**Swagger2 API:**

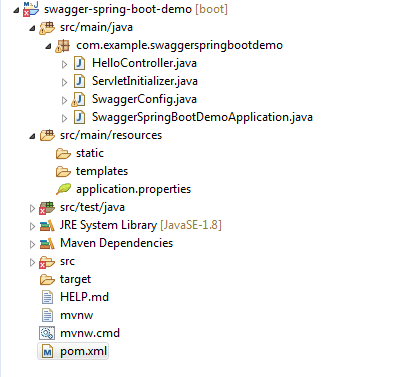
**Spring Boot + Swagger2**

In this example we configure a **spring boot application** to integrate **swagger2**. [**Spring Boot example**](https://www.javainuse.com/spring/SpringBoot_HelloWorld)**we had exposed a REST API. Documentation of such REST Services we develop is very important**. This documentation should help consumers of the service know which all services are available, the signatures, and the expected input. Also there should be some simple way to test if the service is up. The exposed services are bound to change so simultaneously the documentation would also need to be updated. If this is done manually then it will be a very tedious process, especially as the number of REST services increase. This is where swagger comes into picture. It helps automate this documentation process. 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.

**What is Swagger**

Swagger is widely used for visualizing APIs, and with Swagger UI it provides online sandbox for frontend developers. For the tutorial, we will use the Springfox implementation of the Swagger 2 specification. Swagger is a tool, a specification and a complete framework implementation for producing the visual representation of RESTful Web Services. It enables documentation to be updated at the same pace as the server. When properly defined via Swagger, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. Thus Swagger removes the guesswork in calling the service.

**Ex1: Spring Boot Rest+Swagger2:**

****

**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-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>2.1.6.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>swagger-spring-boot-demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<name>swagger-spring-boot-demo</name>

<description>Demo project for Spring Boot</description>

<properties>

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.9.2</version>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

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

<version>2.9.2</version>

</dependency>

<dependency>

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

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

<scope>provided</scope>

</dependency>

<dependency>

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

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

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

<exclusion>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<snapshots>

<enabled>true</enabled>

</snapshots>

</repository>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<snapshots>

<enabled>true</enabled>

</snapshots>

</pluginRepository>

<pluginRepository>

<id>spring-milestones</id>

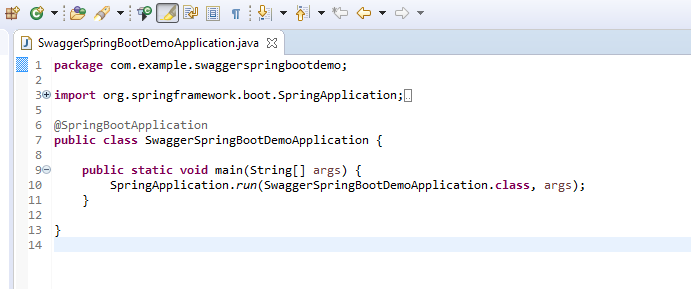
<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

