Swagger

jim stafford

Fall 2022 v2022-07-15: Built: 2022-12-07 06:13 EST

Table of Contents

Chapter 1. Introduction

The core charter of this course is to introduce you to framework solutions in Java and focus on core Spring and SpringBoot frameworks. **Details** of Web APIs, database access, and distributed application design are core topics of other sibling courses. We have been covering a modest amount of Web API topics in these last set of modules to provide a functional front door to our application implementations. You know by now how to implement basic CRUD Web APIs. I now want to wrap up the Web API coverage by introducing a functional way to call those Web APIs with minimal work using Swagger UI. Detailed aspects of configuring Swagger UI is considered out of scope for this course but many example implementation details are included in the Swagger Contest Example set of applications in the examples source tree.

1.1. Goals

You will learn to:

- identify the items in the Swagger landscape and its central point — OpenAPI
- generate an Open API interface specification from Java code
- deploy and automatically configure a Swagger UI based on your Open API interface specification
- invoke Web API endpoint operations using Swagger UI

1.2. Objectives

At the conclusion of this lecture and related exercises, you will be able to:

1. generate an default Open API 3.0 interface specification using Springfox and Springdoc
2. configure and deploy a Swagger UI that calls your Web API using the Open API specification generated by your API
3. make HTTP CRUD interface calls to your Web API using Swagger UI
4. identify the starting point to make configuration changes to Springfox and Springdoc libraries

Chapter 2. Swagger Landscape

The core portion of the Swagger landscape is made up of a line of standards and products geared towards HTTP-based APIs and supported by the company SmartBear. There are two types of things directly related to Swagger: the OpenAPI standard and tools. Although heavily focused on Java implementations, Swagger is generic to all HTTP API providers and not specific to Spring.

2.1. Open API Standard

OpenAPI — is an implementation-agnostic interface specification for HTTP-based APIs. This was originally baked into the Swagger tooling but donated to open source community in 2015 as a way to define and document interfaces.

- Open API 2.0 - released in 2014 as the last version prior to transitioning to open source. This is equivalent to the Swagger 2.0 Specification.
- Open API 3.x - released in 2017 as the first version after transitioning to open source.

2.2. Swagger-based Tools

Within the close Swagger umbrella, there are a set of Tools, both free/open source and commercial that are largely provided by Smartbear.

- Swagger Open Source Tools - these tools are primarily geared towards single API at a time uses.
  - Swagger UI — is a user interface that can be deployed remotely or within an application. This tool displays descriptive information and provides the ability to execute API methods based on a provided OpenAPI specification.
  - Swagger Editor - is a tool that can be used to create or augment an OpenAPI specification.
  - Swagger Codegen - is a tool that builds server stubs and client libraries for APIs defined using OpenAPI.
- Swagger Commercial Tools - these tools are primarily geared towards enterprise usage.
  - Swagger Inspector - a tool to create OpenAPI specifications using external call examples
  - Swagger Hub - repository of OpenAPI definitions

SmartBear offers another set of open source and commercial test tools called SoapUI which is geared at authoring and executing test cases against APIs and can read in OpenAPI as one of its API definition sources.

Our only requirement in going down this Swagger path is to have the capability to invoke HTTP methods of our endpoints with some ease. There are at least two libraries that focus on generating the Open API spec and packaging a version of the Swagger UI to document and invoke the API in Spring Boot applications: Springfox and Springdocs.

2.3. Springfox

Springfox is focused on delivering Swagger-based solutions to Spring-based API implementations but is not an official part of Spring, Spring Boot, or Smartbear. It is hard to even find a link to Springfox on the Spring documentation web pages.

Essentially Springfox is:

- a means to generate Open API specs using Java annotations
- a packaging and light configuring of the Swagger-provided swagger UI

Springfox has been around many years. I found the initial commit in 2012. It supported Open API 2.0 when I originally looked at it in June 2020 (Open API 3.0 was released in 2017). At that time, the Webflux branch was also still in SNAPSHOT. However, a few weeks later a flurry of releases went out that included Webflux support but no releases have occurred in the year since then. It is not a fast evolving library.

