Quarkus - Using OpenAPI and Swagger UI

This guide explains how your Quarkus application can expose its API description through an OpenAPI specification and how you can test it via a user-friendly UI named Swagger UI.

Prerequisites

To complete this guide, you need:

- less than 15 minutes
- an IDE
- JDK 1.8+ installed with `JAVA_HOME` configured appropriately
- Apache Maven 3.6.3

Architecture

In this guide, we create a straightforward REST application to demonstrate how fast you can expose your API specification and benefit from a user interface to test it.

Solution

We recommend that you follow the instructions in the next sections and create the application step by step. However, you can skip right to the completed example.

Clone the Git repository: `git clone https://github.com/quarkusio/quarkus-quickstarts.git`, or download an archive.

The solution is located in the `openapi-swaggerui-quickstart` directory.

Creating the Maven project

First, we need a new project. Create a new project with the following command:

```
mvn io.quarkus:quarkus-maven-plugin:1.8.0.CR1:create \
    -DprojectGroupId=org.acme \
    -DprojectArtifactId=openapi-swaggerui-quickstart \
    -DclassName="org.acme.openapi.swaggerui.FruitResource" \
    -Dpath="/fruits" \
    -Dextensions="resteasy-jsonb"
cd openapi-swaggerui-quickstart
```

This command generates the Maven project with a `/fruits` REST endpoint.

Expose a REST Resource

We will create a `Fruit` bean and a `FruitResouce` REST resource (feel free to take a look to the Writing JSON REST services guide if your want more details on how to build a REST API with Quarkus).

```
package org.acme.openapi.swaggerui;

public class Fruit {

    public String name;
    public String description;

    public Fruit() {
    }

    public Fruit(String name, String description) {
        this.name = name;
        this.description = description;
    }
}
```

```java
package org.acme.openapi.swaggerui;

import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.DELETE;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.Consumes;
import javax.ws.rs.core.MediaType;
import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.Set;

@Path("/fruits")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class FruitResource {

    private Set<Fruit> fruits =
Collections.newSetFromMap(Collections.synchronizedMap(new
LinkedHashMap<>()));

    public FruitResource() {
        fruits.add(new Fruit("Apple", "Winter fruit"));
        fruits.add(new Fruit("Pineapple", "Tropical fruit"));
    }

    @GET
    public Set<Fruit> list() {
        return fruits;
    }

    @POST
    public Set<Fruit> add(Fruit fruit) {
        fruits.add(fruit);
        return fruits;
    }

    @DELETE
    public Set<Fruit> delete(Fruit fruit) {
        fruits.removeIf(existingFruit ->
existingFruit.name.contentEquals(fruit.name));
        return fruits;
    }
}
```

As we changed the API, we also need to update the test:

```java
package org.acme.openapi.swaggerui;

import io.quarkus.test.junit.QuarkusTest;
import org.junit.jupiter.api.Test;

import javax.ws.rs.core.MediaType;

import static io.restassured.RestAssured.given;
import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.Matchers.containsInAnyOrder;

@QuarkusTest
public class FruitResourceTest {

    @Test
    public void testList() {
        given()
                .when().get("/fruits")
                .then()
                .statusCode(200)
                .body("$.size()", is(2),
                        "name", containsInAnyOrder("Apple",
"Pineapple"),
                        "description", containsInAnyOrder("Winter
fruit", "Tropical fruit"));
    }

    @Test
    public void testAdd() {
        given()
                .body("{\"name\": \"Pear\", \"description\":
\"Winter fruit\"}")
                .header("Content-Type", MediaType.APPLICATION_JSON)
                .when()
                .post("/fruits")
                .then()
                .statusCode(200)
                .body("$.size()", is(3),
                        "name", containsInAnyOrder("Apple",
"Pineapple", "Pear"),
                        "description", containsInAnyOrder("Winter
fruit", "Tropical fruit", "Winter fruit"));

        given()
                .body("{\"name\": \"Pear\", \"description\":
\"Winter fruit\"}")
                .header("Content-Type", MediaType.APPLICATION_JSON)
                .when()
                .delete("/fruits")
```

```
                .then()
                .statusCode(200)
                .body("$.size()", is(2),
                        "name", containsInAnyOrder("Apple",
"Pineapple"),
                        "description", containsInAnyOrder("Winter
fruit", "Tropical fruit"));
    }
}
```

Expose OpenAPI Specifications

Quarkus proposes a `smallrye-openapi` extension compliant with the Eclipse MicroProfile OpenAPI specification in order to generate your API OpenAPI v3 specification.

