**RestAssured Automation : sample code**

RestAssured.baseURI = “https://google.com/createAccount”;

RestAssured.port = 443;

RestAssured.reset();

Response response =

RestAssured.given()

.config()

.contentType(ContentType.JSON)

.header(“Content-type”, “application/json”)

.and()

.expect()

.statusCode(400)

. body("lotto.lottoId", equalTo(6))

.param()

.body()

.log().ifValidationFails()

.when()

.request("GET", "/users/eugenp")

.get()

.put()

.post()

.headers(“Content-type”, “application.json”)

.param()

.delete(“/posts/1”)

.patch(“/posts/1”)

.then()

.extract()

.response()

.statusCode(200)

.getCoockies()

.assertThat()

.log()

.log().ifError();

.body("data.leagueId", equalTo(35));

.body("$", hasItems(1, 2, 3));

.body("", hasItems(1, 2, 3));

.log().ifStatusCodeIsEqualTo(500);

.body("odd.ck", equalTo(12.2f));

.log().ifStatusCodeMatches(greaterThan(200));

.log().ifValidationFails()

.time()

.timeIn(TimeUnit.SECONDS)

.path()

**Rest Assured ToolsQA sample code:**

**package** pages;

**import** org.testng.Assert;

**import** org.testng.annotations.Test;

**public** **class** RestAssuredMethods {

@Test

**public** **void** GetBookDetails() {

// Specify the base URL to the RESTful web service

RestAssured.baseURI = "https://demoqa.com/BookStore/v1/Books";

// Specify the request to be sent to the server

RequestSpecification httpRequest = RestAssured.given();

//Store the response

Response response = httpRequest.get("end-point");

// Get response content as String.

System.***out***.println("Response Body is => " + response.asString());

// Get status line from Response store into a variable called statusLine

String statusLine = response.getStatusLine();-

Assert.*assertEquals*(statusLine, "HTTP/1.1 200 OK", "inCorrect status code returned");

Assert.*assertTrue*(statusLine.contains(“OK"));

System.***out***.println("Status received => " + response.getStatusLine());

System.***out***.println("Response=>" + response.asPrettyPrint());

// Get the status code of the request.

//If request is successful, status code will be 200

**int** statusCode = response.getStatusCode();

// Assert that correct status code is returned.

Assert.*assertEquals*(statusCode, 200, "inCorrect status code returned");

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*validate Response headers\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Access header with a given name. Header = Content-Type

String contentType = response.header("Content-Type");

System.***out***.println("Content-Type value: " + contentType);

Assert.*assertEquals*(contentType, "application/json; charset=utf-8");

// Access header with a given name. Header = Server

String serverType = response.header("Server");

System.***out***.println("Server value: " + serverType);

// Access header with a given name. Header = Content-Encoding

String acceptLanguage = response.header("Content-Encoding");

System.***out***.println("Content-Encoding: " + acceptLanguage);

Assert.*assertEquals*(serverType, "nginx/1.17.10 (Ubuntu)");

// Get all the headers and then print each header

Headers allHeaders = response.headers();

// Iterate over all the Headers

**for**(Header header : allHeaders) {

System.***out***.println("Key: " + header.getName() + " Value: " + header.getValue());

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*validate the response body\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// store the body of the Response

Response response = httpRequest.get("/Hyderabad");

ResponseBody body = response.getBody();

// By using the ResponseBody.asString() method, we can convert the body

// into the string representation.

System.***out***.println("Response Body is: " + body.asString());

// To check for sub string presence get the Response body as a String.

// Do a String.contains

String bodyAsString = body.asString();

Assert.*assertEquals*(bodyAsString.contains("Hyderabad"), **true**, "Response body doesn’t contain Hyderabad");

// convert the body into lower case and then do a comparison to ignore casing.

Assert.*assertEquals*(bodyAsString.toLowerCase().contains("hyderabad"),

**true**, "Response body doesn’t contain Hyderabad");

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\* Extract a Node text from Response using JsonPath \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Test

**public** **void** VerifyCityInJsonResponse()

{

RestAssured.baseURI = "https://restapi.demoqa.com/utilities/weather/city";

RequestSpecification httpRequest = RestAssured.given();

Response response = httpRequest.get("/Hyderabad");

// First get the JsonPath object instance from the Response interface

JsonPath jsnPath = response.jsonPath();

// Then simply query the JsonPath object to get a String value of the node

// specified by JsonPath: City (Note: You should not put $. in the Java code)

String city = jsnPath.get("City");

// Let us print the city variable to see what we got

System.***out***.println("City received from Response " + city);

