DZone

Spring Boot RESTful API Documentation With Swagger 2

by John Thompson MVB · Mar. 06, 17 · Integration Zone · Tutorial

Spring Boot makes developing RESTful services ridiculously easy — and using Swagger makes documenting your RESTful services easy.

Building a back-end API layer introduces a whole new area of challenges that goes beyond implementing just endpoints. You now have clients which will now be using your API. Your clients will need to know how to interact with your API. In SOAP-based web services, you had a WSDL to work with. This gave API developers an XML-based contract, which defined the API. However, with RESTFul web services, there is no WSDL. Thus your API documentation becomes more critical.

API documentation should be structured so that it's informative, succinct, and easy to read. But best practices on, how you document your API, its structure, what to include and what not to is altogether a different subject that I won't be covering here. For best practices on documentation, I suggest going through this presentation by Andy Wikinson.

In this post, I'll cover how to use Swagger 2 to generate REST API documentation for a Spring Boot project.

Swagger 2 in Spring Boot

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 project to display the API on the browser. Besides rendering documentation, Swagger UI allows other API developers or consumers to interact with the API's resources without having any of the implementation logic in place.

The Swagger 2 specification, which is known as OpenAPI specification, has several implementations. Currently, Springfox that has replaced Swagger-SpringMVC (Swagger 1.2 and older) is popular for Spring Boot applications. Springfox supports both Swagger 1.2 and 2.0.

We will be using Springfox in our project.

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

```
. . .

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.6.1</version>
    <scope>compile</scope>
</dependency>

. . .
```

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

```
. . .

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.6.1</version>
    <scope>compile</scope>
</dependency>

. . .
```

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`:

```
<?xml version="1.0" encoding="UTF-8"?>
<project
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.

<modelVersion>4.0.0</modelVersion>
```

```
<groupId>guru.springframework</groupId>
<artifactId>spring-boot-web</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
<name>Spring Boot Web Application</name>
<description>Spring Boot Web Application</description>
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>1.4.2.RELEASE</version>
<relativePath/>
<!-- lookup parent from repository -->
</parent>
<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<java.version>1.8</java.version>
</properties>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
<groupId>com.jayway.jsonpath</groupId>
<artifactId>json-path</artifactId>
<scope>test</scope>
</dependency>
<dependency>
```

```
<groupId>io.springfox</groupId>
<artifactId>springfox-swagger-ui</artifactId>
<version>2.6.1</version>
<scope>compile</scope>
</dependency>
<dependency>
<groupId>io.springfox</groupId>
<artifactId>springfox-swagger2</artifactId>
<version>2.6.1</version>
<scope>compile</scope>
</dependency>
<!--WebJars-->
<dependency>
<groupId>com.h2database</groupId>
<artifactId>h2</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
</dependencies>
<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
</plugins>
</build>
</project>
```

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

```
   . . .
@RestController
@RequestMapping("/product")
public class ProductController {

    private ProductService productService;

    @Autowired
    public void setProductService(ProductService productService) {
        this.productService = productService;
    }


    @RequestMapping(value = "/list", method= RequestMethod.GET)
    public Iterable list(Model model){
        Iterable productList = productService.listAllProducts();
        return productList;
    }

    @RequestMapping(value = "/show/{id}", method= RequestMethod.GET)
    public Product showProduct(@PathVariable Integer id, Model model){
       Product product = productService.getProductById(id);
        return product;
    }


    @RequestMapping(value = "/add", method = RequestMethod.POST)
    public ResponseEntity saveProduct(@RequestBody Product product){
        productService.saveProduct(product);
        return new ResponseEntity("Product saved successfully", HttpStatus.OK);
    }


    @RequestMapping(value = "/update/{id}", method = RequestMethod.PUT)
    public ResponseEntity updateProduct(@PathVariable Integer id, @RequestBody Product produc
        Product storedProduct = productService.getProductById(id);
        storedProduct.setDescription(product.getDescription());
        storedProduct.setImageUrl(product.getImageUrl());
        storedProduct.setPrice(product.getPrice());
        productService.saveProduct(storedProduct);
        return new ResponseEntity("Product updated successfully", HttpStatus.OK);
    }


    @RequestMapping(value="/delete/{id}", method = RequestMethod.DELETE)
    public ResponseEntity delete(@PathVariable Integer id){
        productService.deleteProduct(id);
        return new ResponseEntity("Product deleted successfully", HttpStatus.OK);

    }

}
```

```
. . .
```

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

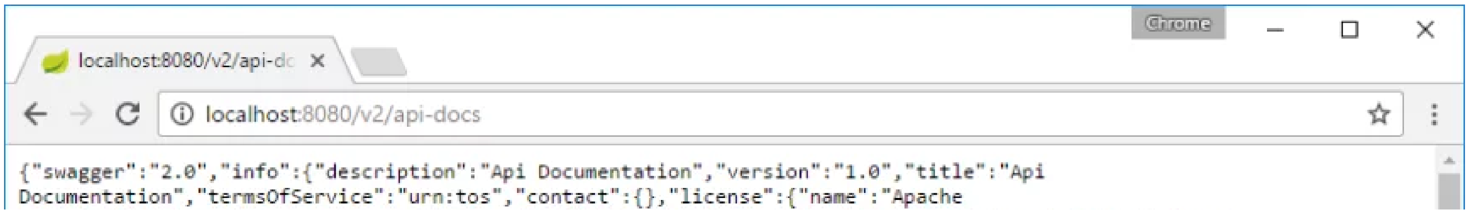
For our application, we will create a Docket bean in a Spring Boot configuration to configure Swagger 2 for the application. A Springfox Docket instance provides the primary API configuration with sensible defaults and convenience methods for configuration. Our Spring Boot configuration class, SwaggerConfig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

```
. . .
@Configuration
@EnableSwagger2
public class SwaggerConfig {
    @Bean
    public Docket productApi() {
        return new Docket(DocumentationType.SWAGGER_2)
                .select()                 .apis(RequestHandlerSelectors.basePackage("guru.spı

                .paths(regex("/product.*"))
                .build();

    }
}
. . .
```