You just need to add the `openapi` extension to your Quarkus application:

```
./mvnw quarkus:add-extension -Dextensions="quarkus-smallrye
-openapi"
```

This will add the following to your `pom.xml`:

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-smallrye-openapi</artifactId>
</dependency>
```

Now, we are ready to run our application:

```
./mvnw compile quarkus:dev
```

Once your application is started, you can make a request to the default `/openapi` endpoint:

```
$ curl http://localhost:8080/openapi
openapi: 3.0.3
info:
  title: Generated API
  version: "1.0"
paths:
  /fruits:
    get:
      responses:
        200:
          description: OK
          content:
            application/json: {}
    post:
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Fruit'
      responses:
        200:
          description: OK
          content:
            application/json: {}
    delete:
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Fruit'
      responses:
        200:
          description: OK
          content:
            application/json: {}
components:
  schemas:
    Fruit:
      properties:
        description:
          type: string
        name:
          type: string
```

> If you do not like the default endpoint location `/openapi`, you can change it by adding the following configuration to your `application.properties`:
>
> ```
> quarkus.smallrye-openapi.path=/swagger
> ```

Hit `CTRL+C` to stop the application.

Providing Application Level OpenAPI Annotations

There are some MicroProfile OpenAPI annotations which describe global API information, such as the following:

- API Title

- API Description

- Version

- Contact Information

- License

All of this information (and more) can be included in your Java code by using appropriate OpenAPI annotations on a JAX-RS `Application` class. Because a JAX-RS `Application` class is not required in Quarkus, you will likely have to create one. It can simply be an empty class that extends `javax.ws.rs.core.Application`. This empty class can then be annotated with various OpenAPI annotations such as `@OpenAPIDefinition`. For example:

```
@OpenAPIDefinition(
    tags = {
            @Tag(name="widget", description="Widget operations."),
            @Tag(name="gasket", description="Operations related to
gaskets")
    },
    info = @Info(
        title="Example API",
        version = "1.0.1",
        contact = @Contact(
            name = "Example API Support",
            url = "http://exampleurl.com/contact",
            email = "techsupport@example.com"),
        license = @License(
            name = "Apache 2.0",
            url = "http://www.apache.org/licenses/LICENSE-
2.0.html"))
)
public class ExampleApiApplication extends Application {
}
```

Another option, that is a feature provided by SmallRye and not part of the specification, is to use configuration to add this global API information. This way, you do not need to have a JAX-RS `Application` class, and you can name the API differently per environment.

For example, add the following to your `application.properties`:

```
mp.openapi.extensions.smallrye.info.title=Example API
%dev.mp.openapi.extensions.smallrye.info.title=Example API
(development)
%test.mp.openapi.extensions.smallrye.info.title=Example API (test)
mp.openapi.extensions.smallrye.info.version=1.0.1
mp.openapi.extensions.smallrye.info.description=Just an example
service
mp.openapi.extensions.smallrye.info.termsOfService=Your terms here
mp.openapi.extensions.smallrye.info.contact.email=techsupport@examp
le.com
mp.openapi.extensions.smallrye.info.contact.name=Example API
Support
mp.openapi.extensions.smallrye.info.contact.url=http://exampleurl.c
om/contact
mp.openapi.extensions.smallrye.info.license.name=Apache 2.0
mp.openapi.extensions.smallrye.info.license.url=http://www.apache.o
rg/licenses/LICENSE-2.0.html
```

This will give you similar information as the `@OpenAPIDefinition` example above.

Loading OpenAPI Schema From Static Files

Instead of dynamically creating OpenAPI schemas from annotation scanning, Quarkus also supports serving static OpenAPI documents. The static file to serve must be a valid document conforming to the OpenAPI specification. An OpenAPI document that conforms to the OpenAPI Specification is itself a valid JSON object, that can be represented in `yaml` or `json` formats.

To see this in action, we'll put OpenAPI documentation under `META-INF/openapi.yaml` for our `/fruits` endpoints. Quarkus also supports alternative OpenAPI document paths if you prefer.