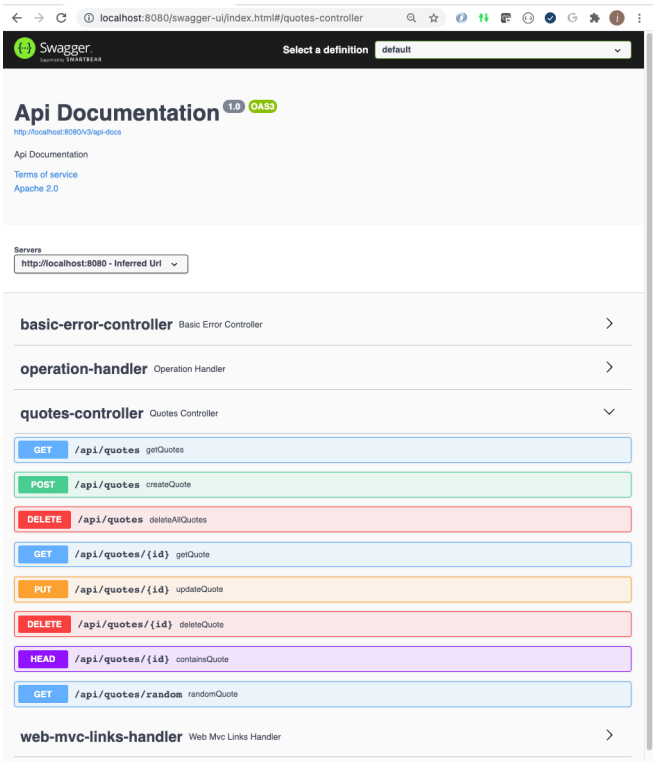


Figure 1. Example Springfox Swagger UI

Springfox does not work with >= Spring Boot 2.6

Springfox does not work with Spring Boot >= 2.6 where a patternParser was deprecated and causes an inspection error during initialization. We can work around the issue for demonstration — but serious use of Swagger (as of July 2022) is now limited to Springdoc.

2.4. Springdoc

Springdoc is an independent project focused on delivering Swagger-based solutions to Spring Boot APIs. Like Springfox, Springdoc has no official ties to Spring, Spring Boot, or Pivotal Software. The library was created because of Springfox's lack of support for Open API 3.x many years after its release.

Springdoc is relatively new compared to Springfox. I found its initial commit in July 2019 and has released several versions per month since. That indicates to me that they have a lot of catch-up to do to complete the product. However, they have the advantage of coming in when the standard is more mature and were able to bypass earlier Open API versions. Springdoc targets integration with the latest Spring Web API frameworks — including Spring MVC and Spring WebFlux.

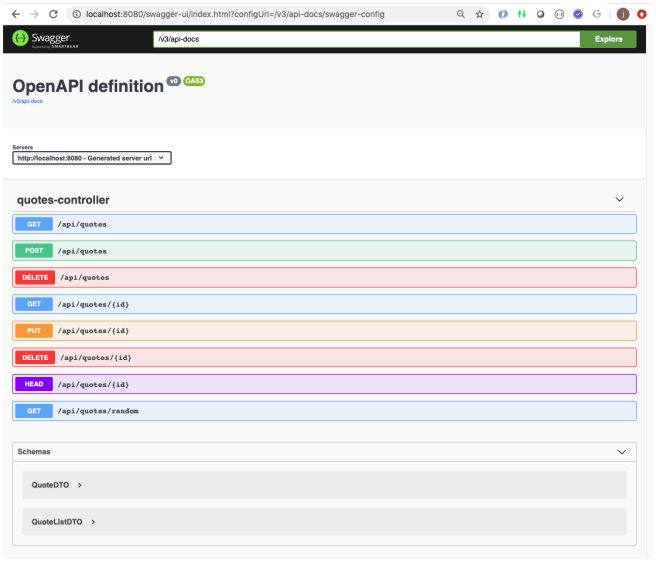


Figure 2. Example Springdoc SwaggerUI

Chapter 3. Minimal Configuration

My goal in bringing the Swagger topics into the course is solely to provide us with a convenient way to issue example calls to our API — which is driving our technology solution within the application. For that reason, I am going to show the least amount of setup required to enable a Swagger UI and have it do the default thing.