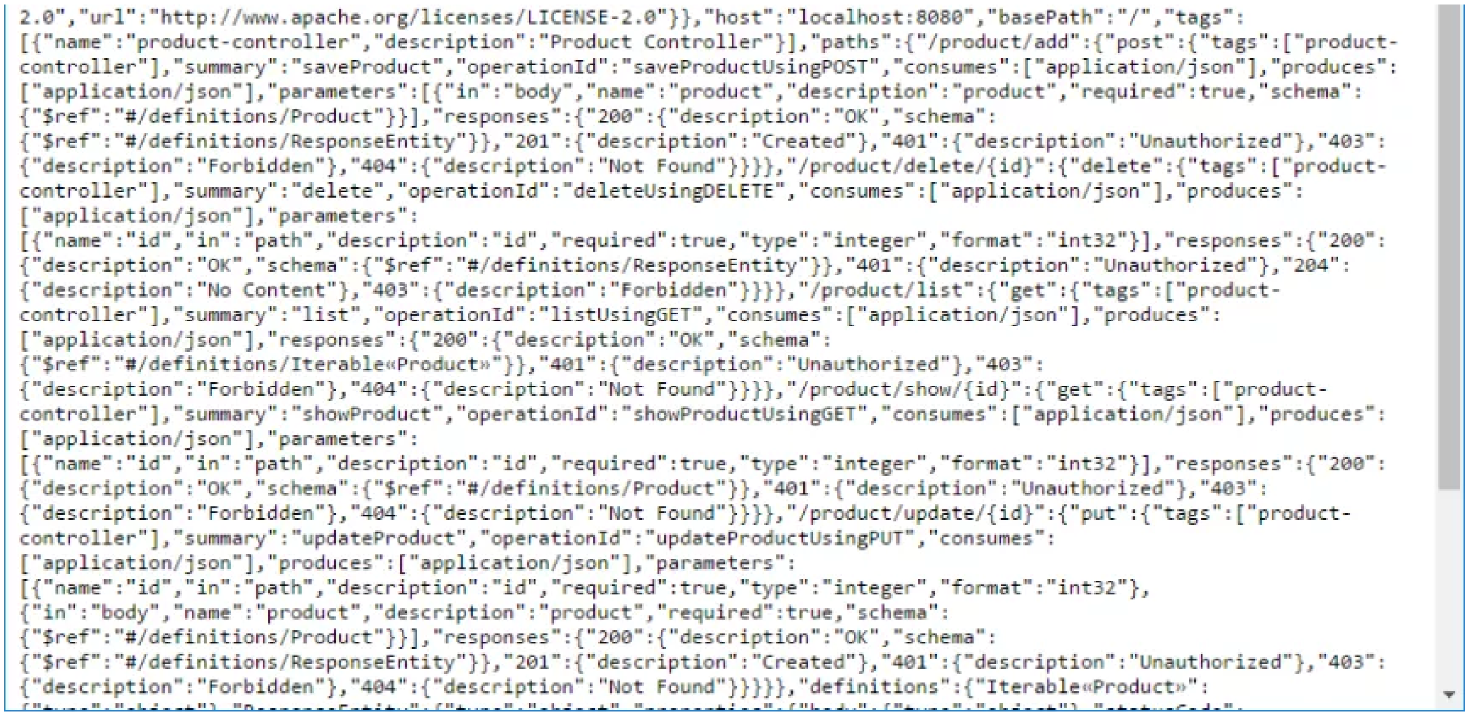
In this configuration class, the `@EnableSwagger2` annotation enables Swagger support in the class. The `select()` method called on the Docket bean instance returns an `ApiSelectorBuilder`, which provides the `apis()` and `paths()` methods that are used to filter the controllers and methods that are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



```
{"swagger":"2.0","info":{"description":"Api Documentation","version":"1.0","title":"Api
Documentation","termsOfService":"urn:tos","contact":{},"license":{"name":"Apache
```