</pluginRepository>

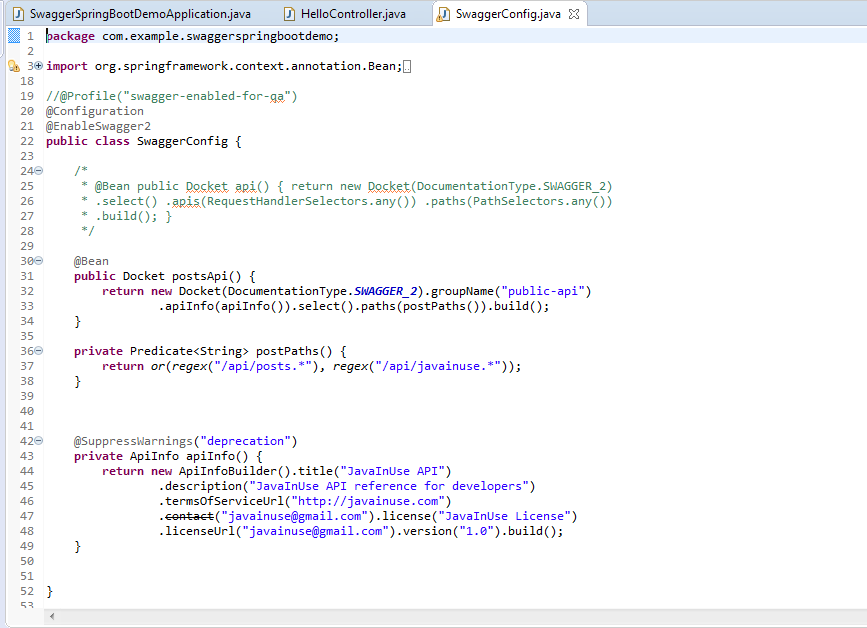
</pluginRepositories>

</project>

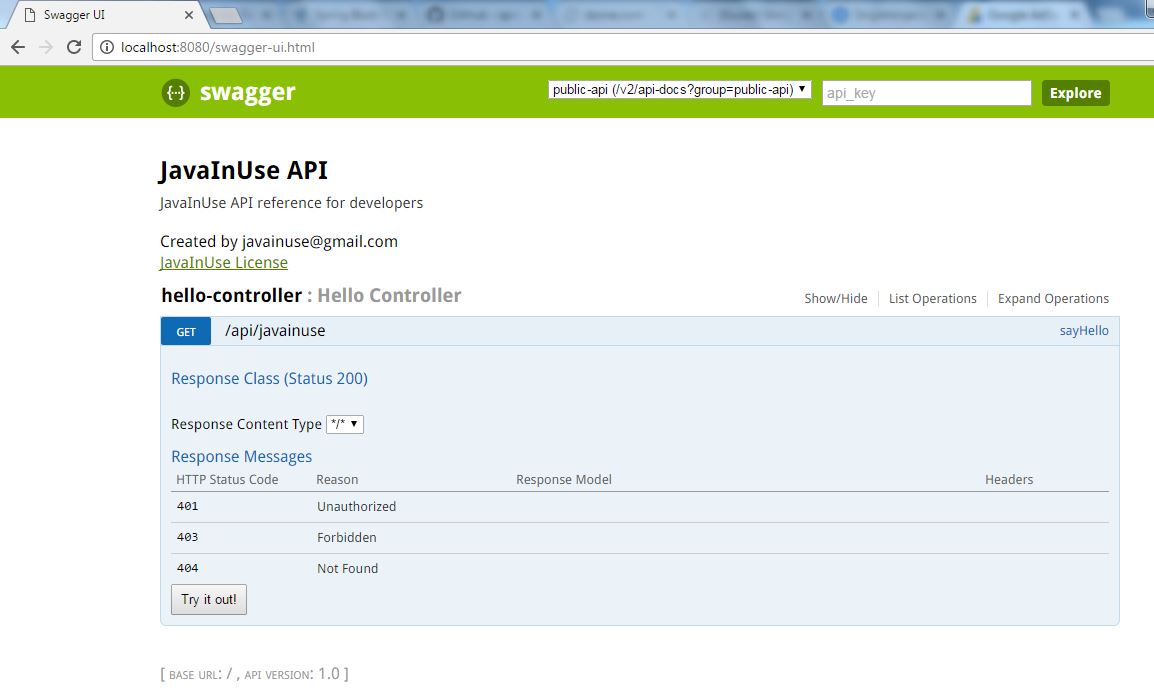
****

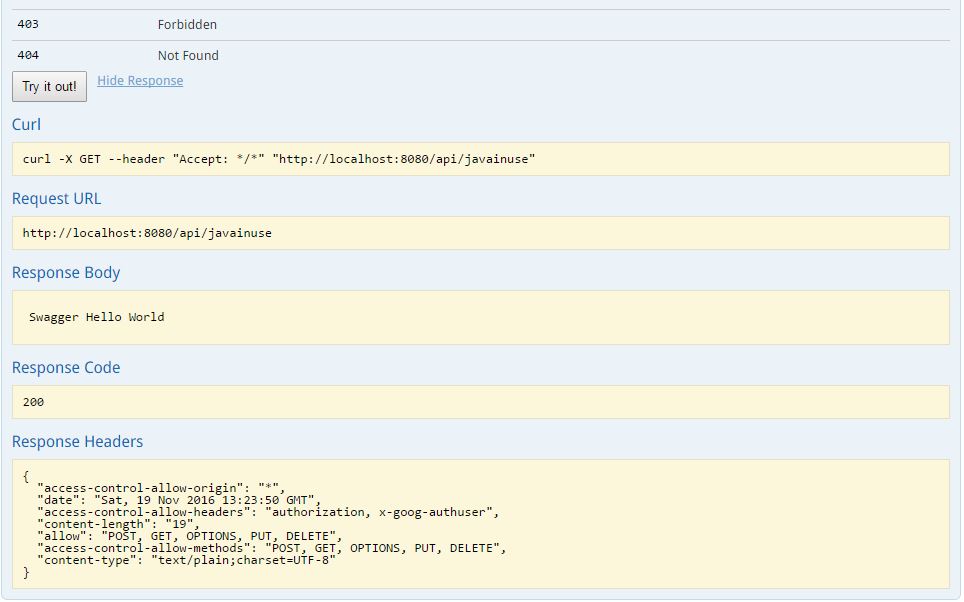
****

* **To enable the Swagger 2** we use the annotation **@EnableSwagger2.**
* A **Docket bean is defined** and using its **select()** method we get an instance of **ApiSelectorBuilder**.
* **ApiSelectorBuilder** we configure the endpoints exposed by Swagger.
* After the **Docket** bean is defined, it’s select() method returns an instance of **ApiSelectorBuilder**, which provides a way to control the endpoints exposed by Swagger.
* Using the **RequestHandlerSelectors** and **PathSelectors** we configure the predicates for selection of RequestHandlers.

****

These are the only changes required. Now go to **http://localhost:8080/swagger-ui.html**.  
We will see the documentation for the exposed API as follows-





# Spring Boot + Swagger2- Understanding various Swagger annotations-

# In [previous example we configured a spring boot application to integrate swagger2.](https://www.javainuse.com/spring/boot_swagger) Various Swagger annotations are available to help document the REST APIs. Let’s take a look at these. We will add these annotations to the sayHello() method we defined in the previous example.

* **@ApiOperation-**

This annotation is used to describe the exposed REST API. It describes an operation or typically a HTTP method against a specific path. It takes the following parameters-

|  |  |
| --- | --- |
| **Annotation Parameter** | **Description** |
