Swagger:

Swagger 2 is an open-source project used to describe and document RESTful APIs. Swagger 2 is language-agnostic and is extensible into new technologies and protocols beyond HTTP. The current version defines a set HTML, JavaScript, and CSS assets to dynamically generate documentation from a Swagger-compliant API. These files are bundled by the [Swagger UI](http://swagger.io/swagger-ui/) project to display the API on the browser.

We will be using Springfox in our project.

To bring it in, we need the following dependency declaration in our Maven POM.

1

. . .

2

​

3

<dependency>

4

<groupId>io.springfox</groupId>

5

<artifactId>springfox-swagger2</artifactId>

6

<version>2.6.1</version>

7

<scope>compile</scope>

8

</dependency>

9

​

10

. . .

In addition to Sprinfox, we also require Swagger UI. The code to include Swagger UI is this.

1

. . .

2

​

3

<dependency>

4

<groupId>io.springfox</groupId>

5

<artifactId>springfox-swagger-ui</artifactId>

6

<version>2.6.1</version>

7

<scope>compile</scope>

8

</dependency>

9

​

10

. . .

## The Spring Boot RESTful Application

Our application implements a set of REST endpoints to manage products. We have a Product JPA entity and a repository named ProductRepository that extends CrudRepository to perform CRUD operations on products against an in-memory H2 database.

The service layer is composed of a ProductService interface and a ProductServiceImpl implementation class.

The Maven POM of the application is this.

pom.xml:

1

<?xml version="1.0" encoding="UTF-8"?>

2

<project

3

xmlns="http://maven.apache.org/POM/4.0.0"

4

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

5

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

6

<modelVersion>4.0.0</modelVersion>

7

<groupId>guru.springframework</groupId>

8

<artifactId>spring-boot-web</artifactId>

9

<version>0.0.1-SNAPSHOT</version>

10

<packaging>jar</packaging>

11

<name>Spring Boot Web Application</name>

12

<description>Spring Boot Web Application</description>

13

<parent>

14

<groupId>org.springframework.boot</groupId>

15

<artifactId>spring-boot-starter-parent</artifactId>

16

<version>1.4.2.RELEASE</version>

17

<relativePath/>

18

<!-- lookup parent from repository -->

19

</parent>

20

<properties>

21

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

22

<java.version>1.8</java.version>

23

</properties>

24

<dependencies>

25

<dependency>

26

<groupId>org.springframework.boot</groupId>

27

<artifactId>spring-boot-starter-data-rest</artifactId>

28

</dependency>

29

<dependency>

30

<groupId>org.springframework.boot</groupId>

31

<artifactId>spring-boot-starter-data-jpa</artifactId>

32

</dependency>

33

<dependency>

34

<groupId>org.springframework.boot</groupId>

35

<artifactId>spring-boot-starter-security</artifactId>

36

</dependency>

37

<dependency>

38

<groupId>org.springframework.boot</groupId>

39

<artifactId>spring-boot-starter-thymeleaf</artifactId>

40

</dependency>

41

<dependency>

42

<groupId>org.springframework.boot</groupId>

43

<artifactId>spring-boot-starter-web</artifactId>

44

</dependency>

45

<dependency>

46

<groupId>com.jayway.jsonpath</groupId>

47

<artifactId>json-path</artifactId>

48

<scope>test</scope>

49

</dependency>

50

<dependency>

51

<groupId>io.springfox</groupId>

52

<artifactId>springfox-swagger-ui</artifactId>

53

<version>2.6.1</version>

54

<scope>compile</scope>

55

</dependency>

56

<dependency>

57

<groupId>io.springfox</groupId>

58

<artifactId>springfox-swagger2</artifactId>

59

<version>2.6.1</version>

60

<scope>compile</scope>

61

</dependency>

62

<!--WebJars-->

63

<dependency>

64

<groupId>com.h2database</groupId>

65

<artifactId>h2</artifactId>

66

</dependency>

67

<dependency>

68

<groupId>org.springframework.boot</groupId>

69

<artifactId>spring-boot-starter-test</artifactId>

70

<scope>test</scope>

71

</dependency>

72

</dependencies>

73

<build>

74

<plugins>

75

<plugin>

76

<groupId>org.springframework.boot</groupId>

77

<artifactId>spring-boot-maven-plugin</artifactId>

78

</plugin>

79

</plugins>

80

</build>

81

</project>

The controller of the application, ProductController, defines the REST API endpoints. The code of ProductController is this:

1

. . .