```yaml
openapi: 3.0.1
info:
  title: Static OpenAPI document of fruits resource
  description: Fruit resources Open API documentation
  version: "1.0"

servers:
  - url: http://localhost:8080/openapi
    description: Optional dev mode server description

paths:
  /fruits:
    get:
      responses:
        200:
          description: OK - fruits list
          content:
            application/json: {}
    post:
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Fruit'
      responses:
        200:
          description: new fruit resource created
          content:
            application/json: {}
    delete:
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Fruit'
      responses:
        200:
          description: OK - fruit resource deleted
          content:
            application/json: {}
components:
  schemas:
    Fruit:
      properties:
        description:
          type: string
        name:
          type: string
```

By default, a request to `/openapi` will serve the combined OpenAPI document from the static file and the model generated from application endpoints code. We can however change this to only serve the static OpenAPI document by adding `mp.openapi.scan.disable=true` configuration into `application.properties`.

Now, a request to `/openapi` endpoint will serve the static OpenAPI document instead of the generated one.

> *About OpenAPI document paths*
>
> Quarkus supports various paths to store your OpenAPI document under. We recommend you place it under `META-INF/openapi.yml`. Alternative paths are:
>
> - `META-INF/openapi.yaml`
> - `META-INF/openapi.yml`
> - `META-INF/openapi.json`
> - `WEB-INF/classes/META-INF/openapi.yml`
> - `WEB-INF/classes/META-INF/openapi.yaml`
> - `WEB-INF/classes/META-INF/openapi.json`
>
> Live reload of static OpenAPI document is supported during development. A modification to your OpenAPI document will be picked up on fly by Quarkus.

Changing the OpenAPI version

By default, when the document is generated, the OpenAPI version used will be `3.0.3`. If you use a static file as mentioned above, the version in the file will be used. You can also define the version in SmallRye using the following configuration:

```
mp.openapi.extensions.smallrye.openapi=3.0.2
```

This might be useful if your API go through a Gateway that needs a certain version.

Auto-generation of Operation Id

The Operation Id can be set using the `@Operation` annotation, and is in many cases useful when using a tool to generate a client stub from the schema. The Operation Id is typically used for the method name in the client stub. In SmallRye, you can auto-generate this Operation Id by using the following configuration:

```
mp.openapi.extensions.smallrye.operationIdStrategy=METHOD
```

Now you do not need to use the `@Operation` annotation. While generating the document, the method name will be used for the Operation Id.

Table 1. The strategies available for generating the Operation Id

Property value	Description
METHOD	Use the method name.
CLASS_METHOD	Use the class name (without the package) plus the method.
PACKAGE_CLASS_METHOD	Use the class name plus the method name.

Use Swagger UI for development

When building APIs, developers want to test them quickly. Swagger UI is a great tool permitting to visualize and interact with your APIs. The UI is automatically generated from your OpenAPI specification.

The Quarkus `smallrye-openapi` extension comes with a `swagger-ui` extension embedding a properly configured Swagger UI page.

By default, Swagger UI is only available when Quarkus is started in dev or test mode.

If you want to make it available in production too, you can include the following configuration in your `application.properties`:

```
quarkus.swagger-ui.always-include=true
```

This is a build time property, it cannot be changed at runtime after your application is built.

By default, Swagger UI is accessible at `/swagger-ui`.

You can update this path by setting the `quarkus.swagger-ui.path` property in your `application.properties`:

```
quarkus.swagger-ui.path=/my-custom-path
```

The value `/` is not allowed as it blocks the application from serving anything else.

Now, we are ready to run our application:

```
./mvnw compile quarkus:dev
```

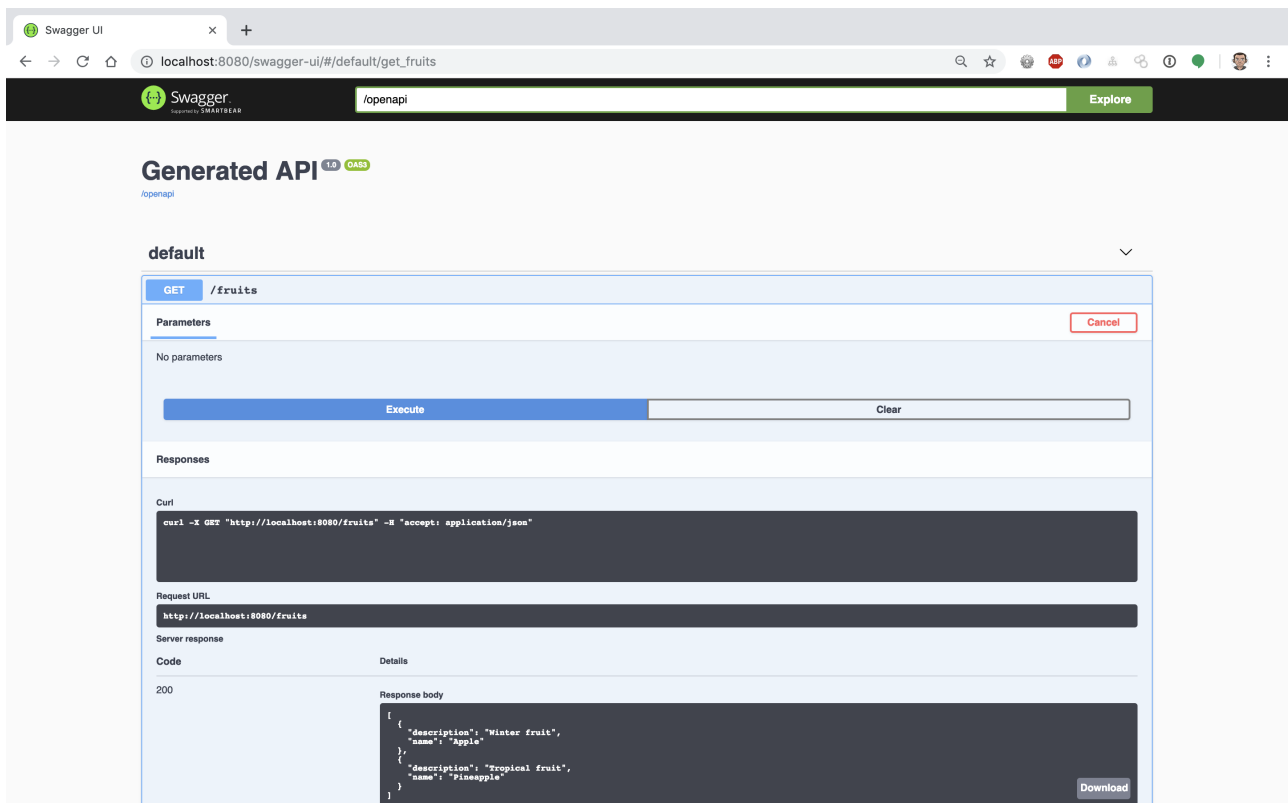You can check the Swagger UI path in your application's log:

```
00:00:00,000 INFO  [io.qua.swa.run.SwaggerUiServletExtension]
Swagger UI available at /swagger-ui
```

Once your application is started, you can go to http://localhost:8080/swagger-ui and play with your API.

You can visualize your API's operations and schemas.

You can also interact with your API in order to quickly test it.

Hit `CTRL+C` to stop the application.

Configuration Reference

OpenAPI

🔒 Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default
🔒 `quarkus.smallrye-openapi.path` The path at which to register the OpenAPI Servlet.	string	`/openapi`
🔒 `quarkus.smallrye-openapi.store-schema-directory` If set, the generated OpenAPI schema documents will be stored here on build. Both openapi.json and openapi.yaml will be stored here if this is set.	path	

Swagger UI

🔒 Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default
🔒 quarkus.swagger-ui.path The path where Swagger UI is available. The value / is not allowed as it blocks the application from serving anything else.	string	/swagger-ui
🔒 quarkus.swagger-ui.always-include If this should be included every time. By default this is only included when the application is running in dev mode.	boolean	false
🔒 quarkus.swagger-ui.enable If Swagger UI should be enabled. By default, Swagger UI is enabled.	boolean	true