The minimal configuration will be missing descriptions for endpoint operations, parameters, models, and model properties. The content will rely solely on interpreting the Java classes of the controller, model/DTO classes referenced by the controller, and their annotations. Springdoc definitely does a better job at figuring out things automatically but they are both functional in this state.

3.1. Springfox Minimal Configuration

Springfox requires one change to a web application to support Open API 3 and the SwaggerUI:

- add Maven dependencies

3.1.1. Springfox Maven POM Dependency

Springfox requires two dependencies — which are both automatically brought in by the following starter dependency.

Springfox Maven POM Dependency

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-boot-starter</artifactId>
</dependency>
```

The starter automatically brings in the following dependencies — that no longer need to be explicitly named.

springfox-boot-starter Dependencies

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId> ①
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId> ②
</dependency>
```

① support for generating the Open API spec

② support for the Swagger UI

If you are implementing a module with only the DTOs or controllers and working to further define the API with annotations, you would only need the `springfox-swagger2` dependency.

3.1.2. Springfox Access

Once that is in place, you can access

- Open API spec: http://localhost:8080/v2/api-docs
- Swagger UI: http://localhost:8080/swagger-ui/index.html

The minimally configured Springfox will display more than what we want — but it has what we want.

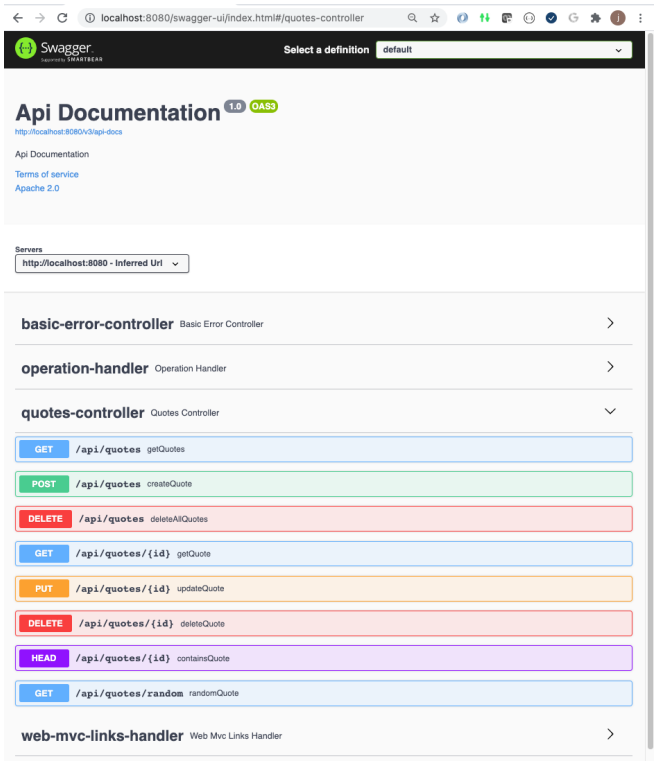


Figure 3. Minimally Configured SpringFox

3.1.3. Springfox Starting Customization

The starting point for adjusting the overall interface is done thru the definition of one or more Dockets. From here, we can control the path and dozens of other options. The specific option shown will reduce the operations shown to those that have paths that start with "/api/".

Springfox Starting Customization

```
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;

@Configuration
public class SwaggerConfig {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
                .select()
                .paths(PathSelectors.regex("/api/.*"))
                .build();
    }
}
```

Textual descriptions are primarily added to the annotations of each controller and model/DTO class.

3.2. Springdoc Minimal Configuration

Springdoc minimal configuration is as simple as it gets. All that is required is a single Maven dependency.

3.2.1. Springdoc Maven Dependency

Springdoc has a single top-level dependency that brings in many lower-level dependencies.