| value | The value of the annotation is a short description on the API. Since this is displayed in the list of operations in Swagger-UI and the location is limited in size, this should be kept short (preferably shorter than 120 characters) |
| notes | The notes allows you to give significantly more details about the operations (e.g. you can include request samples and responses here) |
| nickname | The nickname for this API. |

The code will be as follows for the hello method-

**@ApiOperation(value = "getGreeting", notes="get greeting",nickname = "getGreeting")**

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

public <Hello> sayHello() {

ArrayList<Hello> arrayList= new ArrayList<>();

arrayList.add(new Hello());

return arrayList;

}

****

* **@ApiResponses-**

This annotation is used to describe the expected responses for the REST API. The **@ApiResponse** describes a concrete possible response. It cannot be used directly on the method and needs to be included in the array value of **@ApiResponses (whether there's one response or more)**. It takes the following parameters-

|  |  |
| --- | --- |
| **Annotation Parameter** | **Description** |
| ApiResponse | The @ApiResponse describes a concrete possible response |

**Syntax:**

@ApiOperation(value = "getGreeting", nickname = "getGreeting")

**@ApiResponses(value = {**

**@ApiResponse(code = 500, message = "Server error"),**

**@ApiResponse(code = 404, message = "Service not found"),**

**@ApiResponse(code = 200, message = "Successful retrieval",**

**response = Hello.class, responseContainer = "List") })**

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

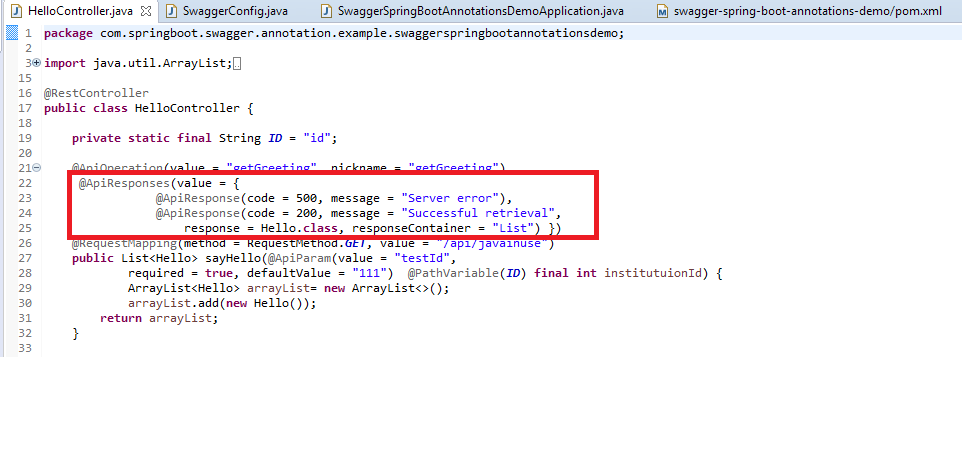
public <Hello> sayHello() {

ArrayList<Hello> arrayList= new ArrayList<>();