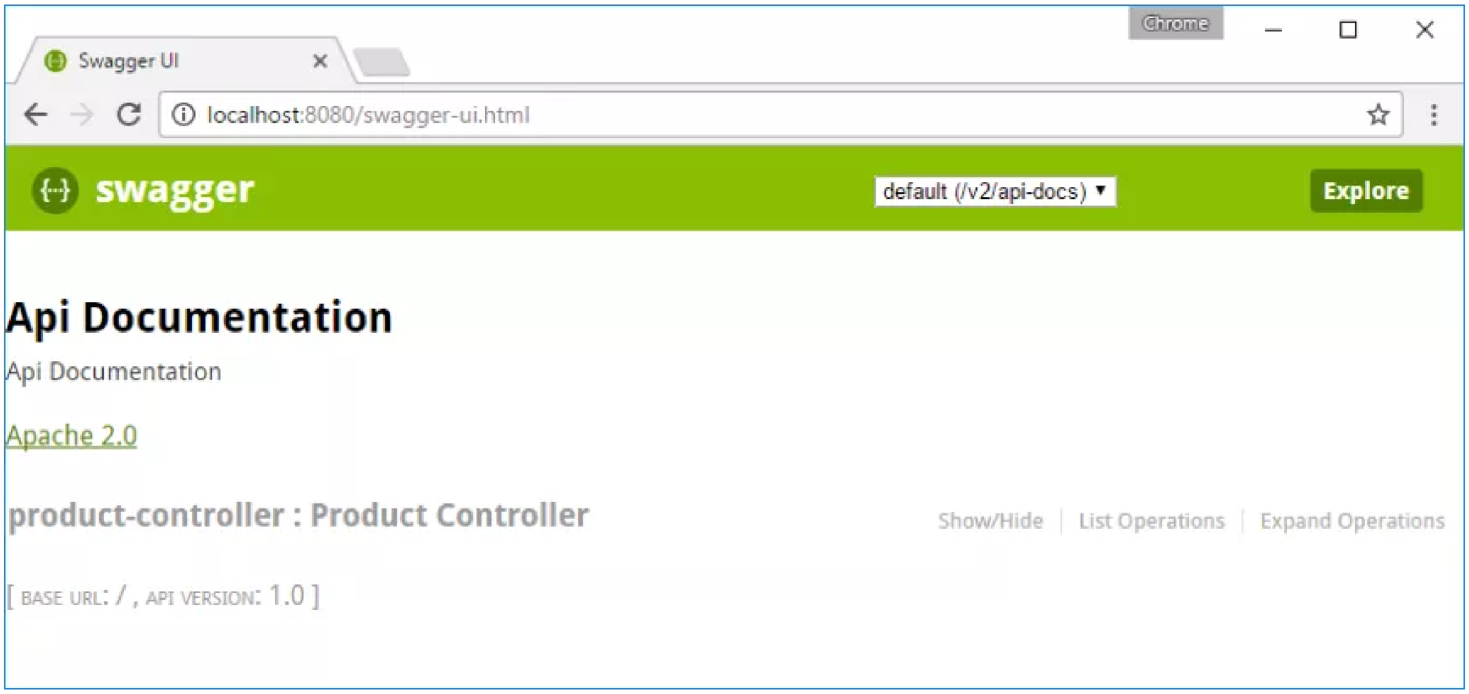


```
2.0","url":"http://www.apache.org/licenses/LICENSE-2.0"}},"host":"localhost:8080","basePath":"/","tags":
[{"name":"product-controller","description":"Product Controller"}],"paths":{"/product/add":{"post":{"tags":["product-
controller"],"summary":"saveProduct","operationId":"saveProductUsingPOST","consumes":["application/json"],"produces":
["application/json"],"parameters":[{"in":"body","name":"product","description":"product","required":true,"schema":
{"$ref":"#/definitions/Product"}}],"responses":{"200":{"description":"OK","schema":
{"$ref":"#/definitions/ResponseEntity"}},"201":{"description":"Created"},"401":{"description":"Unauthorized"},"403":
{"description":"Forbidden"},"404":{"description":"Not Found"}}}},"/product/delete/{id}":{"delete":{"tags":["product-
controller"],"summary":"delete","operationId":"deleteUsingDELETE","consumes":["application/json"],"produces":
["application/json"],"parameters":
[{"name":"id","in":"path","description":"id","required":true,"type":"integer","format":"int32"}],"responses":{"200":
{"description":"OK","schema":{"$ref":"#/definitions/ResponseEntity"}},"401":{"description":"Unauthorized"},"204":
{"description":"No Content"},"403":{"description":"Forbidden"}}}},"/product/list":{"get":{"tags":["product-
controller"],"summary":"list","operationId":"listUsingGET","consumes":["application/json"],"produces":
["application/json"],"responses":{"200":{"description":"OK","schema":
{"$ref":"#/definitions/Iterable«Product»"}},"401":{"description":"Unauthorized"},"403":
{"description":"Forbidden"},"404":{"description":"Not Found"}}}},"/product/show/{id}":{"get":{"tags":["product-
controller"],"summary":"showProduct","operationId":"showProductUsingGET","consumes":["application/json"],"produces":
["application/json"],"parameters":
[{"name":"id","in":"path","description":"id","required":true,"type":"integer","format":"int32"}],"responses":{"200":
{"description":"OK","schema":{"$ref":"#/definitions/Product"}},"401":{"description":"Unauthorized"},"403":
{"description":"Forbidden"},"404":{"description":"Not Found"}}}},"/product/update/{id}":{"put":{"tags":["product-
controller"],"summary":"updateProduct","operationId":"updateProductUsingPUT","consumes":
["application/json"],"produces":["application/json"],"parameters":
[{"name":"id","in":"path","description":"id","required":true,"type":"integer","format":"int32"},
{"in":"body","name":"product","description":"product","required":true,"schema":
{"$ref":"#/definitions/Product"}}],"responses":{"200":{"description":"OK","schema":
{"$ref":"#/definitions/ResponseEntity"}},"201":{"description":"Created"},"401":{"description":"Unauthorized"},"403":
{"description":"Forbidden"},"404":{"description":"Not Found"}}}}},"definitions":{"Iterable«Product»":
```

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:



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.