// Validate the response

Assert.*assertEquals*(city, "Hyderabad", " Hyderabad is not availble in the Response");

// Print the temperature node

System.***out***.println("Temperature received from Response" + jsnPath.get("Temperature"));

// Print the humidity node

System.***out***.println("Humidity received from Response " + jsnPath.get("Humidity"));

// Print weather description

System.***out***.println("Weather description received from Response " + jsnPath.get("Weather"));

// Print Wind Speed

System.***out***.println("City received from Response " + jsnPath.get("WindSpeed"));

// Print Wind Direction Degree

System.***out***.println("City received from Response " + jsnPath.get("WindDirectionDegree"));

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*send a request using Query Parameters\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Test

**public** **void** queryParameter() {

RestAssured.baseURI= "https://bookstore.toolsqa.com/BookStore/v1";

RequestSpecification httpRequest = RestAssured.given();

//Passing the resource details

Response res = httpRequest.queryParam("ISBN","9781449325862").get("/Book");

//Retrieving the response body using getBody() method

ResponseBody body = res.body();

//Converting the response body to string object

String stringBody = body.asString();

//Creating object of JsonPath and passing the string response body as parameter

//JsonPath jonPath = response.jsonPath();

JsonPath jpath = **new** JsonPath(stringBody);

//Store publisher name in a string variable

String title = jpath.getString("title");

System.***out***.println("The book title is - "+title);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* POST request method \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **void** UserRegistrationSuccessful()

{

RestAssured.baseURI = "https://demoqa.com/BookStore/v1/Books";

RequestSpecification request = RestAssured.given();

// Add/specify a header to the Request body as JSON

request.header("Content-Type", "application/json");

// Create a JSON request which contains all the fields

// JSONObject is a class that represents a Simple JSON.

// We can add Key - Value pairs using the put method

JSONObject json\_object = **new** JSONObject();

json\_object.put("userId", "TQ123");

json\_object.put("isbn", "9781449325862");

// send body in map format

Map<String, String> map = **new** HashMap<String, String>();

map.put("userId", "TQ123");

map.put("isbn", "9781449325862");

//Add JSON body in the request and send the Request

// Add the Json objects to the body of the request

request.body(json\_object.toJSONString());

request.body(map);

// Post the request and check the response

// Validate the Response

Response response = request.post("/BookStoreV1BooksPost");

System.***out***.println("The status received: " + response.statusLine());

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PUT request Method\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

String userId= "toolsqa\_test";

String baseUrl="https://demoqa.com";

String token = "eyJhbGckpXVCJ9.eyJ1c2VyTmFtNjQyMjF9.lW8JJvJF7jKoIIMJQ4";

String isbn ="9781449325865";

@Test

**public** **void** updateBook() {

RestAssured.baseURI = baseUrl;

RequestSpecification httpRequest = RestAssured.given()

.header("Authorization", "Bearer " + token)

.header("Content-Type", "application/json");

Response res = httpRequest

.body("{ \"isbn\": \"" + isbn + "\", \"userId\": \"" + userId + "\"}")

.put("/BookStore/v1/Book/9781449325862");

//Fetching the response code from the request and validating the same

System.***out***.println("The response code - " +res.getStatusCode());

Assert.*assertEquals*(res.getStatusCode(),200);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*delete request Method\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Test

**public** **void** deleteBook() {

RestAssured.baseURI = baseUrl;

RequestSpecification httpRequest = RestAssured.given()

.header("Authorization", "Bearer " + token)

.header("Content-Type", "application/json");

//Calling the Delete API with request body

Response res = httpRequest.body("{ \"isbn\": \"" + isbn + "\", \"userId\": \"" + userId + "\"}")

.delete("/BookStore/v1/Book");

//Fetching the response code from the request and validating the same

System.***out***.println("The response code is - " +res.getStatusCode());

Assert.*assertEquals*(res.getStatusCode(),204);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Preemptive Authentication (basic Auth2)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Test

**public** **void** getUserData() {

//Using the preemptive directive of basic auth to send credentials to the server

RequestSpecification httpRequest = RestAssured.given().auth().preemptive().basic("postman", "password");

Response res = httpRequest.get("https://postman-echo.com/basic-auth");

ResponseBody body = res.body();

//Converting the response body to string

String rbdy = body.asString();

System.***out***.println("Data from the GET API- "+rbdy);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*validate Response statusLine\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RestAssured.baseURI = "https://demoqa.com/BookStore/v1/Books";

RequestSpecification httpRequest = RestAssured.given();

Response response = httpRequest.get("");

String statusLine = response.getStatusLine();

**int** statusCode = response.getStatusCode();

String string\_response = response.asString();