2

@RestController

3

@RequestMapping("/product")

4

public class ProductController {

5

​

6

private ProductService productService;

7

​

8

@Autowired

9

public void setProductService(ProductService productService) {

10

this.productService = productService;

11

}

12

​

13

​

14

@RequestMapping(value = "/list", method= RequestMethod.GET)

15

public Iterable list(Model model){

16

Iterable productList = productService.listAllProducts();

17

return productList;

18

}

19

​

20

@RequestMapping(value = "/show/{id}", method= RequestMethod.GET)

21

public Product showProduct(@PathVariable Integer id, Model model){

22

Product product = productService.getProductById(id);

23

return product;

24

}

25

​

26

​

27

@RequestMapping(value = "/add", method = RequestMethod.POST)

28

public ResponseEntity saveProduct(@RequestBody Product product){

29

productService.saveProduct(product);

30

return new ResponseEntity("Product saved successfully", HttpStatus.OK);

31

}

32

​

33

​

34

@RequestMapping(value = "/update/{id}", method = RequestMethod.PUT)

35

public ResponseEntity updateProduct(@PathVariable Integer id, @RequestBody Product product){

36

Product storedProduct = productService.getProductById(id);

37

storedProduct.setDescription(product.getDescription());

38

storedProduct.setImageUrl(product.getImageUrl());

39

storedProduct.setPrice(product.getPrice());

40

productService.saveProduct(storedProduct);

41

return new ResponseEntity("Product updated successfully", HttpStatus.OK);

42

}

43

​

44

​

45

@RequestMapping(value="/delete/{id}", method = RequestMethod.DELETE)

46

public ResponseEntity delete(@PathVariable Integer id){

47

productService.deleteProduct(id);

48

return new ResponseEntity("Product deleted successfully", HttpStatus.OK);

49

​

50

}

51

​

52

}

53

. . .

In this controller, the @RestController annotation introduced in Spring 4.0 marks ProductController as a REST API controller. Under the hood, @RestController works as a convenient annotation to annotate the class with the @Controller and @ResponseBody.

The @RequestMapping class-level annotation maps requests to /product onto the ProductController class. The method-level @RequestMapping annotations map web requests to the handler methods of the controller.

## Configuring Swagger 2 in the Application

For our application, we will create a Docket bean in a Spring Boot configuration to configure Swagger 2 for the application. A Springfox Docket instance provides the primary API configuration with sensible defaults and convenience methods for configuration. Our Spring Boot configuration class, SwaggerConfig is this.

1

. . .

2

@Configuration

3

@EnableSwagger2

4

public class SwaggerConfig {

5

@Bean

6

public Docket productApi() {

7

return new Docket(DocumentationType.SWAGGER\_2)

8

.select() .apis(RequestHandlerSelectors.basePackage("guru.springframework.controllers"))

9

.paths(regex("/product.\*"))

10

.build();

11

​

12

}

13

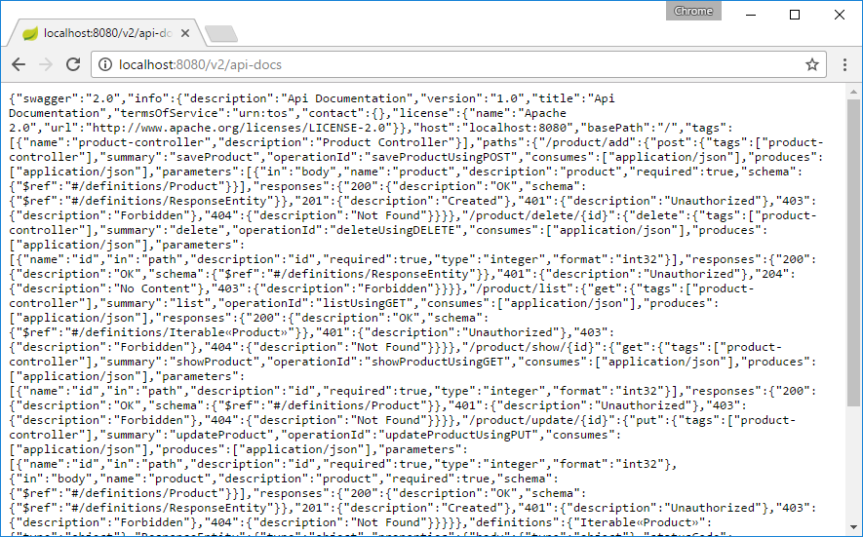
}

14

. . .