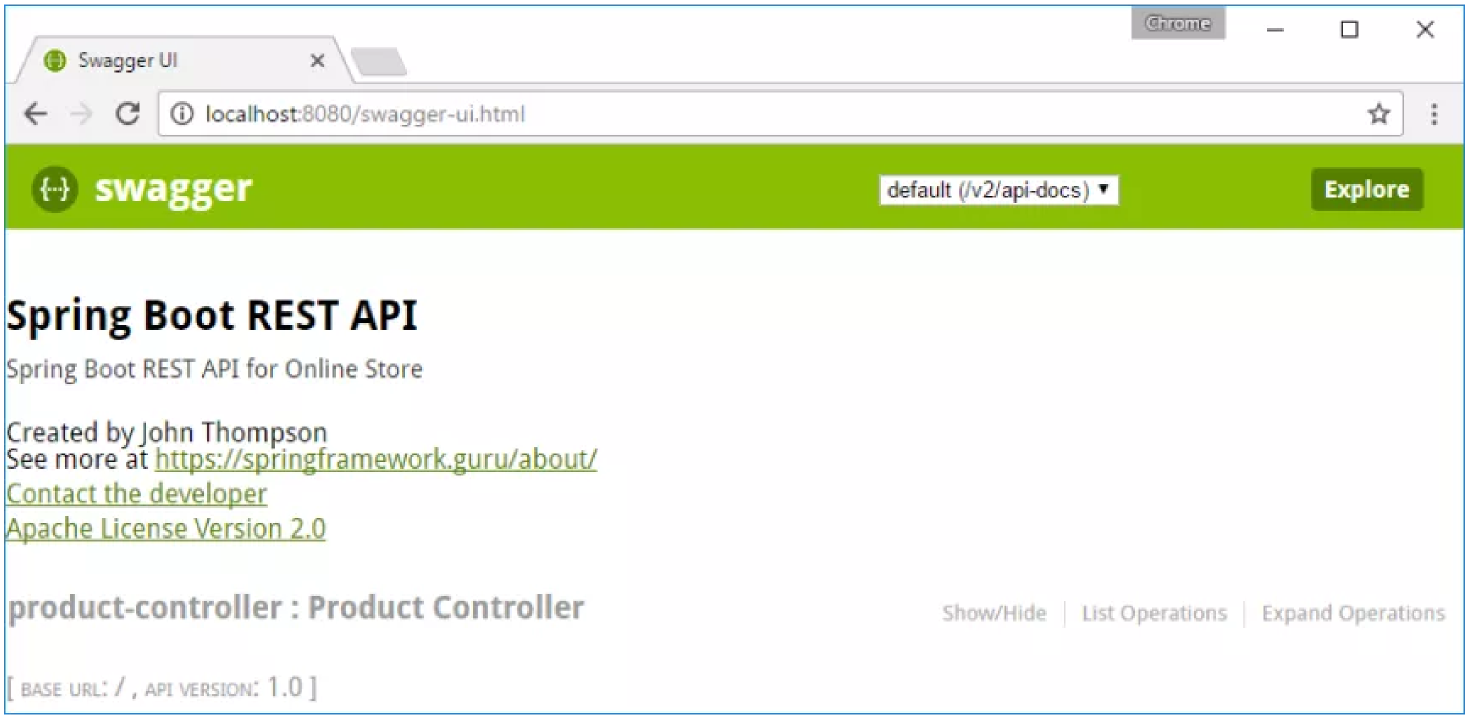
Let’s start customizing Swagger by providing information about our API in the SwaggerConfig class like this.