String pretty\_response = response.prettyPrint();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*validate Response headers\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

String Header\_contentType = response.header("Content-Type");

String Header\_serverType = response.header("Server");

String Header\_acceptLanguage = response.header("Content-Encoding");

Headers allHeaders = response.headers();

**for**(Header header : allHeaders) {

System.out.println("Key: " + header.getName() + " Value: " + header.getValue());

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*validate the response body\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Response response = httpRequest.get("/Hyderabad");

ResponseBody body = response.getBody();

String bodyAsString = body.asString();

String bodyStringLowerCase = body.asString.toLowerCase();

Assert.assertEquals(bodyAsString.toLowerCase().contains("hyderabad"), **true**, "Response body not contains Hyderabad");

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Extract a Node text from Response using JsonPath\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Response response = httpRequest.get("/Hyderabad");

JsonPath jsnPath = response.jsonPath();

String city = jsnPath.get("City");

String temperature = jsnPath.getString("Temperature");

String humidity = jsnPath.get("Humidity"));

String weather = jsnPath.get("Weather"));

String windSpeed = jsnPath.get("WindSpeed"));

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*send GET request using Query Parameters\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RestAssured.baseURI= "https://bookstore.toolsqa.com/BookStore/v1";

RequestSpecification httpRequest = RestAssured.given();

Response response = httpRequest.queryParam("ISBN","9781449325862").get("/Book");

ResponseBody body = response.body();

String bodyString = body.asString();

JsonPath jpath = **new** JsonPath(bodyString);

String title = jpath.getString("title");

System.out.println("The book title is - "+title);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*POST request method\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RestAssured.baseURI = "https://demoqa.com/BookStore/v1/Books";

RequestSpecification request = RestAssured.given();

request.header("Content-Type", "application/json");

JSONObject json\_object = **new** JSONObject();

json\_object.put("userId", "TQ123");

json\_object.put("isbn", "9781449325862");

request.body(json\_object.toJSONString());

Response response = request.post("/BookStoreV1BooksPost");

System.out.println("The status received: " + response.statusLine());

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PUT request Method \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

String userId= "toolsqa\_test";

String baseUrl="https://demoqa.com";

String token = "eyJhbGckpXVCJ9.eyJ1c2VyTmFtNjQyMjF9.lW8JJvJF7jKoIIMJQ4";

String isbn ="9781449325865";

@Test

**public** **void** updateBook() {

RestAssured.baseURI = baseUrl;

RequestSpecification httpRequest = RestAssured.given()

.header("Authorization", "Bearer " + token)

.header("Content-Type", "application/json");

Response res = httpRequest.body("{ \"isbn\": \"" + isbn + "\", \"userId\": \"" + userId + "\"}")

.put("/BookStore/v1/Book/9781449325862");

Assert.assertEquals(res.getStatusCode(),200);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*delete request Method\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RestAssured.baseURI = baseUrl;

RequestSpecification httpRequest = RestAssured.given()

.header("Authorization", "Bearer " + token)

.header("Content-Type", "application/json");

Response res = httpRequest.body("{ \"isbn\": \"" + isbn + "\", \"userId\": \"" + userId + "\"}").delete("/BookStore/v1/Book");

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Preemptive Authentication (basic Auth2)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RequestSpecification httpRequest = RestAssured.given().auth().preemptive().basic("postman", "password");

Response res = httpRequest.get("https://postman-echo.com/basic-auth");

ResponseBody body = res.body();

String rbdy = body.asString();

System.out.println("Data from the GET API- "+rbdy);

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Test with Query and Path parameters \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

@Test

**public** **void** test\_Query\_path\_parameters() {

//RestAssured.baseURI = "https://reqres.in/api/users?page=2";

RestAssured.*given*()

.pathParam("**myPath**", "users")

.queryParam("page", 2)

.queryParam("id", 4)

.when()

.get("https://reqres.in/api/{**myPath**}")

.then()

.statusCode(200)

.log().all();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* API Test Framework \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Folders : reports folder.**

**package** com.employeeapi.base;

**public** **class** TestBase {

**public** **static** RequestSpecification httpRequest;

**public** **static** Response response;

//Hard coded - Input for Get details of Single Employee & update employee.

