Integration Test - REST

In this lesson, we will learn how to write an automated integration test using RESTful APIs for our demo web API.

We'll cover the following

Integration scenario

In this scenario, we are simulating and automating the integration flow for a single service.

We will use our demo REST API to automate the below integration flow as follows:

1. Create a new `Student`
2. Verify that `Student` is created
3. Search the newly-created `Student` by `id`
4. Verify the search result
5. Delete the created `Student`
6. Verify the `Student` has been deleted

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import org.testng.annotations.Test;
import static org.testng.Assert.assertEquals;
import static org.testng.Assert.assertTrue;

import com.fasterxml.jackson.annotation.JsonProperty;
import io.restassured.RestAssured;
import io.restassured.response.ResponseOptions;

public class StudentIntegrationTest {

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

        @Test
        public void testIntegrationFlow() {
            //Target resource
        String url = "http://ezifyautomationlabs.com:6565/educative-rest/students";
        // Message body object.

        Student body = new Student("David", "Paul", "Male");

            //Step 1 - Create a new Student
        ResponseOptions<?> create = RestAssured.given()
                .header("accept", "application/json")
                .header("content-type", "application/json")
                .body(body)
                .post(url)
                                .andReturn();

        //print response message body in JSON format
        LOG.info("Server response for created Student record : " +create.getBody().prettyPrint());

                //Step -2 Assert that request was successful
        assertTrue(create.statusCode() == 201);
                long createdStudentId = create.getBody().jsonPath().getLong("id");

                //Step 3 - fetch the newly created Student details
        ResponseOptions<?> fetch = RestAssured.given()
                        .get(url + "/" + createdStudentId)
                                .andReturn();

        //print response message body in JSON format
        LOG.info("fetch newly created Student's record with id "+createdStudentId+  ":" +fetch.getB

                //Step -4 parse the response data and verify and assert fetch Student details
        assertTrue(fetch.statusCode() == 200);
                long studentID = fetch.getBody().jsonPath().getLong("id");
                assertEquals(createdStudentId,studentID);

                //Step 5 - Delete the newly created Student Object
                 LOG.info("Delete Student with id "+studentID+"'s record");
                 ResponseOptions<?> delete = RestAssured.given()
                                .delete(url + "/" + studentID);
                //Assert request was successful
            assertTrue(delete.statusCode() == 204);

                // Step 6 - Try getting Deleted  Student's record
        ResponseOptions<?> deletedStudent = RestAssured.given()
                        .get(url + "/" + createdStudentId)
                                .andReturn();

        //Assert record is deleted - NO record found
        LOG.info("HTTP GET response statusLine of deleted record "+deletedStudent.getStatusLine())
            assertTrue(deletedStudent.statusCode() == 404);
        }

}

// This POJO class will be used for serialization and deserialzation of the data
class Student {

    public Student(String firstName, String lastName, String gender) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.gender = gender;
    }

        @JsonProperty("id")
    Long id;

    @JsonProperty("first_name")
    String firstName;

    @JsonProperty("last_name")
    String lastName;

    @JsonProperty("gender")
    String gender;
}
```

Understanding the code

Since the code above has enough in-line comments, we will only discuss some important code snippets here.

- Uses `TestNG` for writing the test method
- **Creates a new `Student`**

  ```
  ResponseOptions<?> create =  RestAssured.given()
          .header("accept", "application/json")
          .header("content-type", "application/json")
          .body(body)
          .post(url)
          .andReturn();
  ```

  Here, the `ResponseOptions` object stores the response instance which can be used to fetch the response data like headers and messages.

  Two request headers are sent, `accept` and `content-type` along with the message body in the `POST` request.

- **Prints a response message body in JSON format**

  ```
  create.getBody().prettyPrint()
  ```

  Get the response body from the `ResponseOptions` object and prints it in JSON format using the `prettyPrint()` method

- **Gets the details of the newly-created student**

  ```
  ResponseOptions<?> fetch = RestAssured.given()
  ```

```
ResponseOptions<?> fetch = RestAssured.given()
                            .get(url + "/" + createdStudentId)
                            .andReturn();
```

Here, we have appended the `id` in the URL, it is called path param. Now, the requested resource will look for the student with `id` stored in a variable called `createdStudentId`.

- **Deletes the newly-created Student**

```
ResponseOptions<?> delete = RestAssured.given()
                             .delete(url + "/" + studentID)
                             .andReturn();
```

Here, we have appended the `id` in the URL, it is called path param and it identifies the resource to be deleted.

- **Verifies the student record has been deleted**

```
ResponseOptions<?> deletedStudent = RestAssured.given()

                           .get(url + "/" + createdStudentId)
                           .andReturn();
assertTrue(deletedStudent.statusCode() == 404);
```

Here, we are fetching the same student with the given id and the server returned status code `404` which means record not found, as expected. The method `statusCode()` returns the HTTP status code.

In the next lesson, we will learn to write an automated integration test for a SOAP use case.