`SwaggerConfig.java`:

```
package guru.springframework.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
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;

@Configuration
@EnableSwagger2
public class SwaggerConfig {
    @Bean
    public Docket productApi() {
        return new Docket(DocumentationType.SWAGGER_2)
                .select()
                .apis(RequestHandlerSelectors.basePackage("guru.springframework.controllers"
                .paths(regex("/product.*"))
                .build()
                .apiInfo(metaData());
    }
    private ApiInfo metaData() {
        ApiInfo apiInfo = new ApiInfo(
                "Spring Boot REST API",
                "Spring Boot REST API for Online Store",
                "1.0",
                "Terms of service",
                new Contact("John Thompson", "https://springframework.guru/about/", "john@spr
               "Apache License Version 2.0",
                "https://www.apache.org/licenses/LICENSE-2.0");
        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:



Spring Boot REST API

Spring Boot REST API for Online Store

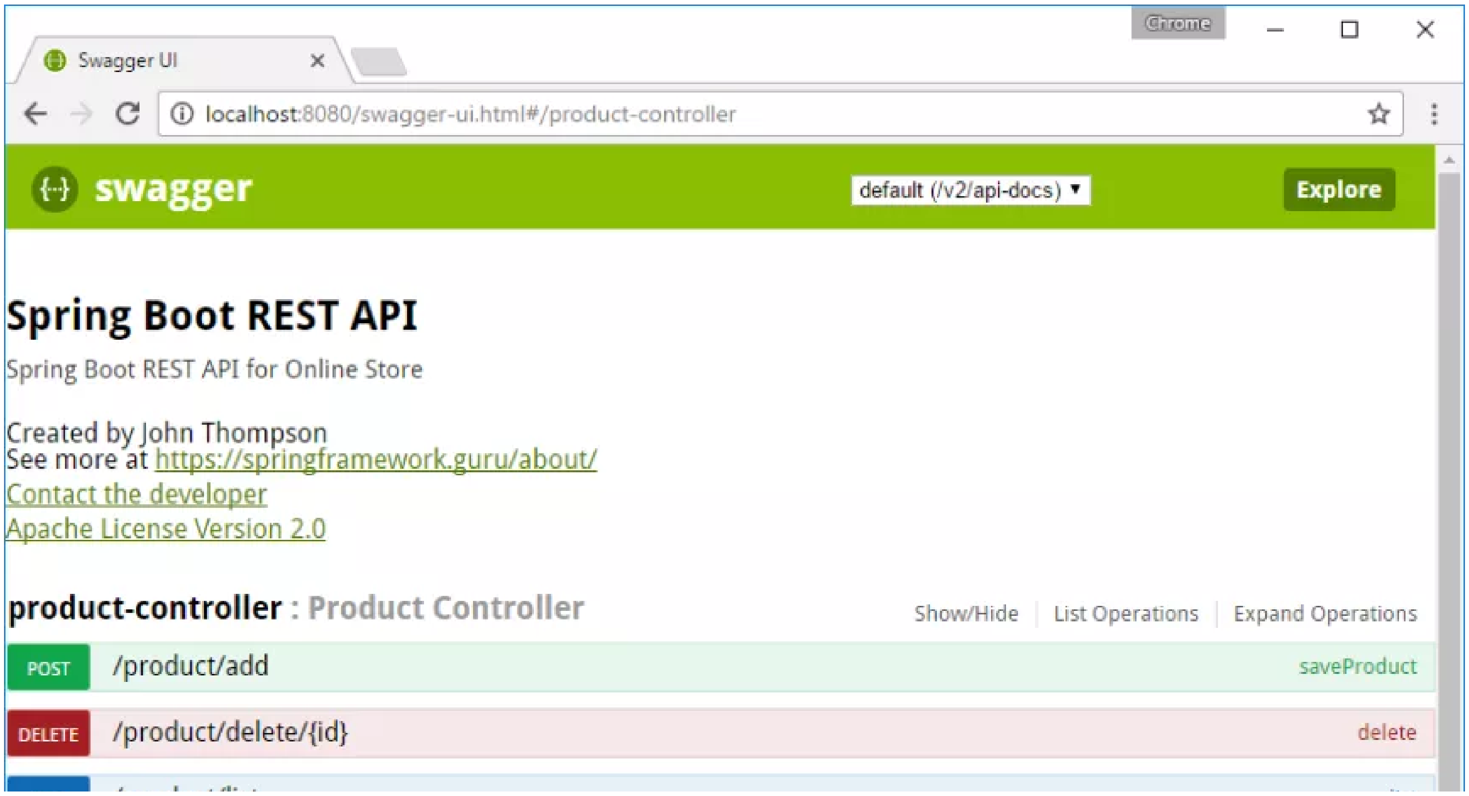
Created by John Thompson
See more at https://springframework.guru/about/
Contact the developer
Apache License Version 2.0

product-controller : Product Controller

[BASE URL: / , API VERSION: 1.0]

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:



Spring Boot REST API

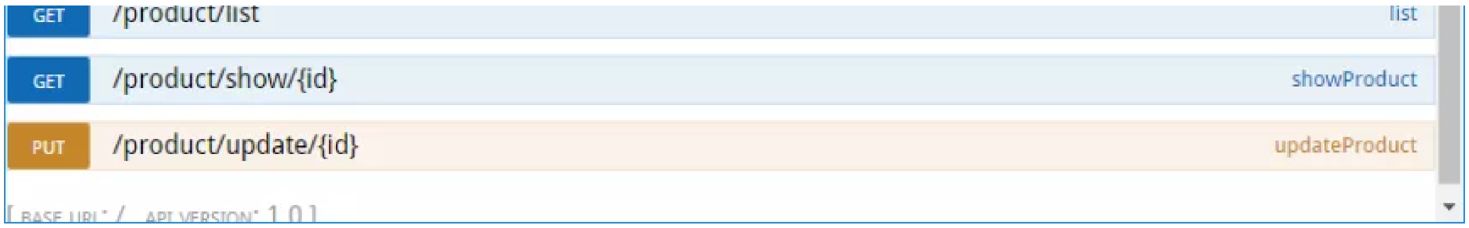
Spring Boot REST API for Online Store

Created by John Thompson
See more at https://springframework.guru/about/
Contact the developer
Apache License Version 2.0

product-controller : Product Controller

POST /product/add saveProduct

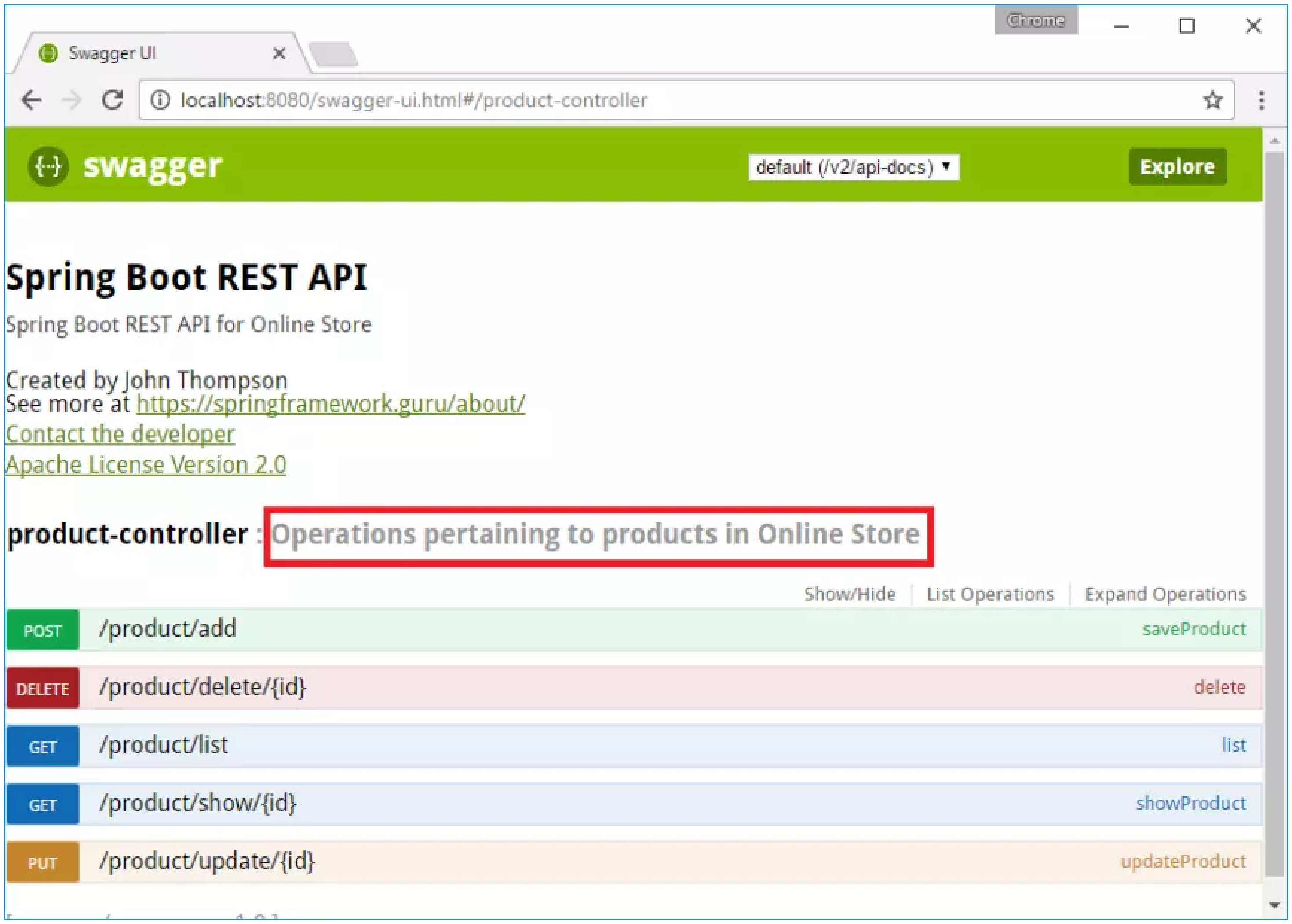
DELETE /product/delete/{id} delete


GET /product/list list
GET /product/show/{id} showProduct
PUT /product/update/{id} updateProduct
[BASE URL: / , API VERSION: 1.0]

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

