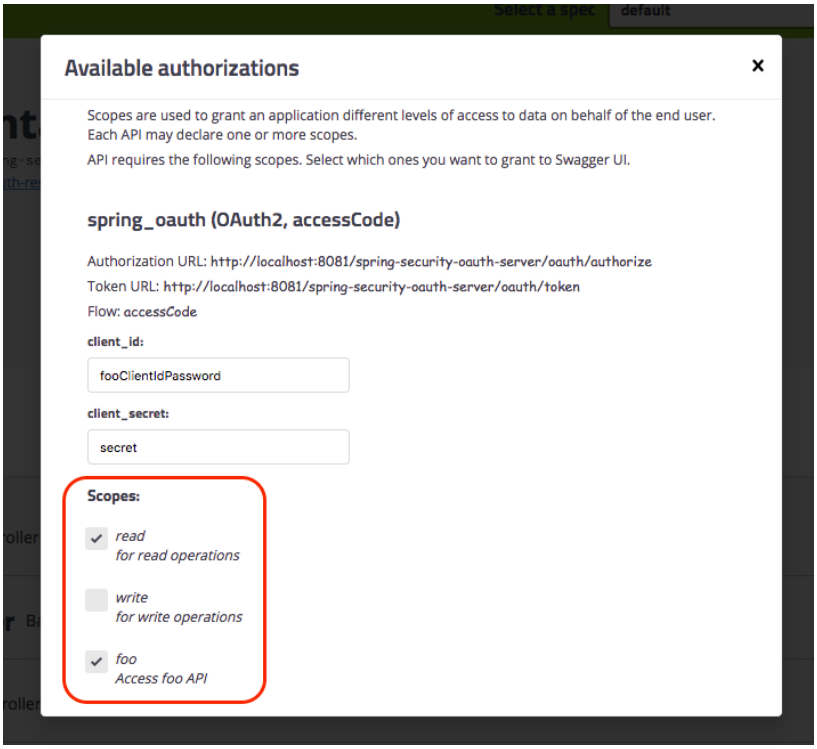
We can access the Swagger UI locally:

```
http://localhost:8082/spring-security-oauth-resource/swagger-ui.html
```

As we can see a new button "Authorize" now exist due to our security configurations:



(/wp-content/uploads/2015/12/swagger_1-1024x362.png)
When we click the "Authorize" button we can see the following pop-up – to authorize our Swagger UI to access the secured API:

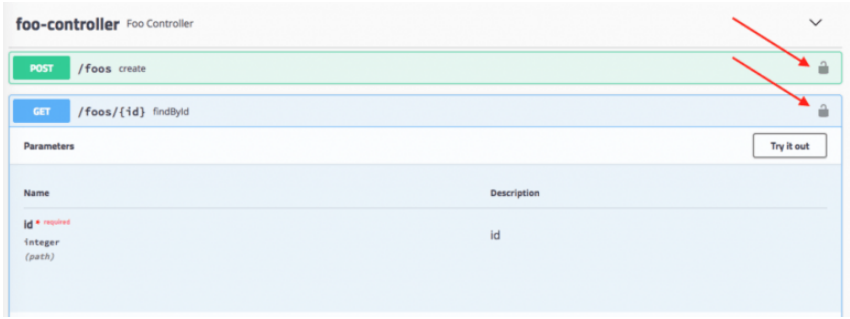


(/wp-content/uploads/2015/12/swagger_2.png)

Note that:

- We can already see the CLIENT_ID and CLIENT_SECRET – as we've pre-configured them earlier (but we can still change them)
- We can now select the scopes we need

Here's how the secured APIs is marked:



(/wp-content/uploads/2015/12/swagger_3-1024x378.png)

And now, finally, we can hit our API!

Of course, it almost goes without saying that we need to be careful how we expose Swagger UI externally, now that this security configuration is active.

8. 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. And finally, we looked at a simple OAuth configuration for Swagger.

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. To see the set up in a Boot project, check out this GitHub module (https://github.com/eugenp/tutorials/tree/master/spring-boot-mvc).

For the OAuth section, the code is available in our spring-security-oauth (https://github.com/Baeldung/spring-security-oauth) repository.

And, if you're a student of REST With Spring (/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.

▲ newest ▲ oldest ▲ most voted

Guest

Muhammad Yousaf Sajjad

Can you share the code on github?

+ 0 −

3 years ago

Guest

Eugen Paraschiv (https://www.baeldung.com/)

I just added the link at the very end of the article. Cheers,
Eugen.

+ 3 −

3 years ago

Guest

Daniel TwumTutorial also worked for me.

Thanks for a nice writeup.

+ 0 −

3 years ago

Guest

nelaturuk

Hi, I am not using spring boot. I am not able to add the addResourceHandler method. I am using Spring 4.0 and in the websecurityadapter I have added EnableWebSecurity. Even if i make it EnableWebMvc doesnt seem to work for overrriding the method. Any suggestion for this?

+ 0 −

3 years ago

Guest

Eugen Paraschiv (https://www.baeldung.com/)

The addResourceHandler method is not Boot specific. Actually, section 4.2 covers the extra config you'll need to do when not using Boot.
And, generally speaking – there's no reason you should need Boot to set up Swagger. Hope that helps.
Cheers,
Eugen.

+ 0 −

3 years ago

Guest

nelaturuk

Thanks Eugen. I added the Swagger Config in the WebMVCSecurityAdapater as metioned by you and it worked!

+ 1 −

3 years ago

Guest

Eugen Paraschiv (https://www.baeldung.com/)

Sounds good, glad everything worked out. Cheers,
Eugen.

+ 1 −

3 years ago

Guest

J Walls

Just starting working on integrating Swagger on my current project. This tutorial was super helpful and well put together, I was able to easily work through initial setup and then more refined tweaks using this guide. Bravo.

+ 1 −

3 years ago

Guest

Eugen Paraschiv (https://www.baeldung.com/)

Glad you found it helpful J. Cheers,
Eugen.

+ 3 −

3 years ago

Guest

ladyNi

Hi Eugen,

Can u possibly explain how to add a header parameter to be passed with every request made.
I actually have to send a CSRF token for every request other than GET.
I tried using their APIKey, but not able to get it working.

+ 1 −

3 years ago

Guest

Eugen Paraschiv (https://www.baeldung.com/)

The CSRF token does indeed make some other aspects more difficult to work with, and Swagger can be one of those aspects. I haven't looked into the option of integrating Swagger with Spring Security so that you can then have the CSRF token available in Swagger. It may be possible, I just haven't looked into it. Cheers,
Eugen.

+ 1 −

3 years ago

Guest

jothi manickam

Hi I got the response message for user input request but did not work for parameter..please guide me to achieve this … need to set any annotation for bean class

+ 1 −

3 years ago

Guest

Eugen Paraschiv (https://www.baeldung.com/)

Well Jothy – I'd have to look at code to know what's happening there. Go ahead and open an issue over on Github with the details and I'd be happy to have a look. Cheers,
Eugen.

+ 0 −

3 years ago

Load More Comments

Comments are closed on this article!

CATEGORIES

SPRING (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
HTTP CLIENT-SIDE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)

ABOUT