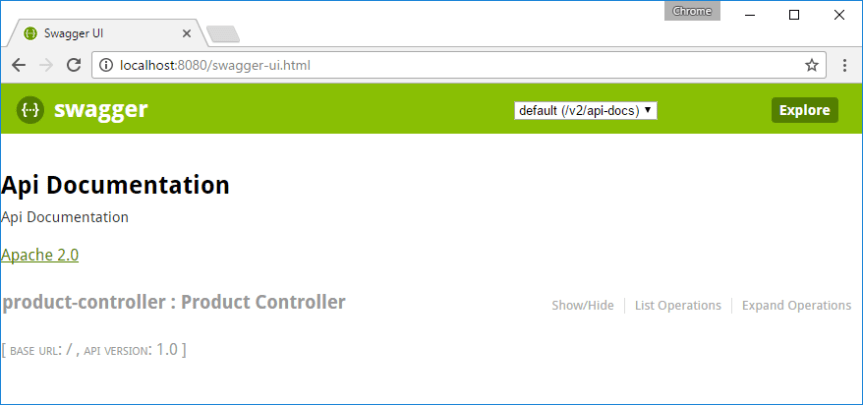
In this configuration class, the @EnableSwagger2 annotation enables Swagger support in the class. The select() method called on the Docket bean instance returns an ApiSelectorBuilder, which provides the apis() and paths() methods that are used to filter the controllers and methods that are being documented using String predicates.

In the code, the RequestHandlerSelectors.basePackage predicate matches the guru.springframework.controllers base package to filter the API. The regex parameter passed to paths()acts as an additional filter to generate documentation only for the path starting with /product.

At this point, you should be able to test the configuration by starting the app and pointing your browser to http://localhost:8080/v2/api-docs.  
[](https://i1.wp.com/springframework.guru/wp-content/uploads/2017/02/Swagger_JSON_Output.png?ssl=1)Obviously, the above JSON dump that Swagger 2 generates for our endpoints is not something we want.

What we want is some nice human readable structured documentation, and this is where Swagger UI takes over.

On pointing your browser to http://localhost:8080/swagger-ui.html, you will see the generated documentation rendered by Swagger UI, like this:

[](https://i2.wp.com/springframework.guru/wp-content/uploads/2017/02/Swagger_Default_Documentation.png?ssl=1)

As you can see, Swagger 2 used sensible defaults to generate the documentation of our ProductController.

Then, Swagger UI wrapped everything up to provide us an intuitive UI. This was all done automatically. We did not write any code or other documentation to support Swagger.

## Customizing Swagger

So far, we’ve been looking at Swagger documentation as it comes out of the box — but Swagger 2 has some great customization options.

Let’s start customizing Swagger by providing information about our API in the SwaggerConfig class like this.

SwaggerConfig.java:

1

package guru.springframework.config;

2

​

3

import org.springframework.context.annotation.Bean;

4

import org.springframework.context.annotation.Configuration;

5

import springfox.documentation.builders.RequestHandlerSelectors;

6

import springfox.documentation.service.ApiInfo;

7

import springfox.documentation.service.Contact;

8

import springfox.documentation.spi.DocumentationType;

9

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

10

import springfox.documentation.swagger2.annotations.EnableSwagger2;

11

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

12

​

13

@Configuration

14

@EnableSwagger2

15

public class SwaggerConfig {

16

@Bean

17

public Docket productApi() {

18

return new Docket(DocumentationType.SWAGGER\_2)

19

.select()

20

.apis(RequestHandlerSelectors.basePackage("guru.springframework.controllers"))

21

.paths(regex("/product.\*"))

22

.build()

23

.apiInfo(metaData());