arrayList.add(new Hello());

return arrayList;

}

****

If the user has default response messages which are to be applied to all the REST APIs then these can be specified when defining the Docket bean. Hence these will not need to be applied at the method level. For example if the response for code 404 and 500 is going to be same throughout all services

@Bean

public Docket postsApi() {

Docket docket=new Docket(DocumentationType.SWAGGER\_2);

docket.groupName("public-api")

.apiInfo(apiInfo()).select().paths(postPaths()).build();

**docket.globalResponseMessage(RequestMethod.GET, ImmutableList.of(new ResponseMessageBuilder()**

**.code(400)**

**.message("Bad Request")**

**.responseModel(new ModelRef("Error")).build(),new ResponseMessageBuilder()**

**.code(500)**

**.message("Internal Server Error")**

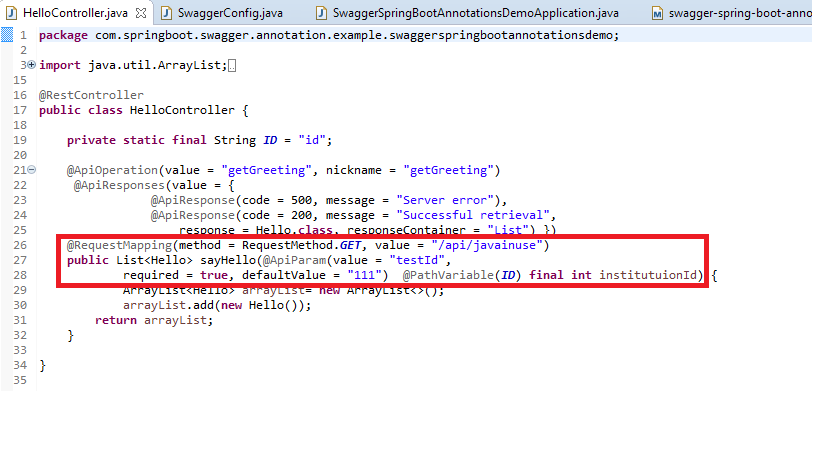
**.responseModel(new ModelRef("Error")).build()));**

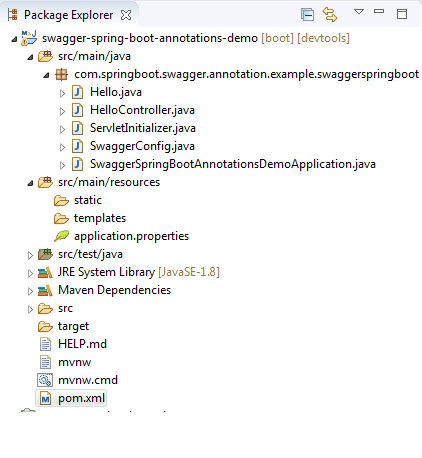
return docket;

* **@ApiParam-**

This annotation is used to describe the exposed REST API. It takes the following parameters-

|  |  |
| --- | --- |
| **Annotation Parameter** | **Description** |
| value | The value is a short description of the parameter |
| required | If the parameter is optional or required. |
| defaultValue | Specify defaultValue of the parameter. |

****

****

**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-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>2.1.6.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.springboot.swagger.annotation.example</groupId>

<artifactId>swagger-spring-boot-annotations-demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<name>swagger-spring-boot-annotations-demo</name>

<description>Demo project for Spring Boot</description>

<properties>

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.9.2</version>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

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

<version>2.9.2</version>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

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

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

<scope>provided</scope>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<!-- <dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-test</artifactId>