```
@RestController @RequestMapping("/product") @Api(value="onlinestore", description="Operation
```

The Swagger UI-generated documentation will reflect the description and now looks like this:



Spring Boot REST API

Spring Boot REST API for Online Store

Created by John Thompson
See more at https://springframework.guru/about/
Contact the developer
Apache License Version 2.0

product-controller : Operations pertaining to products in Online Store

Show/Hide | List Operations | Expand Operations

POST /product/add saveProduct
DELETE /product/delete/{id} delete
GET /product/list list
GET /product/show/{id} showProduct
PUT /product/update/{id} updateProduct

For each of our operation endpoints, we can use the `@ApiOperation` annotation to describe the endpoint and its response type, like this:

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

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

public Iterable list(Model model){
    Iterable productList = productService.listAllProducts();
    return productList;
    return productList;
}
. . .
```

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.

```
. . .
@ApiOperation(value = "View a list of available products", response = Iterable.class)
@ApiResponses(value = {
        @ApiResponse(code = 200, message = "Successfully retrieved list"),
        @ApiResponse(code = 401, message = "You are not authorized to view the resource"),
        @ApiResponse(code = 403, message = "Accessing the resource you were trying to reach i
        @ApiResponse(code = 404, message = "The resource you were trying to reach is not foun
}
)
@RequestMapping(value = "/list", method= RequestMethod.GET, produces = "application/json")
public Iterable list(Model model){
    Iterable productList = productService.listAllProducts();
    return productList;
}
. . .
```

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.

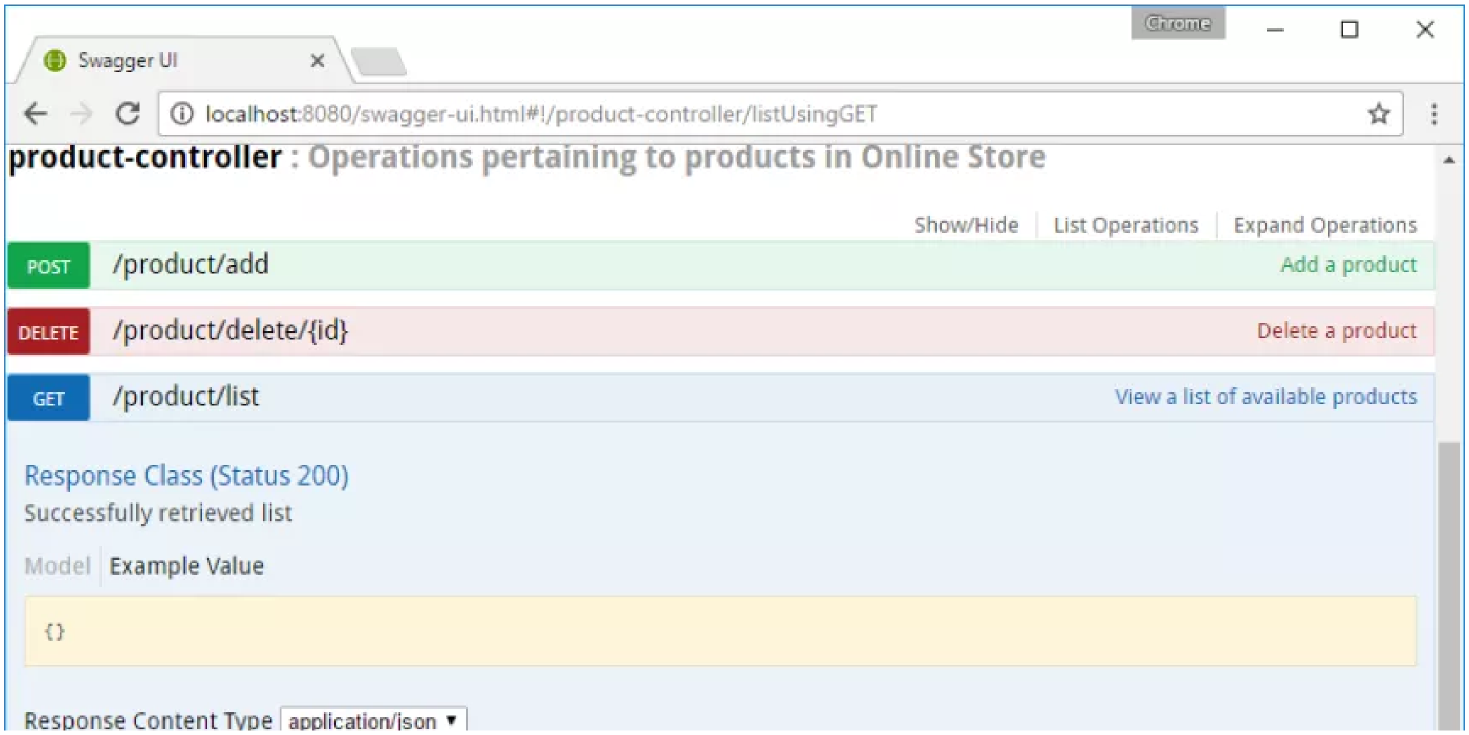
`ProductController.java`:

```
package guru.springframework.controllers;

import guru.springframework.domain.Product;
import guru.springframework.services.ProductService;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiResponse;
import io.swagger.annotations.ApiResponses;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
```

```java

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

public class ProductController {

    private ProductService productService;

    @Autowired
    public void setProductService(ProductService productService) {
        this.productService = productService;
    }

    @ApiOperation(value = "View a list of available products",response = Iterable.class)
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "Successfully retrieved list"),
            @ApiResponse(code = 401, message = "You are not authorized to view the resource"

            @ApiResponse(code = 403, message = "Accessing the resource you were trying to rea

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

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

    public Iterable<Product> list(Model model){
        Iterable<Product> productList = productService.listAllProducts();
        return productList;
    }
    @ApiOperation(value = "Search a product with an ID",response = Product.class)
    @RequestMapping(value = "/show/{id}", method= RequestMethod.GET, produces = "application,

    public Product showProduct(@PathVariable Integer id, Model model){
       Product product = productService.getProductById(id);
        return product;
    }

    @ApiOperation(value = "Add a product")
    @RequestMapping(value = "/add", method = RequestMethod.POST, produces = "application/jsor

    public ResponseEntity saveProduct(@RequestBody Product product){
        productService.saveProduct(product);
```