24

}

25

private ApiInfo metaData() {

26

ApiInfo apiInfo = new ApiInfo(

27

"Spring Boot REST API",

28

"Spring Boot REST API for Online Store",

29

"1.0",

30

"Terms of service",

31

new Contact("John Thompson", "https://springframework.guru/about/", "john@springfrmework.guru"),

32

"Apache License Version 2.0",

33

"https://www.apache.org/licenses/LICENSE-2.0");

34

return apiInfo;

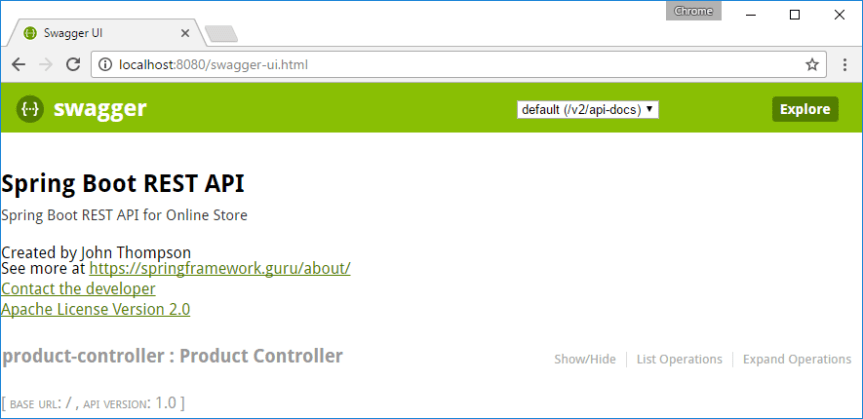
35

}

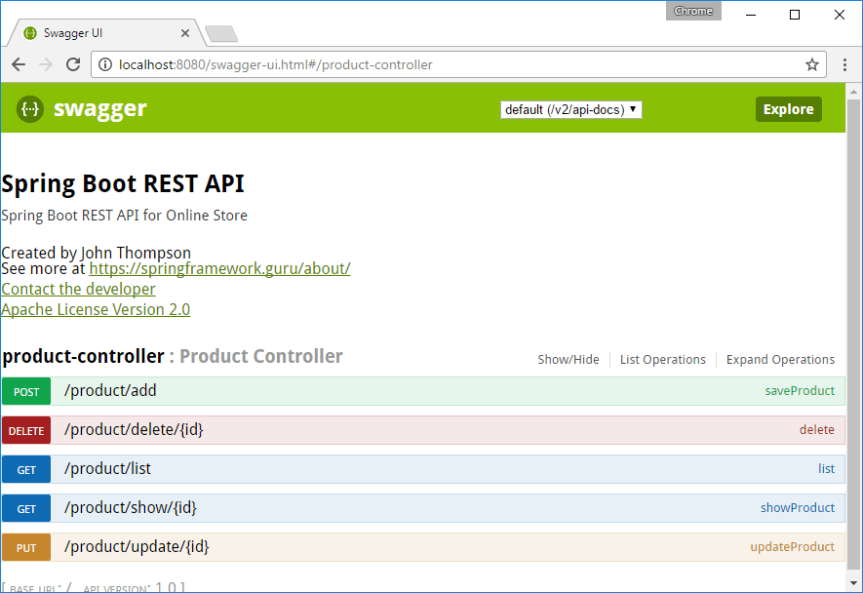
36

}

In the SwaggerConfig class, we have added a metaData() method that returns and ApiInfo object initialized with information about our API. Line 23 initializes the Docket with the new information.

The Swagger 2-generated documentation now looks similar to this:  
[](https://i1.wp.com/springframework.guru/wp-content/uploads/2017/02/swagger-ui_with_API_information-2.png?ssl=1)