Springdoc Maven Dependency

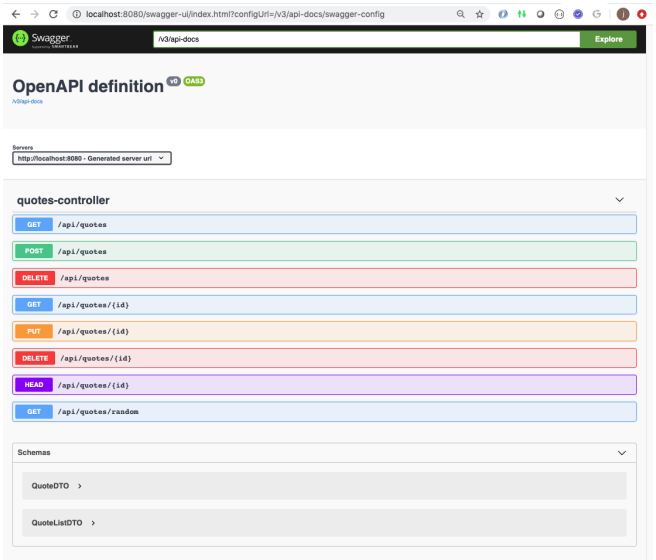
```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-ui</artifactId>
</dependency>
```

3.2.2. Springdoc Access

Once that is in place, you can access

- Open API spec: http://localhost:8080/v3/api-docs
- Swagger UI: http://localhost:8080/swagger-ui.html

The minimally configured Springdoc automatically filters out some of the Springboot overhead APIs and what we get is more tuned towards our developed API.



Avoid the Petstore

The `/swagger-ui.html` URI gets redirected to `/swagger-ui/index.html?configUrl=/v3/api-docs/swagger-config` when called. The `/swagger-ui/index.html` URI defaults to the Petstore application if `configUrl` is not supplied. Be sure to use the `/swagger-ui.html` URI when trying to serve up the current application or supply the `configUrl` parameter.

3.2.3. Springdoc Starting Customization

The starting point for adjusting the overall interface for Springdoc is done thru the definition of one or more GroupedOpenApi objects. From here, we can control the path and countless other options. The specific option shown will reduce the operations shown to those that have paths that start with

"/api/".

Springdoc Starting Customization

```
...
import org.springdoc.core.GroupedOpenApi;
import org.springdoc.core.SpringDocUtils;

@Configuration
public class SwaggerConfig {
   @Bean
    public GroupedOpenApi api() {
        SpringDocUtils.getConfig();
        //...
        return GroupedOpenApi.builder()
                .group("contests")
                .pathsToMatch("/api/**")
                .build();
    }
}
```

Textual descriptions are primarily added to the annotations of each controller and model/DTO class.

Chapter 4. Example Use

By this point in time we are past-ready for a live demo. You are invited to start both the Springfox and Springdoc version of the Contests Application and poke around. The following commands are being run from the parent `swagger-contest-example` directory. They can also be run within the IDE.

Start Springfox Demo App

```
$ mvn spring-boot:run -f springfox-contest-svc \①
-Dspring-boot.run.arguments=--server.port=8081 ②
```

① starts the web application from within Maven

② passes arguments from command line, thru Maven, to the Spring Boot application

Start Springdoc Demo App

```
$ mvn spring-boot:run -f springdoc-contest-svc \
-Dspring-boot.run.arguments=--server.port=8082 ①
```