**public** String empID="58512";

**public** Logger logger;

@BeforeClass

**public** **void** setup(){

logger=Logger.getLogger("EmployeesRestAPI"); //added Logger

PropertyConfigurator.configure("Log4j.properties"); //added logger

logger.setLevel(Level.DEBUG);

}

}

**package** com.employeeapi.utilities;

**public** **class** Listeners **extends** TestListenerAdapter

{

**public** ExtentHtmlReporter htmlReporter;

**public** ExtentReports extent;

**public** ExtentTest test;

**public** **void** onStart(ITestContext testContext)

{

//specify location of the report

htmlReporter=**new** ExtentHtmlReporter(System.getProperty("user.dir")

+ "/Reports/myReport.html");

// Title of report

htmlReporter.config().setDocumentTitle("Automation Report");

// name of the report

htmlReporter.config().setReportName("Rest API Testing Report");

//htmlReporter.config().setTestViewChartLocation(ChartLocation.TOP); //location of the chart

htmlReporter.config().setTheme(Theme.DARK);

extent=**new** ExtentReports();

extent.attachReporter(htmlReporter);

extent.setSystemInfo("Project Name","Employee Database API");

extent.setSystemInfo("Host name","localhost");

extent.setSystemInfo("Environemnt","QA");

extent.setSystemInfo("user","pavan");

}

**public** **void** onTestSuccess(ITestResult result)

{

//test=extent.createTest(result.getClass().getName());

//test.createNode(result.getName());

// create new entry in th report

test=extent.createTest(result.getName());

test.log(Status.PASS, "Test Case PASSED IS " + result.getName());

}

**public** **void** onTestFailure(ITestResult result)

{

// create new entry in th report

test=extent.createTest(result.getName());

// to add name in extent report

test.log(Status.FAIL, "TEST CASE FAILED IS " + result.getName());

// to add error/exception in extent report

test.log(Status.FAIL, "TEST CASE FAILED IS " + result.getThrowable());

}

**public** **void** onTestSkipped(ITestResult result)

{

// create new entry in th report

test=extent.createTest(result.getName());

test.log(Status.SKIP, "Test Case SKIPPED IS " + result.getName());

}

**public** **void** onFinish(ITestContext testContext)

{

extent.flush();

}

}

**package** com.employeeapi.utilities;

**import** org.apache.commons.lang3.RandomStringUtils;

**public** **class** RestUtils {

**public** **static** String empName() {

**String** generatedString = RandomStringUtils.randomAlphabetic(1);

**return** ("John"+generatedString);

}

**public** **static** String empSal() {

**String** generatedString = RandomStringUtils.randomNumeric(5);

**return** (generatedString);

}

**public** **static** String empAge() {

**String** generatedString = RandomStringUtils.randomNumeric(2);

**return** (generatedString);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Test Name: Get all employees data

URI: http: //dummy.restapiexample.com/api/v1/employees

Request Type: GET

Request Payload(Body): NA

\*\*\*\*\*\*\*\*\* Validations \*\*\*\*\*\*\*\*\*\*

Status Code : 200

Status Line : HTTP/1.1 200 OK

Content Type : text/html; charset=UTF-8

Server Type : nginx/1.14.1

Content Encoding : gzip

Content Length <800

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**package** com.employeeapi.testCases;

**public** **class** TC001\_Get\_All\_Employees **extends** TestBase{

@BeforeClass

**void** getAllEmployees() **throws** InterruptedException

{

logger.info("\*\*\*\*\*\*\*\*\* Started TC001\_Get\_All\_Employees \*\*\*\*\*\*\*\*\*\*");

RestAssured.baseURI = "http://dummy.restapiexample.com/api/v1";

httpRequest = RestAssured.given();

response = httpRequest.request(Method.GET,"/employees");

Thread.sleep(5000);

}

@Test

**void** checkResponseBody()

{

logger.info("\*\*\*\*\*\*\*\*\*\*\* Checking Response Body \*\*\*\*\*\*\*\*\*\*");

String responseBody = response.getBody().asString();

logger.info("Response Body==>"+responseBody);

Assert.assertTrue(responseBody!=**null**);

}

@Test

**void** checkStatusCode()

{

logger.info("\*\*\*\*\*\*\*\*\*\*\* Checking Status Code \*\*\*\*\*\*\*\*\*\*");

**int** statusCode = response.getStatusCode(); // Gettng status code

logger.info("Status Code is ==>" + statusCode); //200

Assert.assertEquals(statusCode, 200);

}

@Test

**void** checkResponseTime()

{

logger.info("\*\*\*\*\*\*\*\*\*\*\* Checking Response Time \*\*\*\*\*\*\*\*\*\*");

**long** responseTime = response.getTime(); // Getting status Line

logger.info("Response Time is ==>" + responseTime);

**if**(responseTime>2000)

logger.warn("Response Time is greater than 2000");

Assert.assertTrue(responseTime<10000);