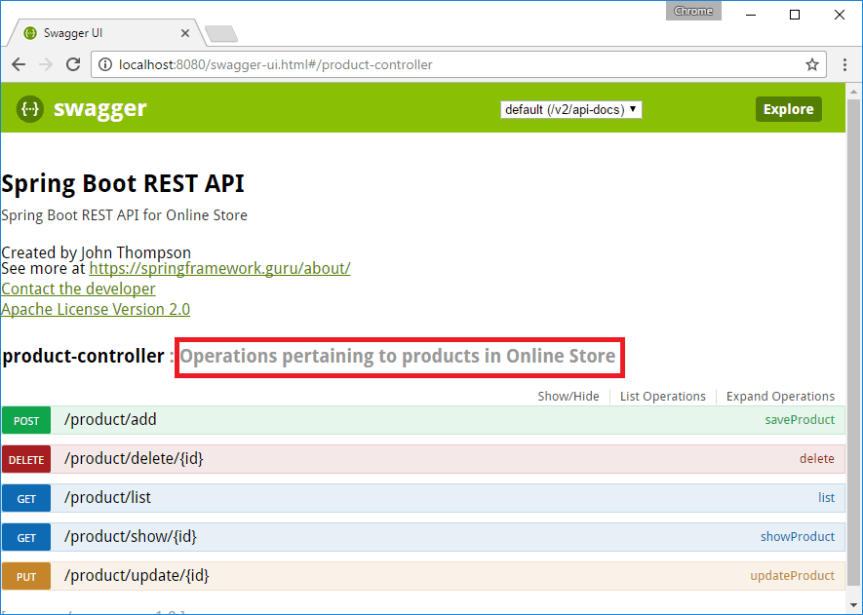
## Swagger 2 Annotations for REST Endpoints

At this point, if you click the product controller link, Swagger UI will display the documentation of our operation endpoints, like this:  
[](https://i1.wp.com/springframework.guru/wp-content/uploads/2017/02/swagger-ui_with_default_endpoint_documentation.png?ssl=1)

We can use the @Api annotation on our ProductController class to describe our API.

1

@RestController @RequestMapping("/product") @Api(value="onlinestore", description="Operations pertaining to products in Online Store") public class ProductController { . . . . }

The Swagger UI-generated documentation will reflect the description and now looks like this:  
[](https://i2.wp.com/springframework.guru/wp-content/uploads/2017/02/swagger-ui_with_API_Description.png?ssl=1)For each of our operation endpoints, we can use the @ApiOperation annotation to describe the endpoint and its response type, like this:

1

. . .

2

@ApiOperation(value = "View a list of available products", response = Iterable.class)

3

​

4

@RequestMapping(value = "/list", method= RequestMethod.GET,produces = "application/json")

5

public Iterable list(Model model){

6

Iterable productList = productService.listAllProducts();

7

return productList;

8

}

9

. . .

Swagger 2 also allows overriding the default response messages of HTTP methods. You can use the @ApiResponse annotation to document other responses, in addition to the regular HTTP 200 OK, like this.

1

. . .

2

@ApiOperation(value = "View a list of available products", response = Iterable.class)

3

@ApiResponses(value = {

4

@ApiResponse(code = 200, message = "Successfully retrieved list"),

5

@ApiResponse(code = 401, message = "You are not authorized to view the resource"),

6

@ApiResponse(code = 403, message = "Accessing the resource you were trying to reach is forbidden"),

7

@ApiResponse(code = 404, message = "The resource you were trying to reach is not found")

8

}

9

)

10

@RequestMapping(value = "/list", method= RequestMethod.GET, produces = "application/json")

11

public Iterable list(Model model){

12

Iterable productList = productService.listAllProducts();

13

return productList;

14

}

15

. . .

One undocumented thing that took quite some of my time was related to the value of Response Content Type. Swagger 2 generated \*/\*, while I was expecting application/json for Response Content Type. It was only after updating the @RequestMapping annotation, which produces = "application/json", that the desired value got generated. The annotated ProductController is below.

ProductController.java:

1

package guru.springframework.controllers;

2

​

3

import guru.springframework.domain.Product;

4

import guru.springframework.services.ProductService;

5

import io.swagger.annotations.Api;

6

import io.swagger.annotations.ApiOperation;

7

import io.swagger.annotations.ApiResponse;

8

import io.swagger.annotations.ApiResponses;

9

import org.springframework.beans.factory.annotation.Autowired;

10

import org.springframework.http.HttpStatus;

11

import org.springframework.http.ResponseEntity;

12

import org.springframework.ui.Model;

13

import org.springframework.web.bind.annotation.\*;

14

​

15

@RestController

16

@RequestMapping("/product")

17

@Api(value="onlinestore", description="Operations pertaining to products in Online Store")

