Specification Builder

In this lesson, we would learn about creating specification in Rest Assured.

We'll cover the following

What is a specification?

In order to avoid duplicate request parameters and/or response expectations for multiple tests, we can create specification objects which are reusable.

There are two types of specification builder Java classes as mentioned below:

1. *RequestSpecBuilder*
2. *ResponseSpecBuilder*

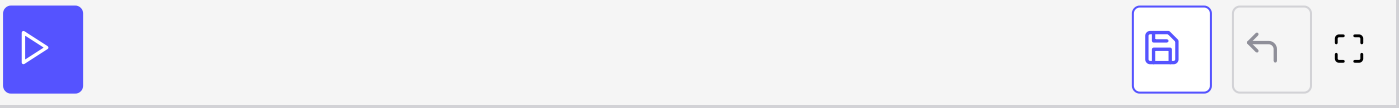
`RequestSpecification`

This is used when a few common parameters are needed for multiple and/or different tests while creating a request.

Let's understand this concept better with the help of a sample code snippet for creating a `RequestSpecification` using *RequestSpecBuilder*.

```
import static org.hamcrest.Matchers.anything;
import static org.hamcrest.Matchers.is;


import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.testng.annotations.Test;
import io.restassured.RestAssured;
import io.restassured.builder.RequestSpecBuilder;
import io.restassured.builder.ResponseSpecBuilder;
import io.restassured.response.Response;
import io.restassured.specification.RequestSpecification;
import io.restassured.specification.ResponseSpecification;
```

```
public class APIDemo {

	private static Logger LOG = LoggerFactory.getLogger(APIDemo.class);

	@Test
	public void test_RequestSpecificationWithQueryParam() {

		LOG.info("Step - 2 : Make a get() call using RequestSpecification to fetch John's
		Response response = RestAssured
							.given()
							.spec(getRequestSpecification())
							.queryParam("first_name", "John")
							.when()
							.get();

		LOG.info("Step - 3 : Print the JSON response body");
		response.getBody().prettyPrint();

	}

	@Test
	public void test_RequestSpecification() {

		LOG.info("Step - 4 : Make a get() call using RequestSpecification to fetch all Stu
		Response response = RestAssured
							.given()
							.spec(getRequestSpecification())
							.when()
							.get();

		LOG.info("Step -5 : Print the JSON response body");
		response.getBody().prettyPrint();

	}



	// Helper method
    public RequestSpecification getRequestSpecification() {

		LOG.info("Step - 1 : Create RequestSpecification using  RequestSpecBuilder ");
		RequestSpecBuilder builder = new RequestSpecBuilder();

		builder.setBaseUri ("http://ezifyautomationlabs.com:6565");
		builder.setBasePath ("/educative-rest/students");

		RequestSpecification requestSpec = builder.build();
		return requestSpec;
	}

}
```



Understanding the example code above:

- The method `getRequestSpecification()` creates and returns an object of `RequestSpecification` which can be reused in multiple tests. It needs the same

attributes like - base URI, base path and query param.

- The code below sets the `RequestSpecification` object using `spec(...)` method in the request builder while making a `get()` call.

```
Response response = RestAssured.given().spec(getRequestSpecification()).wh
en().get();
```

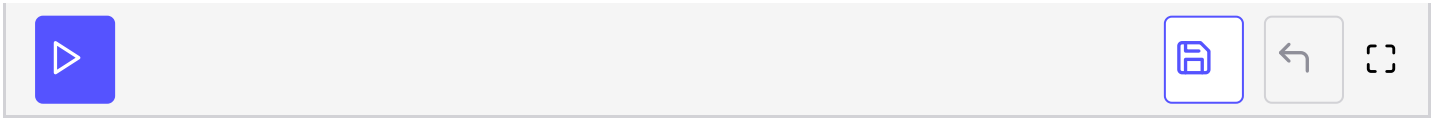
`ResponseSpecification`

This is used to validate a common response or a response needed for multiple tests from the body. We can also merge additional body expectations must all be fulfilled for the test to pass.

