DELETE Request

In this lesson, we will learn how to automate a DELETE request for an existing record.

We'll cover the following

- HTTP DELETE request automation
  - Example 1 – DELETE request for a particular id
  - Example 2 – DELETE request for a non-existing Student id

`HTTP DELETE` request automation

In this lesson, we will discuss deleting a record using the `DELETE` request method.

Example 1 – *DELETE request for a particular id*

- `HTTP` Method: *DELETE*
- Target URL: `http://ezifyautomationlabs.com:6565`
- Resource path: *`/educative-rest/students/{id}`*
- Take a look at the code below:

```
import static org.testng.Assert.assertTrue;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.testng.annotations.Test;

import io.restassured.RestAssured;
import io.restassured.response.Response;

import com.fasterxml.jackson.annotation.JsonProperty;

public class DELETERequestTest {

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

        @Test
        public void testDelete() {

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

                LOG.info("Step - 1 : Create a new Student [POST]");
                Student body = new Student("NewUser1", "DeleteUser", "Female");
                Response response = RestAssured.given().header("accept", "application/json")
                                .header("content-type", "application/json").body(body).post(url).a
                LOG.info("Created Student Record");
                response.getBody().prettyPrint();

                String id = response.getBody().jsonPath().getString("id");
                LOG.info("Get the created Student ID: " + id);

                LOG.info("Step - 2 : Delete the created record. [DELETE ]");
                String url1 = url + "/" + id;
                Response response1 = RestAssured.given().delete(url1).andReturn();

                LOG.info("Step - 3 : Print the response message and assert the status");
                LOG.info("Response Status Code: " + response1.getStatusCode());
                assertTrue(response1.getBody().prettyPrint().isEmpty());
                assertTrue(response1.getStatusCode()==204);
                LOG.info("Student with id: " +id+ " is deleted");
        }

}
         // 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;
        }
```

Let’s understand this example code.

The code above uses the `TestNG` and `Rest Assured` libraries for automating the `HTTP` DELETE request and sends a Student `id` which has to be deleted.

- **Step 1** – create a new Student using the `POST` request

```
Student body = new Student("NewUser1", "DeleteUser", "Female");
Response response = RestAssured.given()
                    .header("accept", "application/json")
                    .header("content-type", "application/json")
                    .body(body)
```

```
                        .body(body)
                        .post(url)
                        .andReturn();

LOG.info("Created Student Record");
response.getBody().prettyPrint();
```

- **Step 2** – gets the created Student `id` and `Delete` that Student’s record using `delete(url1)` method and returns the `Response` object using `andReturn()` method

```
String id = response.getBody().jsonPath().getString("id");
LOG.info("Get the created Student ID: " + id);

LOG.info("Step - 2 : Delete the created record. [DELETE ]");
String url1 = url + "/" + id;
Response response1 = RestAssured.given().delete(url1).andReturn();
```

- **Step 3** – verifies that the Student’s record is deleted and the `Response` body is empty. `response1.getBody().prettyPrint().isEmpty()` and the response status code is `204`

```
LOG.info("Step - 3 : Print the response message and assert the status");
LOG.info("Response Status Code: " + response.getStatusCode());
assertTrue(response1.getBody().prettyPrint().isEmpty());
assertTrue(response1.getStatusCode()==204);
LOG.info("Student with id: " +id+ " is deleted");
```

Example 2 – *DELETE request for a non-existing Student id*

- `HTTP` Method: *DELETE*
- Target URL: `http://ezifyautomationlabs.com:6565`
- Resource path: `/educative-rest/students/{id}`
- Take a look at the code below:

```
import static org.testng.Assert.assertTrue;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.testng.annotations.Test;

import io.restassured.RestAssured;
import io.restassured.response.Response;

import com.fasterxml.jackson.annotation.JsonProperty;
```

```
public class DELETERequestTest {

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


        @Test
        public void testDeleteNonExistingStudentId() {

        String url = "http://ezifyautomationlabs.com:6565/educative-rest/students";
        String id = "48";
                LOG.info("Step - 2 : Delete the created record. [DELETE ]");
                String url1 = url + "/" + id;
                Response response1 = RestAssured.given().delete(url1).andReturn();

                LOG.info("Step - 3 : Print the response message and assert the status");
                LOG.info("Response Status Code: " + response1.getStatusCode());
                assertTrue(response1.getBody().prettyPrint().isEmpty());
                assertTrue(response1.getStatusCode()==404);
        }

}
```

Let’s understand this above example code.

The code uses the `TestNG` and `Rest Assured` libraries to automate the `HTTP` DELETE request and sends a non-existing Student `id` for deletion.

In this case, the server response will have an error message with status code `404`.

```
{
    "timestamp": "2020-05-01T08:23:54.599+0000",
    "status": 404,
    "error": "Not Found",
    "message": "Student does not exist",
    "path": "/educative-rest/students/48"
}
```

Let’s go through a quick quiz to test your understanding of request methods.