① using a different port number to be able to compare side-by-side

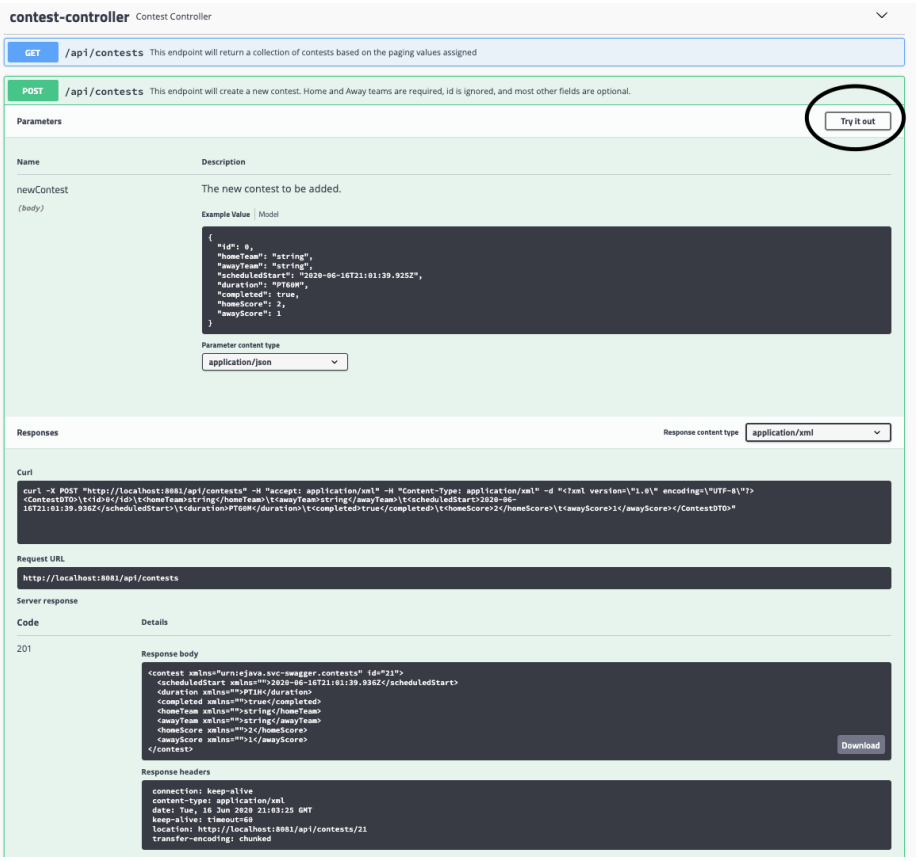
Access the two versions of the application using

- Springfox: http://localhost:8081/swagger-ui/index.html
- Springdoc: http://localhost:8082/swagger-ui.html

I will show an example thread here that is common to both.

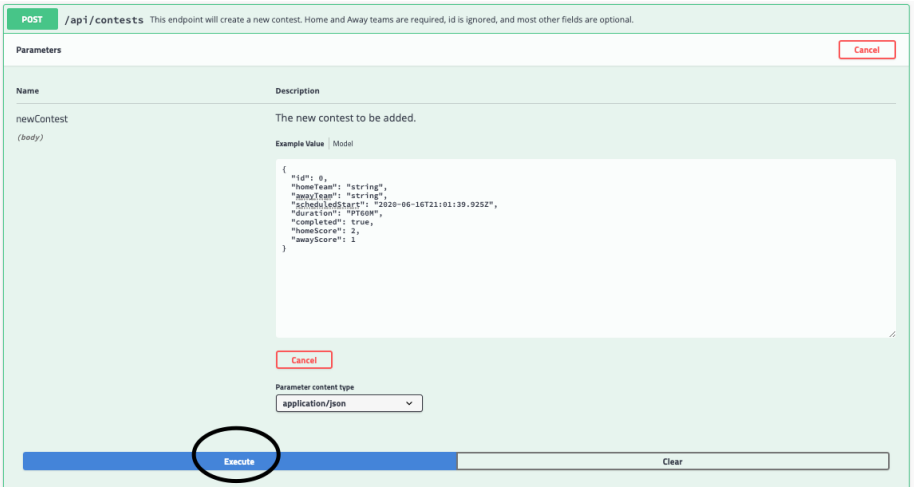
4.1. Access Contest Controller POST Command

1. click on the `POST /api/contests` line and the details of the operation will be displayed.
2. select a content type (`application/json` or `application/xml`)
3. click on the "Try it out" button.



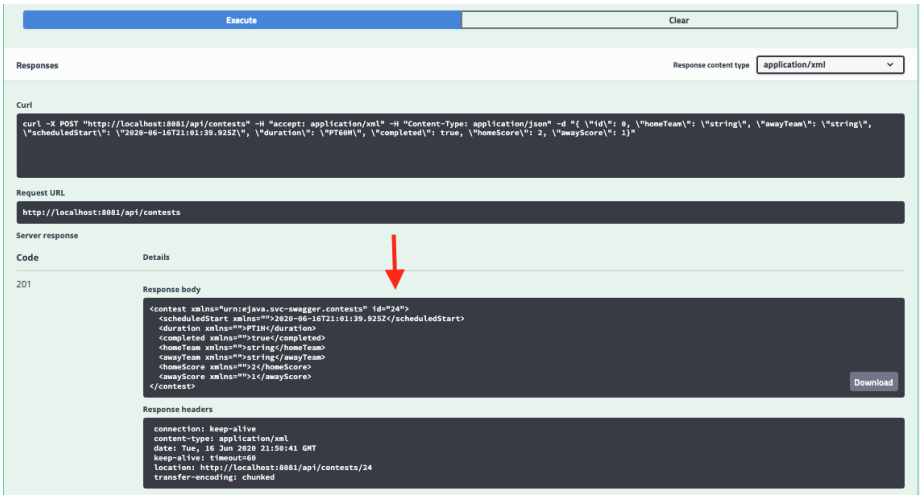
4.2. Invoke Contest Controller POST Command