Let's get a better understanding of how this works with the example code for creating a `ResponseSpecification` using ResponseSpecBuilder.

```
import static org.hamcrest.Matchers.anything;
import static org.hamcrest.Matchers.is;


import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.testng.annotations.Test;
import io.restassured.RestAssured;
import io.restassured.builder.RequestSpecBuilder;
import io.restassured.builder.ResponseSpecBuilder;
import io.restassured.response.Response;
import io.restassured.specification.RequestSpecification;
import io.restassured.specification.ResponseSpecification;

public class APIDemo {

        private static Logger LOG = LoggerFactory.getLogger(APIDemo.class);

   @Test
        public void test_ResponseSpecification1() {

                String url = "http://ezifyautomationlabs.com:6565/educative-rest/students";

                LOG.info("Step - 2 : Make a get() call using ResponseSpecification and validate st
                RestAssured
                .when()
                .get(url)
                .then()
                .spec(getResponseSpecification());
        }


        @Test
        public void test_ResponseSpecification2() {

                String url = "http://ezifyautomationlabs.com:6565/educative-rest/students";
```

```
                LOG.info("Step - 3 : Make a get() call using ResponseSpecification and validate st
                RestAssured

                .when()
                .get(url)
                .then()
                .spec(getResponseSpecification())
                .body("John", is(anything()));
        }

        // Helper method
        public ResponseSpecification getResponseSpecification() {

                LOG.info("Step - 1 : Create ResponseSpecification using ResponseSpecBuilder ");
                ResponseSpecBuilder builder = new ResponseSpecBuilder();
                builder.expectStatusCode(200);

                ResponseSpecification responseSpec = builder.build();
                return responseSpec;
        }
}
```



Understanding the example code above:

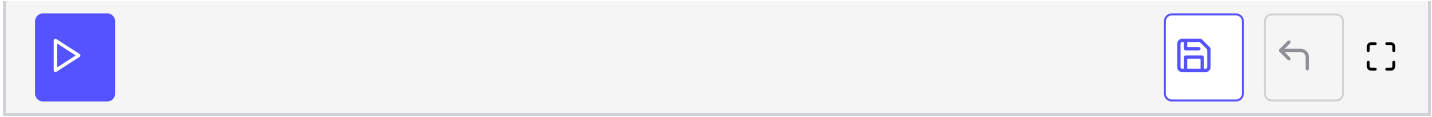
- The method `getResponseSpecification()` creates and returns an object of `ResponseSpecification` which can be reused in multiple tests to validate a response body like; statuscode is `200`.
- The code below gets the response and validates it using the `ResponseSpecification` criteria and an external body validation. The object `ResponseSpecification` is set using the `spec(...)` method in the request builder while making a `get()` call.

```
RestAssured.when().get(url).then().spec(getResponseSpecification()).body(
"John", is(anything()));
```

`RequestSpecification` and `ResponseSpecification` combined

```
import static org.hamcrest.Matchers.anything;
import static org.hamcrest.Matchers.is;


import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.testng.annotations.Test;
import io.restassured.RestAssured;
import io.restassured.builder.RequestSpecBuilder;
import io.restassured.builder.ResponseSpecBuilder;
import io.restassured.response.Response;
```

```
import io.restassured.specification.RequestSpecification;
import io.restassured.specification.ResponseSpecification;


public class APIDemo {

        private static Logger LOG = LoggerFactory.getLogger(APIDemo.class);

   @Test
        public void test_combineRequestResponseSpecification() {

                LOG.info("Step - 1 : Make a get() call using ResponseSpecification and ResponseSpe
                RestAssured
                .given()
                .spec(getRequestSpecification())
                .when()
                .get()
                .then()
                .spec(getResponseSpecification());
        }

    /**
        * Helper methods for creating RequestSpecification and ResponseSpecification
        **/
        public RequestSpecification getRequestSpecification() {

                LOG.info("Step - 2 : Create RequestSpecification using  RequestSpecBuilder ");
                RequestSpecBuilder builder = new RequestSpecBuilder();

                builder.setBaseUri ("http://ezifyautomationlabs.com:6565");
                builder.setBasePath ("/educative-rest/students");
                builder.addQueryParam("gender", "male");

                RequestSpecification requestSpec = builder.build();
                return requestSpec;
        }

        public ResponseSpecification getResponseSpecification() {

                LOG.info("Step - 3 : Create ResponseSpecification using ResponseSpecBuilder ");
                ResponseSpecBuilder builder = new ResponseSpecBuilder();
                builder.expectStatusCode(200);

                ResponseSpecification responseSpec = builder.build();
                return responseSpec;
        }
}
```



The code above demonstrates how to use both `RequestSpecification` and `ResponseSpecification` while making a `get()` call. The specification creation is similar to what we have already discussed above.

That's all about specification. In the next lesson, we will learn how to upload a file using `Rest Assured`.