<scope>test</scope>

</dependency> -->

</dependencies>

<build>

<plugins>

<plugin>

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

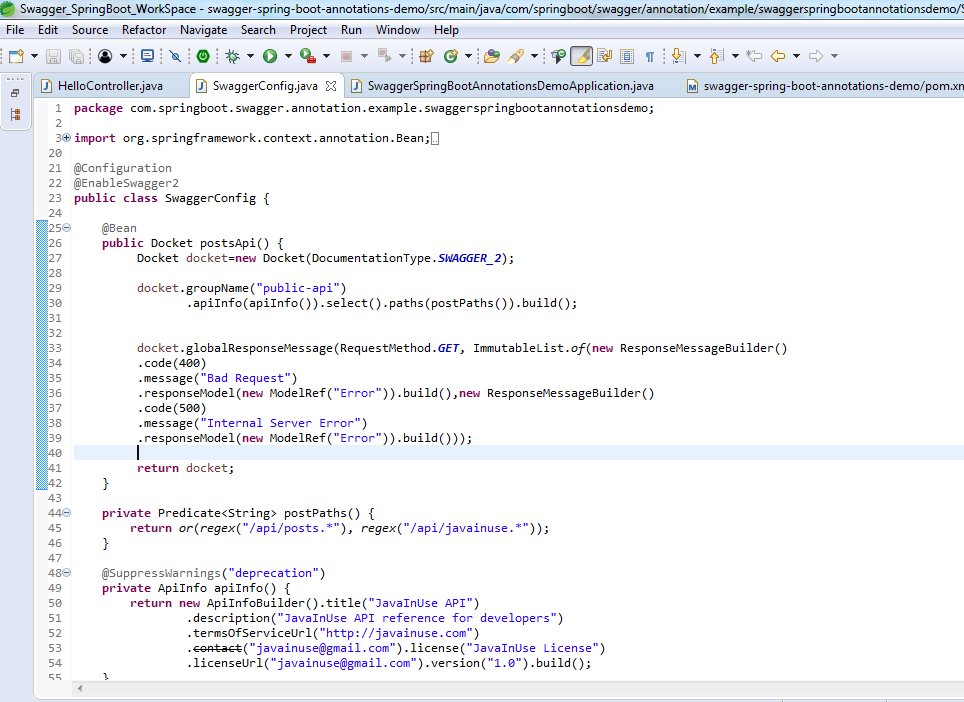
<artifactId>spring-boot-maven-plugin</artifactId>