18

public class ProductController {

19

​

20

private ProductService productService;

21

​

22

@Autowired

23

public void setProductService(ProductService productService) {

24

this.productService = productService;

25

}

26

​

27

@ApiOperation(value = "View a list of available products",response = Iterable.class)

28

@ApiResponses(value = {

29

@ApiResponse(code = 200, message = "Successfully retrieved list"),

30

@ApiResponse(code = 401, message = "You are not authorized to view the resource"),

31

@ApiResponse(code = 403, message = "Accessing the resource you were trying to reach is forbidden"),

32

@ApiResponse(code = 404, message = "The resource you were trying to reach is not found")

33

}

34

)

35

@RequestMapping(value = "/list", method= RequestMethod.GET, produces = "application/json")

36

public Iterable<Product> list(Model model){

37

Iterable<Product> productList = productService.listAllProducts();

38

return productList;

39

}

40

@ApiOperation(value = "Search a product with an ID",response = Product.class)

41

@RequestMapping(value = "/show/{id}", method= RequestMethod.GET, produces = "application/json")

42

public Product showProduct(@PathVariable Integer id, Model model){

43

Product product = productService.getProductById(id);

44

return product;

45

}

46

​

47

@ApiOperation(value = "Add a product")

48

@RequestMapping(value = "/add", method = RequestMethod.POST, produces = "application/json")

49

public ResponseEntity saveProduct(@RequestBody Product product){

50

productService.saveProduct(product);

51

return new ResponseEntity("Product saved successfully", HttpStatus.OK);

52

}

53

​

54

@ApiOperation(value = "Update a product")

55

@RequestMapping(value = "/update/{id}", method = RequestMethod.PUT, produces = "application/json")

56

public ResponseEntity updateProduct(@PathVariable Integer id, @RequestBody Product product){

57

Product storedProduct = productService.getProductById(id);

58

storedProduct.setDescription(product.getDescription());

59

storedProduct.setImageUrl(product.getImageUrl());

60

storedProduct.setPrice(product.getPrice());

61

productService.saveProduct(storedProduct);

62

return new ResponseEntity("Product updated successfully", HttpStatus.OK);

63

}

64

​

65

@ApiOperation(value = "Delete a product")

66

@RequestMapping(value="/delete/{id}", method = RequestMethod.DELETE, produces = "application/json")

67

public ResponseEntity delete(@PathVariable Integer id){

68

productService.deleteProduct(id);

69

return new ResponseEntity("Product deleted successfully", HttpStatus.OK);

70

​

71

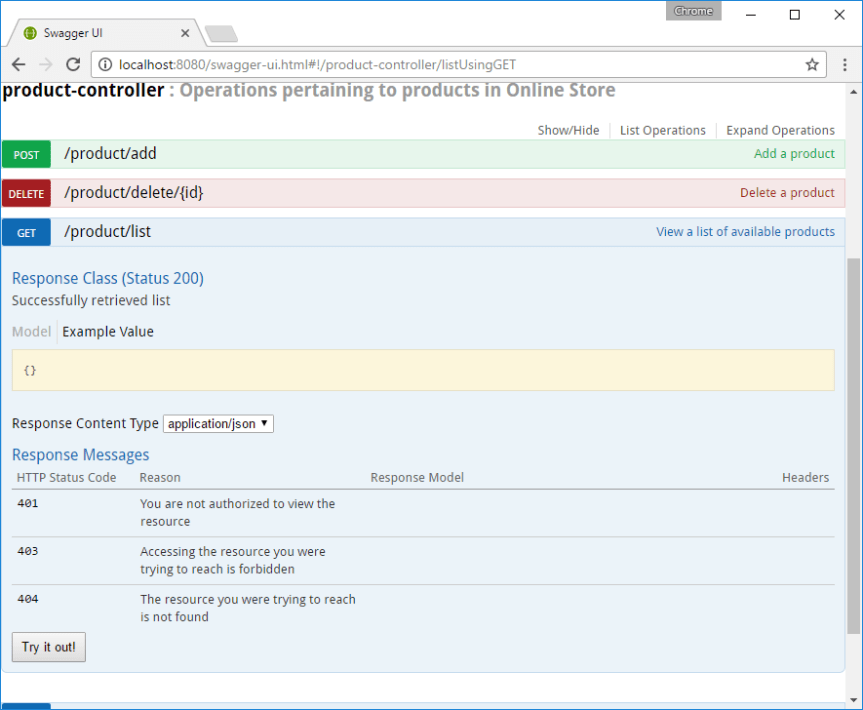
}

72

​

73

}

The output of the operation endpoints on the browser is this:  
[](https://i2.wp.com/springframework.guru/wp-content/uploads/2017/02/swagger-ui_with_endpoint_documentation.png?ssl=1)The current documentation is missing one thing: documentation of the Product JPA entity. We will generate documentation for our model next.

