POST Request

In this lesson, we will learn how to automate a POST request for different use cases.

We'll cover the following

In this lesson, we will discuss two variations of the POST Request:

1. POST with string body
2. POST request using POJO (*Plain Old Java Object*) class

`HTTP` `POST` request automation

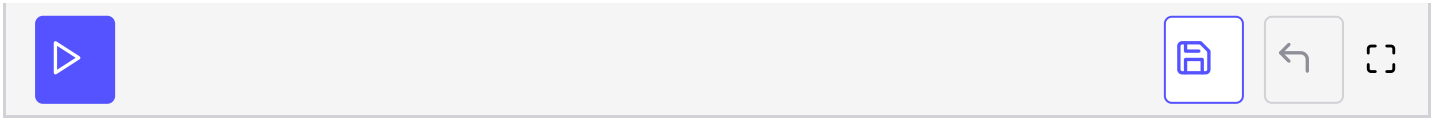
In this lesson, we will discuss two variations of POST Request.

Example 1 – POST with a string body

- `HTTP` Method: *POST*
- Target URL: `http://ezifyautomationlabs.com:6565`
- Resource path: `/educative-rest/students`
- Message body: *As a String Object*
- 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;


public class POSTRequestTest {

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

```
        @Test
        public void testPOSTStringBody() {


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

        LOG.info("Step - 1 : Target resource ( server ) : " + url);

        String body = "{\"first_name\": \"Jack\", \"last_name\": \"Preacher\", \"gender\": \"Male\
        LOG.info("Step - 2 : Message body: " + body);

        LOG.info("Step - 3 : Send a POST Request");
        Response response = RestAssured.given()
                .header("accept", "application/json")
                .header("content-type", "application/json")
                .body(body)
                .post(url)
                .andReturn();

        LOG.info("Step - 4 : Print the response message and assert the status response code is 201
        response.getBody().prettyPrint();
                assertTrue(response.getStatusCode() == 201);

        }
}
```



Let's understand the example code above.

The code above uses the `TestNG` and `Rest Assured` libraries for automating the `HTTP` POST request and sends a string message for creating a new resource.

- **Step 1** – the target `URL`

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

- **Step 2** – the request message body as a `String`

```
String body = "{\"first_name\": \"Jack\", \"last_name\": \"Preacher\", \"g
ender\": \"Male\" }";
```

- **Step 3** – makes a *POST* Request with `accept` and `content-type` headers along with a message `body` and returns a `Response` object

```
Response response = RestAssured.given()
                .header("accept", "application/json")
                .header("content-type", "application/json")
                .body(body)
```

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

- **Step 4** – logs the response body in `JSON` format and assert response status code as `201`

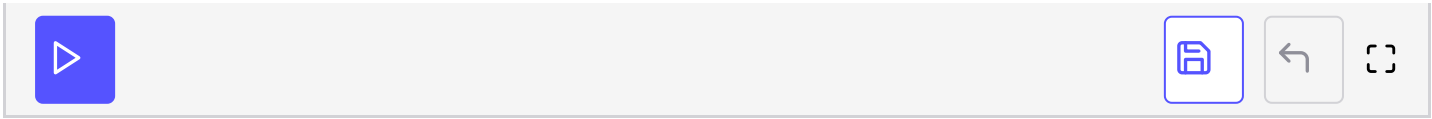
```
response.getBody().prettyPrint();
assertTrue(response.getStatusCode()==201);
```

Example 2 – POST request using POJO class

- `HTTP` Method: *POST*
- Target URL: `http://ezifyautomationlabs.com:6565`
- Resource path: *`/educative-rest/students`*
- Message Body: *As a `Java` Object*
- 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 POSTRequestTest {

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

        @Test
        public void testPOSTusingPOJO() {

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

        LOG.info("Step - 1 : Target resource ( server ) : " + url);

        Student body = new Student("Katherine", "AK", "Female");
        LOG.info("Step - 2 : Message body: " + body);

        LOG.info("Step - 3 : Send a POST Request");
        Response response = RestAssured.given()
                .header("accept", "application/json")
                .header("content-type", "application/json")
                .body(body)
                .post(url)
                .andReturn();

        LOG.info("Step - 4 : Print the response message and assert the status response code is 201
        response.getBody().prettyPrint();
                assertTrue(response.getStatusCode() == 201);
```

```
        }
}

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 uses the `TestNG` and `Rest Assured` libraries for automating the `HTTP` POST request and sends a `Java` Object (POJO) in the message body for creating a new resource.

- **Step 1** – the target `URL`

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

- **Step 2** – the request message body as a `Java` object (`Student` class object)

```
Student body = new Student("Katherine", "AK", "Female");
```

- **Step 3** – makes a *POST* Request using `post(url)` method with `accept` and `content-type` headers using `header(key,value)` method along with message `body` using `body(body)` method and returns a `Response` object using `andReturn()`

```
Response response = RestAssured.given()
                .header("accept", "application/json")
```

```
                .header("content-type", "application/json")
                .body(body)

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

- **Step 4** – logs the response body in `JSON` format and assert response status code as `201`

```
response.getBody().prettyPrint();
assertTrue(response.getStatusCode()==201);
```

- There is a `Student` class that has a few fields annotated with `@JsonProperty`. This is a marker annotation to define logical property to be used in serialization and deserialization of `JSON`.

```
public 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;
}
```

In the next lesson, we'll learn about automating the HTTP PUT request.