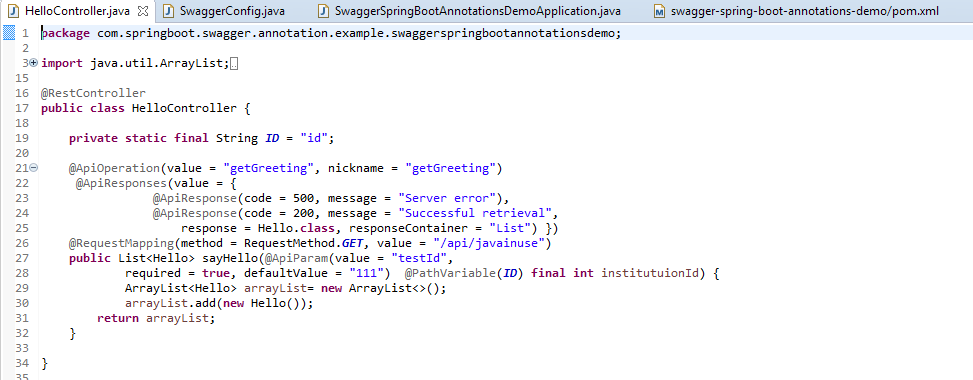
</plugin>

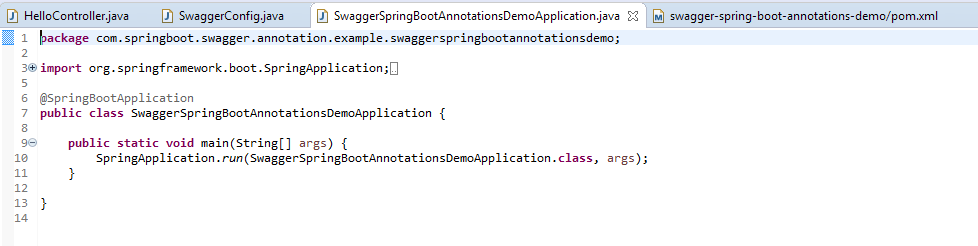
</plugins>

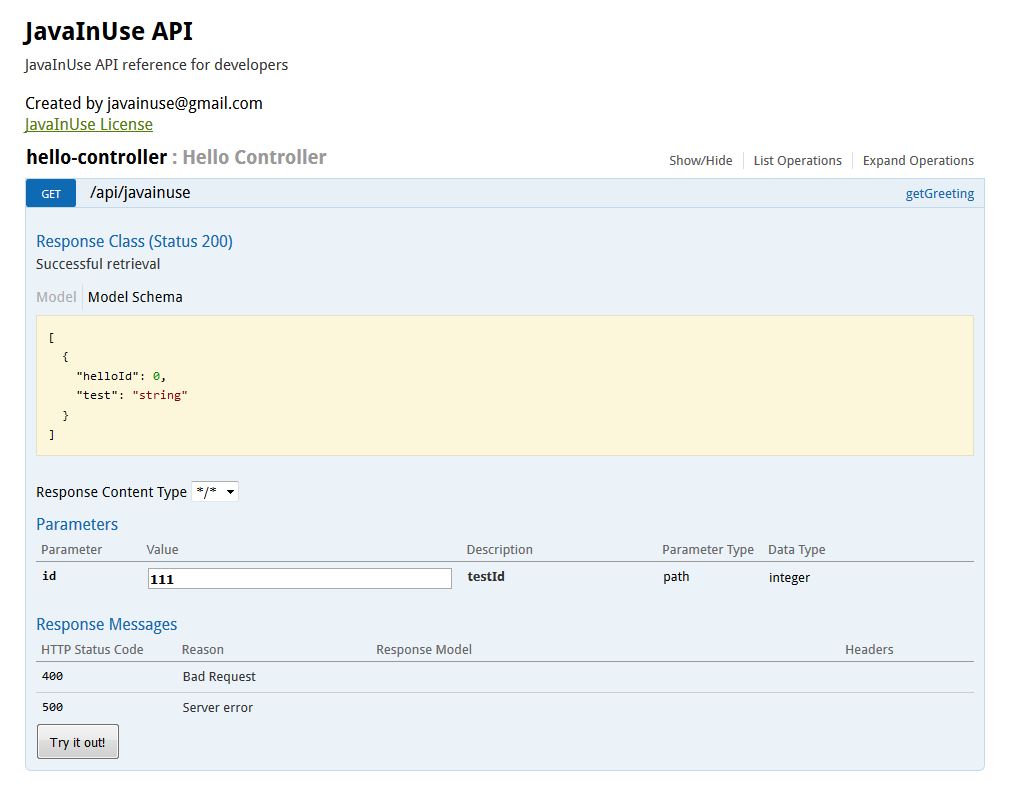
</build>

</project>

****

****

****



**Understanding the Annotations for the model class-**

The sayHello() method above is returning a list of Hello class. We can use annotations to define a default response that we expect-

* **@ApiModel-**

The @ApiModel allows you to manipulate the meta data of a model from a simple description or name change to a definition of polymorphism. We have used it to create a response class Hello with default values. It takes the following parameters-

|  |  |
| --- | --- |
| **Annotation Parameter** | **Description** |
| value | The name displayed for the Model class |
| description | The description of the model class |

* **@ApiModelProperty-**

The @ApiModelProperty allows controlling Swagger-specific definitions such as allowed values, and additional notes. It also offers additional filtering properties in case you want to hide the property in certain scenarios. We use this parameter for specifying default values to the Response model class Hello. It takes the following parameters-

|  |  |
| --- | --- |
| **Annotation Parameter** | **Description** |
| position | The position of the field in the response class during display using swagger. |
| value | The value of the field when using Swagger. For example the default value of the path variable will be 111 for class Hello. |
| required | If the field is optional or required. |

package com.javainuse.swaggertest;

import io.swagger.annotations.ApiModel;

import io.swagger.annotations.ApiModelProperty;

**@ApiModel**

public class Hello {

private int helloId;

private String test;

**@ApiModelProperty(position = 1, required = true, value = "1")**

public int getHelloId() {

return helloId;

}

public void setHelloId(int helloId) {

this.helloId = helloId;

}

**@ApiModelProperty(position = 2, required = true, value = "helloTest")**

public String getTest() {

return test;

}

public void setTest(String test) {

this.test = test;

}

}