1. Overwrite the values of the example with your values.
2. Select the desired response content type (`application/json` or `application/xml`)
3. press "Execute" to invoke the command



4.3. View Contest Controller POST Command Results

1. look below the "Execute" button to view the results of the command. There will be a payload and some response headers.
2. notice the payload returned will have an ID assigned
3. notice the headers returned will have a location header with a URL to the created contest
4. if your payload was missing a required field (e.g., home or away team), a 422/UNPROCESSABLE_ENTITY status is returned with a message payload containing the error text.



Chapter 5. Useful Configurations

I have created a set of examples under the `Swagger Contest Example` that provide a significant amount of annotations to add descriptions, provide accurate responses to dynamic outcomes, etc. for both Springfox and Springdoc to get a sense of how they performed.

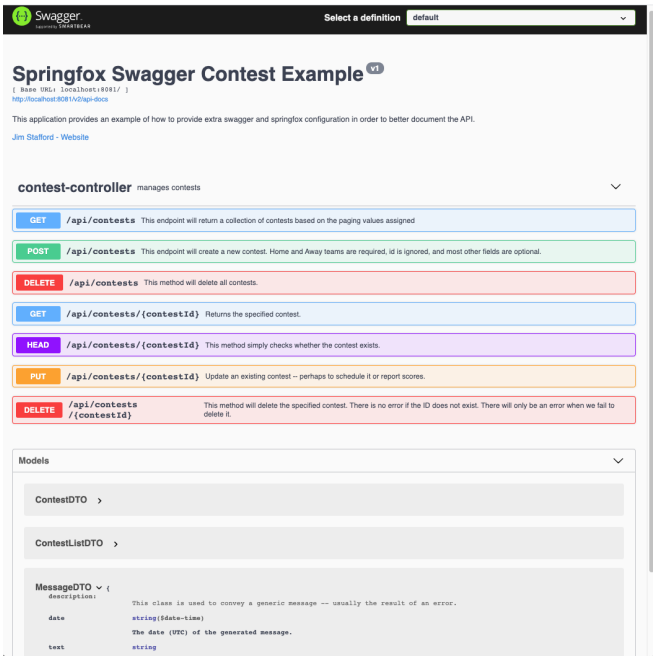


Figure 4. Fully Configured Springfox Example

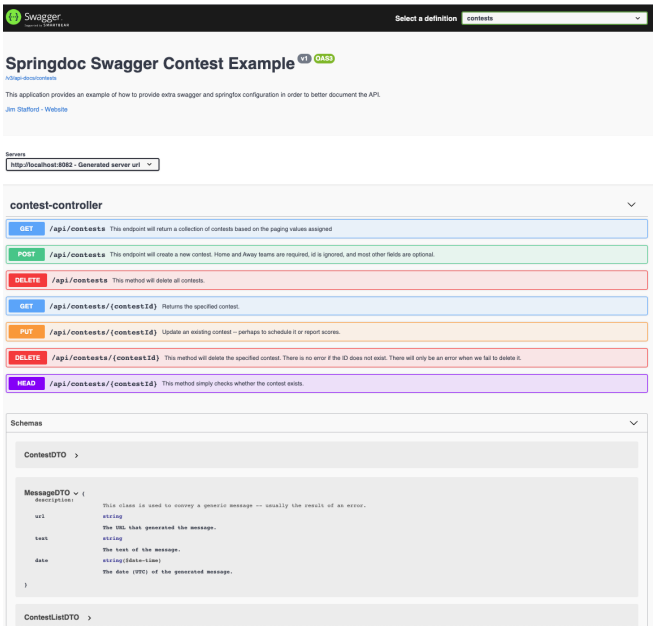


Figure 5. Fully Configured Springdoc Example

That is a lot of detail work and too much to cover here for what we are looking for. Feel free to look at the examples for details. However, I did encounter a required modification that made a feature go from unusable to usable and will show you that customization in order to give you a sense of how you might add other changes.