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

        @ApiOperation(value = "Update a product")
        @RequestMapping(value = "/update/{id}", method = RequestMethod.PUT, produces = "applicat:
        public ResponseEntity updateProduct(@PathVariable Integer id, @RequestBody Product produ
            Product storedProduct = productService.getProductById(id);
            storedProduct.setDescription(product.getDescription());
            storedProduct.setImageUrl(product.getImageUrl());
            storedProduct.setPrice(product.getPrice());
            productService.saveProduct(storedProduct);
            return new ResponseEntity("Product updated successfully", HttpStatus.OK);
        }

        @ApiOperation(value = "Delete a product")
        @RequestMapping(value="/delete/{id}", method = RequestMethod.DELETE, produces = "applicat
        public ResponseEntity delete(@PathVariable Integer id){
            productService.deleteProduct(id);
            return new ResponseEntity("Product deleted successfully", HttpStatus.OK);

        }

    }
```

The output of the operation endpoints on the browser is this:


Chrome
Swagger UI
localhost:8080/swagger-ui.html#!/product-controller/listUsingGET
Show/Hide
List Operations
Expand Operations

product-controller : Operations pertaining to products in Online Store

POST /product/add Add a product

DELETE /product/delete/{id} Delete a product

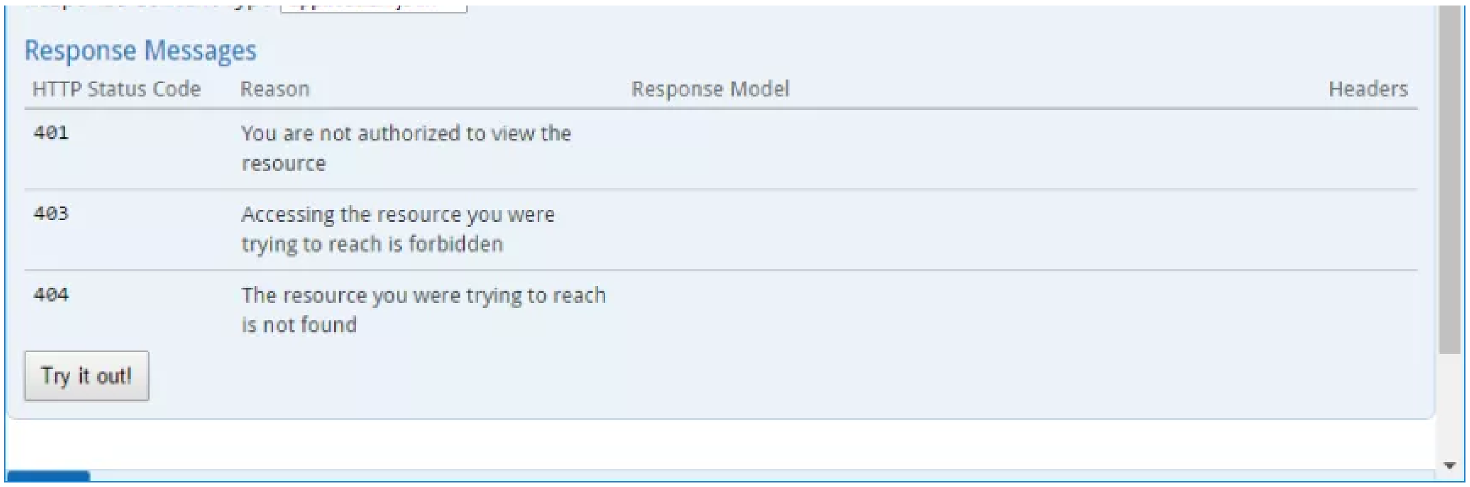
GET /product/list View a list of available products

Response Class (Status 200)
Successfully retrieved list

Model Example Value