## Swagger 2 Annotations for Model

You can use the @ApiModelProperty annotation to describe the properties of the Product model. With @ApiModelProperty, you can also document a property as required.

The code of our Product class is this.

Product.java:

1

package guru.springframework.domain;

2

​

3

import io.swagger.annotations.ApiModelProperty;

4

​

5

import javax.persistence.\*;

6

import java.math.BigDecimal;

7

​

8

@Entity

9

public class Product {

10

@Id

11

@GeneratedValue(strategy = GenerationType.AUTO)

12

@ApiModelProperty(notes = "The database generated product ID")

13

private Integer id;

14

@Version

15

@ApiModelProperty(notes = "The auto-generated version of the product")

16

private Integer version;

17

@ApiModelProperty(notes = "The application-specific product ID")

18

private String productId;

19

@ApiModelProperty(notes = "The product description")

20

private String description;

21

@ApiModelProperty(notes = "The image URL of the product")

22

private String imageUrl;

23

@ApiModelProperty(notes = "The price of the product", required = true)

24

private BigDecimal price;

25

​

26

public String getDescription() {

27

return description;

28

}

29

​

30

public void setDescription(String description) {

31

this.description = description;

32

}

33

​

34

public Integer getVersion() {

35

return version;

36

}

37

​

38

public void setVersion(Integer version) {

39

this.version = version;

40

}

41

​

42

public Integer getId() {

43

return id;

44

}

45

​

46

public void setId(Integer id) {

47

this.id = id;

48

}

49

​

50

public String getProductId() {

51

return productId;

52

}

53

​

54

public void setProductId(String productId) {

55

this.productId = productId;

56

}

57

​

58

public String getImageUrl() {

59

return imageUrl;

60

}

61

​

62

public void setImageUrl(String imageUrl) {

63

this.imageUrl = imageUrl;

64

}

65

​

66

public BigDecimal getPrice() {

67

return price;

68

}

69

​

70

public void setPrice(BigDecimal price) {

71

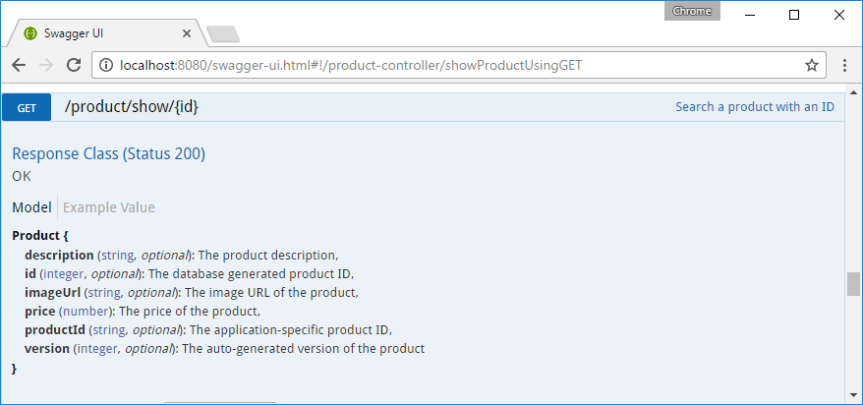
this.price = price;

72

}

73

}

The Swagger 2 generated documentation for Product is this:  
[](https://i2.wp.com/springframework.guru/wp-content/uploads/2017/02/swagger_model_documentation.png?ssl=1)

#### 7. Swagger 2 Annotation

Let’s inspect some annotation available with API which we can use to customize the output:

#### 7.1. @Api annotation

This annotation is useful to add some basic information to our method. Let’s see how to use this for our controller:

@Api(value = "ProductController" , tags = {"Product Controller"})

@SwaggerDefinition(tags = {

@Tag(name = "Product Controller", description = "Write description here")

})

@RestController("/v2")

public class ProductController {

}

#### 7.2. @ApiOperation and @ApiResponse

To provide the information about the method and response, add these annotations to the methods:

@ApiOperation(value = "List of all products", response = ArrayList.class, tags = "getProducts")

@ApiResponses(value = {

@ApiResponse(code = 200, message = "OK"),

@ApiResponse(code =404, message = "404 error")

})

@GetMapping("/products")

public List getProducts(){

return products;

}

#### 7.3. The @ApiModelProperty

Add this annotation to add description to the output:

public class Product {

@ApiModelProperty(name = "code",required = true, value = "123", notes = "product unique code")

private String code;

@ApiModelProperty(name = "name",required = true,value = "demo name", notes = "name of the product")

private String name;

@ApiModelProperty(name = "description", required = true,value = "sample description", notes = "product description")

private String description;

@ApiModelProperty(name = "price", required = true, value = "23.45", notes = "product price")

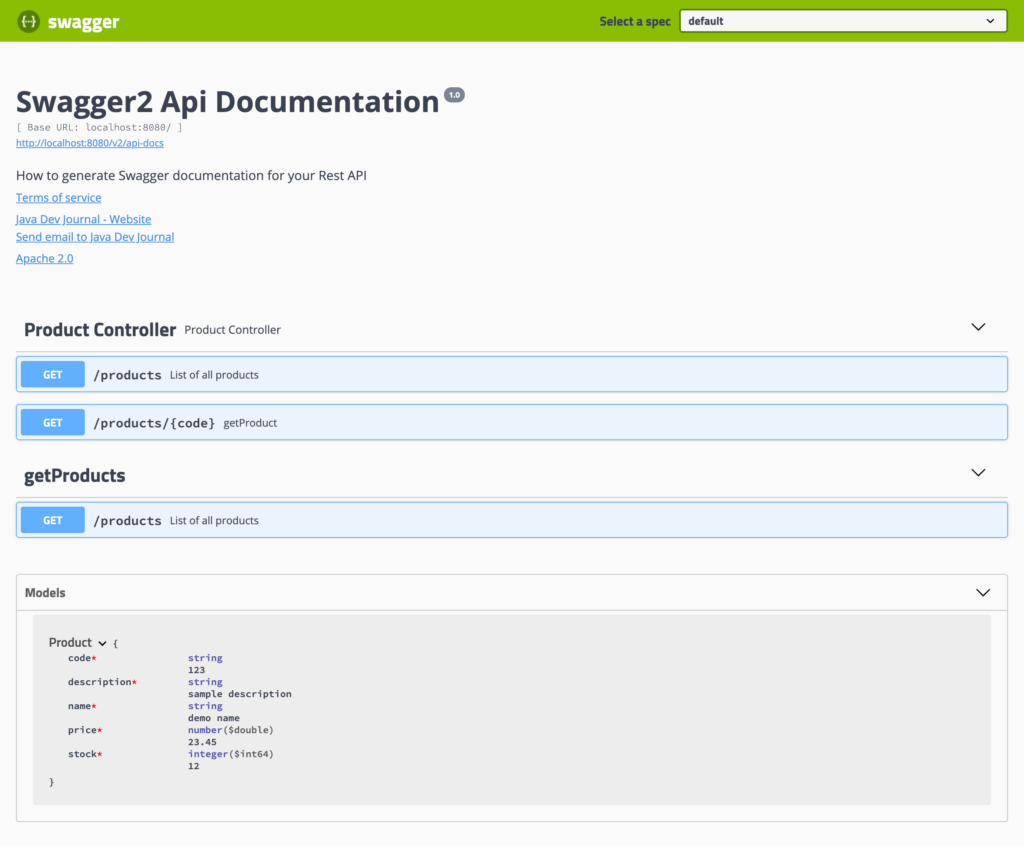
private double price;

@ApiModelProperty(name = "stock", required = true, value = "12", notes = "product stock")

private long stock;

}

Here is our final Swagger UI output:



Git Link for Swagger Project:

https://github.com/springframeworkguru/springboot\_swagger\_example