5.1. Customizing Type Expressions

`java.time.Duration` has a simple ISO string format expression that looks like `PT60M` or `PT3H` for periods of time.

5.1.1. OpenAPI 2 Model Property Annotations

The following snippet shows the Duration property enhanced with Open API 2 annotations to use a default `PT60M` example value.

ContestDTO Snippet with Duration and OpenAPI 2 Annotations

```
package info.ejava.examples.svc.springfox.contests.dto;

import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlProperty;
import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlRootElement;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
```

```
@JacksonXmlRootElement(localName="contest", namespace=ContestDTO.CONTEST_NAMESPACE)
@ApiModel(description="This class describes a contest between a home and, " +
        " away team, either in the past or future.")
public class ContestDTO {
    @JsonProperty(required = false)
    @ApiModelProperty(position = 4,
        example = "PT60M",
        value="Each scheduled contest should have a period of time specified " +
            "that identifies the duration of the contest. e.g., PT60M, PT2H")
    private Duration duration;
```

5.1.2. OpenAPI 3 Model Property Annotations

The following snippet shows the Duration property enhanced with Open API 3 annotations to use a default `PT60M` example value.

ContestDTO Snippet with Duration and OpenAPI 3 Annotations

```
package info.ejava.examples.svc.springdoc.contests.dto;

import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlProperty;
import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlRootElement;
import io.swagger.v3.oas.annotations.media.Schema;

@JacksonXmlRootElement(localName="contest", namespace=ContestDTO.CONTEST_NAMESPACE)
@Schema(description="This class describes a contest between a home and away team, "+
        "either in the past or future.")
public class ContestDTO {
    @JsonProperty(required = false)
    @Schema(example = "PT60M",
        description="Each scheduled contest should have a period of time specified "+
            "that identifies the duration of the contest. e.g., PT60M, PT2H")
    private Duration duration;
```

5.2. Duration Example Renderings

Both Springfox and Springdoc derive a more complicated schema either for JSON, XML, or both that was desired or usable.

Springfox has a complex definition for `java.util.Duration` for both JSON and XML.

Springfox Default Duration JSON Expression

```
  "duration": {
    "nano": 0,
    "negative": true,
    "seconds": 0,
    "units": [
      {
        "dateBased": true,
        "durationEstimated": true,
        "timeBased": true
      }
    ],
    "zero": true
  },
```

Springfox Default Duration XML Expression

```
<duration>
  <nano>0</nano>
  <negative>true</negative>
  <seconds>0</seconds>
  <units>
    <durationEstimated>
true</durationEstimated>
    <timeBased>true</timeBased>
  </units>
  <zero>true</zero>
</duration>
```

Springdoc has a fine default for JSON but a similar issue for XML.

Springdoc Default Duration JSON Expression

```
"duration": "PT60M",
```

Springdoc Default Duration XML Expression

```
<duration>
    <seconds>0</seconds>
    <negative>true</negative>
    <zero>true</zero>
    <units>
        <durationEstimated>
true</durationEstimated>
        <duration>
            <seconds>0</seconds>
            <negative>true</negative>
            <zero>true</zero>
            <nano>0</nano>
        </duration>
        <dateBased>true</dateBased>
        <timeBased>true</timeBased>
    </units>
    <nano>0</nano>
</duration>
```

We can correct the problem in Springfox by mapping the Duration class to a String. I originally found this solution for one of the other `java.time` types and it worked here as well.

Example Springfox Map Class to Alternate Type

```
@Bean
public Docket api(SwaggerConfiguration config) {
    return new Docket(DocumentationType.SWAGGER_2)
            .select()
```

```
            .paths(PathSelectors.regex("/api/.*"))
            .build()
            .directModelSubstitute(Duration.class, String.class)
            //...
            ;
}
```

With the above configuration in place, Springfox provides an example that uses a simple string to express ISO duration values.