```
{}
```

Response Content Type application/json



Response Messages

HTTP Status Code	Reason	Response Model	Headers
401	You are not authorized to view the resource		
403	Accessing the resource you were trying to reach is forbidden		
404	The resource you were trying to reach is not found		

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

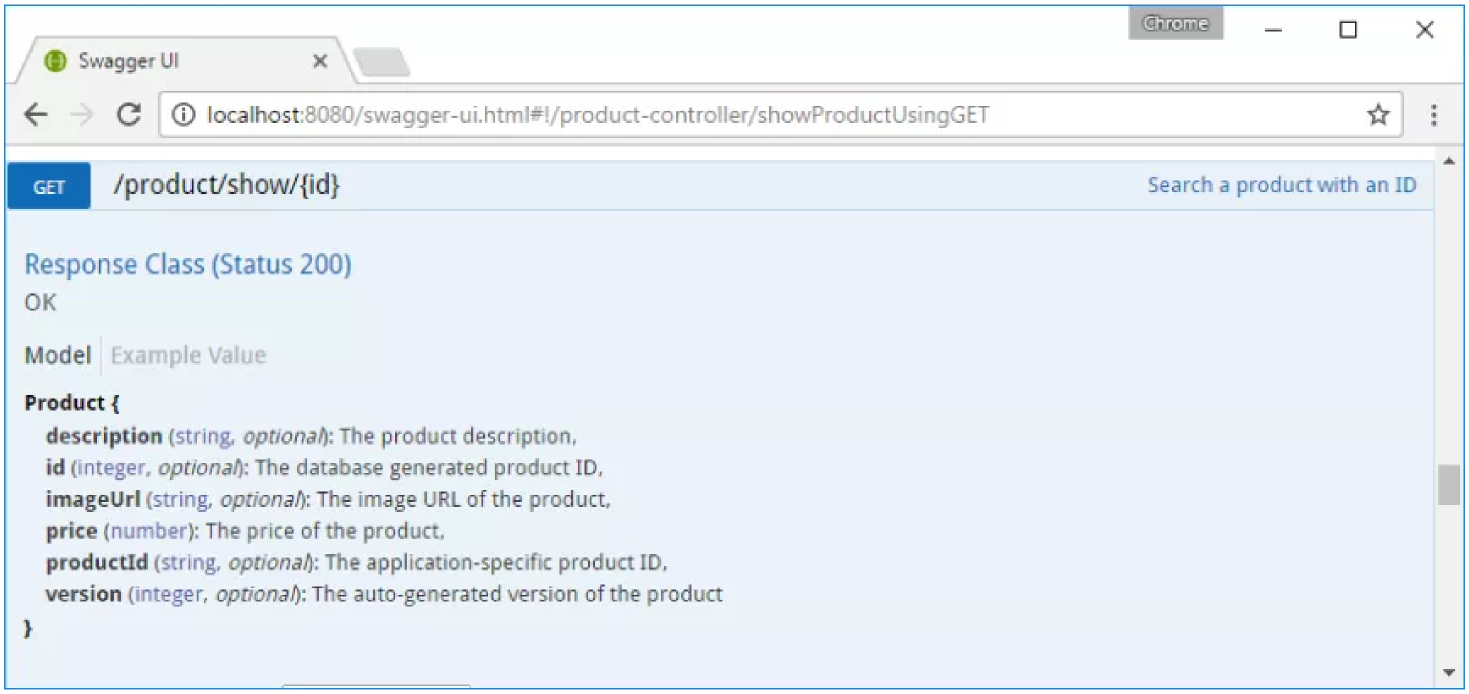
You can use the `@ApiModelProperty` annotation to describe the properties of the Product model. With `@ApiModelProperty`, you can also document a property as required.

The code of our Product class is this.

`Product.java`:

```java
package guru.springframework.domain;

import io.swagger.annotations.ApiModelProperty;

import javax.persistence.*;
import java.math.BigDecimal;

@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @ApiModelProperty(notes = "The database generated product ID")
    private Integer id;
    @Version
    @ApiModelProperty(notes = "The auto-generated version of the product")
    private Integer version;
    @ApiModelProperty(notes = "The application-specific product ID")
    private String productId;
    @ApiModelProperty(notes = "The product description")
    private String description;
    @ApiModelProperty(notes = "The image URL of the product")
    private String imageUrl;
    @ApiModelProperty(notes = "The price of the product", required = true)
    private BigDecimal price;

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Integer getVersion() {
        return version;
    }

    public void setVersion(Integer version) {
        this.version = version;
    }

    public Integer getId() {
        return id;
    }

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

    public String getProductId() {
        return productId;
    }

    public void setProductId(String productId) {
        this.productId = productId;
    }

    public String getImageUrl() {
        return imageUrl;
    }

    public void setImageUrl(String imageUrl) {
        this.imageUrl = imageUrl;
    }

    public BigDecimal getPrice() {
        return price;
    }

    public void setPrice(BigDecimal price) {
        this.price = price;
    }
}
```

The Swagger 2 generated documentation for Product is this:


Chrome
Swagger UI
localhost:8080/swagger-ui.html#!/product-controller/showProductUsingGET
GET
/product/show/{id}
Search a product with an ID
Response Class (Status 200)
OK
Model
Example Value

```
Product {
  description (string, optional): The product description,
  id (integer, optional): The database generated product ID,
  imageUrl (string, optional): The image URL of the product,
  price (number): The price of the product,
  productId (string, optional): The application-specific product ID,
  version (integer, optional): The auto-generated version of the product
}
```

Summary

Besides REST API documentation and presentation with Swagger Core and Swagger UI, Swagger 2 has a whole lot of other uses beyond the scope of this post. One of my favorites is Swagger Editor, a tool to design new APIs or edit existing ones. The editor visually renders your Swagger definition and provides real-time error-feedback. Another one is Swagger Codegen, a code generation framework for building Client SDKs, servers, and documentation from Swagger definitions.

Swagger 2 also supports Swagger definition through JSON and YAML files. It is something you should try if you want to avoid implementation-specific code in your codebase by externalizing them in JSON and YAML files — something that I will cover in a future post.

The code for this post is available for download here.

Like This Article? Read More From DZone

Spring Boot 2 RESTful API Documentation With Swagger 2 Tutorial

Static API Documentation With Spring and Swagger

Versioning a REST API With Spring Boot and Swagger

Free DZone Refcard
Open Source API Management

Topics: SPRING BOOT , REST API , DOCUMENTATION , INTEGRATION , TUTORIAL , SWAGGER 2