Springfox Duration Mapped to String JSON Expression

```
"duration": "PT60M",
```

Springfox Duration Mapped to String XML Expression

```
<duration>PT60M</duration>
```

Judging by the fact that Springdoc is new — post Java 8 and expresses a Duration as a string for JSON, tells me there has to be a good solution for the XML side. I did not have the time to get a perfect solution, but found a configuration option that at least expressed the Duration as an empty string that was easy to enter in a value.

Example Springdoc Map Class to Alternate Type

```
@Bean
public GroupedOpenApi api(SwaggerConfiguration config) {
    SpringDocUtils.getConfig()
            .replaceWithSchema(Duration.class,
                    new Schema().example("PT120M")
            );
    return GroupedOpenApi.builder()
            .group("contests")
            .pathsToMatch("/api/contests/**")
            .build();
}
```

The examples below shows the configuration above improved the XML example without breaking the JSON example that we were targeting from the beginning. I purposely chose an alternate Duration value so we could see that the global configuration for property types is overriding the individual annotations.

Springfox Duration Mapped to String JSON Expression

```
  "duration": "PT120M",
```

Springfox Duration Mapped to String XML Expression

```
<duration>
</duration>
```

Chapter 6. Springfox / Springdoc Analysis

Both of these packages are surprisingly functional right out of the box with the minimal configuration — with the exception of some complex types. In early June 2020, Springdoc definitely understood the purpose of the Java code better than Springfox. That is likely because Springdoc is very much aware of Spring Boot 2.x and Springfox is slow to evolve.

The one feature I could not get to work in either — that I assume works — is "examples" for complex types. I worked until I got a legal JSON and XML example displayed but fell short of being able to supply an example that was meaningful to the problem domain (e.g., supplying team names versus "string"). A custom example is quite helpful if the model class has a lot of optional fields that are rarely used and unlikely to be used by someone using the Swagger UI.

(In early June 2020) Springfox had better documentation that shows you features ahead of time in logical order. Springdoc's documentation was primarily a Q&A FAQ that showed features in random order. I could not locate a good Springdoc example — but after implementing with Springfox first, the translation was extremely easy.

Springfox has been around a long time but with the change from Open API 2 to 3, the addition of Webflux, and their slow rate of making changes — that library will likely not be a good choice for Open API or Webflux users. Springdoc seems like it is having some learning pains — where features may work easier but don't always work 100%, lack of documentation and examples to help correct, and their existing FAQ samples do not always match the code. However, it seems solid already (in early June 2020) for our purpose and they are issuing many releases per month since they first commit in July 2019. By the time you read this much will have changed.

One thing I found after adding annotations for the technical frameworks (e.g., Lombok, WebMVC, Jackson JSON, Jackson XML) and then trying to document every corner of the API for Swagger in order to flesh out issues — it was hard to locate the actual code. My recommendation is to continue to make the names of controllers, models/DTO classes, parameters, and properties immediately understandable to save on the extra overhead of Open API annotations. Skip the obvious descriptions one can derive from the name and type, but still make it document the interface and usable to developers learning your API.

Chapter 7. Summary

In this module we:

- learned that Swagger is a landscape of items geared at delivering HTTP-based APIs
- learned that the company Smartbear originated Swagger and then divided up the landscape into a standard interface, open source tools, and commercial tools
- learned that the Swagger standard interface was released to open source at version 2 and is now Open API version 3
- learned that two tools — Springfox and Springdoc — are focused on implementing Open API for Spring and Spring Boot applications and provide a packaging of the Swagger UI.
- learned that Springfox and Springdoc have no formal ties to Spring, Spring Boot, Pivotal, Smartbear, etc. They are their own toolset and are not as polished as we have come to expect from the Spring suite of libraries.
- learned that Springfox is older, originally supported Open API 2 and SpringMVC for many years, but now supports Open API 3 and WebFlux
- learned that Springdoc is newer, active, and supports Open API 3, SpringMVC, and Webflux
- learned how to minimally configure Springfox and Springdoc into our web application in order to provide the simple ability to invoke our HTTP endpoint operations.