}

@Test

**void** checkstatusLine()

{

logger.info("\*\*\*\*\*\*\*\*\*\*\* Checking Status Line \*\*\*\*\*\*\*\*\*\*");

String statusLine = response.getStatusLine(); // Getting status Line

logger.info("Status Line is ==>" + statusLine);

Assert.assertEquals(statusLine, "HTTP/1.1 200 OK");

}

@Test

**void** checkContentType()

{

logger.info("\*\*\*\*\*\*\*\*\*\*\* Checking Content Type \*\*\*\*\*\*\*\*\*\*");

String contentType = response.header("Content-Type");

logger.info("Content type is ==>" + contentType);

Assert.assertEquals(contentType, "text/html; charset=UTF-8");

}

@Test

**void** checkserverType()

{

logger.info("\*\*\*\*\*\*\*\*\*\*\* Checking Server Type \*\*\*\*\*\*\*\*\*\*");

String serverType = response.header("Server");

logger.info("Server Type is =>" +serverType);

Assert.assertEquals(serverType, "nginx/1.14.1");

}

@Test

**void** checkcontentEncoding()

{

logger.info("\*\*\*\*\*\*\*\*\*\*\* Checking Content Encoding\*\*\*\*\*\*\*\*\*\*");

String contentEncoding = response.header("Content-Encoding");

logger.info("Content Encoding is==>" +contentEncoding);

Assert.assertEquals(contentEncoding, "gzip");

}

@Test

**void** checkContentLenght()

{

logger.info("\*\*\*\*\*\*\*\*\*\*\* Checking Content Lenght\*\*\*\*\*\*\*\*\*\*");

String contentLength = response.header("Content-Length");

logger.info("Content Length is==>" +contentLength);

**if**(Integer.parseInt(contentLength)<100)

logger.warn("Content Length is less than 100");

Assert.assertTrue(Integer.parseInt(contentLength)>100);

}

@Test

**void** checkCookies()

{

logger.info("\*\*\*\*\*\*\*\*\*\*\* Checking Cookies \*\*\*\*\*\*\*\*\*\*");

String cookie = response.getCookie("PHPSESSID");

//Assert.assertEquals(cookie,"1esuvsfslcmiee2bfrsgnijtg0");

}

@AfterClass

**void** tearDown()

{

logger.info("\*\*\*\*\*\*\*\*\* Finished TC001\_Get\_All\_Employees \*\*\*\*\*\*\*\*\*\*");

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Test Name:Get a single employee data

URI: http://dummy.restapiexample.com/api/v1/employee/{id}

Request Type: GET

Request Payload(Body): NA

\*\*\*\*\*\*\*\*\* Validations \*\*\*\*\*\*\*\*\*\*

Status Code : 200

Status Line : HTTP/1.1 200 OK

Content Type : text/html; charset=UTF-8

Server Type : nginx/1.14.1

Content Encoding : gzip

Content Length <800

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**package** com.employeeapi.testCases;

**public** **class** TC002\_Get\_Single\_Employee\_Record **extends** TestBase{

RequestSpecification httpRequest;

Response response;

@BeforeClass

**void** getEmployeeData() **throws** InterruptedException

{

logger.info("\*\*\*\*\*\*\*\*\*Started TC002\_Get\_Single\_Employee\_Record \*\*\*\*\*\*\*\*\*\*");

RestAssured.baseURI = "http://dummy.restapiexample.com/api/v1";

httpRequest = RestAssured.given();

response = httpRequest.request(Method.GET, "/employee/"+empID);

Thread.sleep(7000);

}

@Test

**void** checkResponseBody()

{

String responseBody = response.getBody().asString();

Assert.assertEquals(responseBody.contains(empID), **true**);

}

@Test

**void** checkStatusCode()

{

**int** statusCode = response.getStatusCode(); // Gettng status code

Assert.assertEquals(statusCode, 200);

}

@Test

**void** checkResponseTime()

{

**long** responseTime = response.getTime(); // Getting status Line

Assert.assertTrue(responseTime<6000);

}

@Test

**void** checkstatusLine()

{

String statusLine = response.getStatusLine(); // Gettng status Line

Assert.assertEquals(statusLine, "HTTP/1.1 200 OK");

}

@Test

**void** checkContentType()

{

String contentType = response.header("Content-Type");

Assert.assertEquals(contentType, "text/html; charset=UTF-8");

}

@Test

**void** checkserverType()

{

String serverType = response.header("Server");

Assert.assertEquals(serverType, "nginx/1.14.1");

}

@Test

**void** checkContentLenght()

{

String contentLength = response.header("Content-Length");

Assert.assertTrue(Integer.parseInt(contentLength)<1500);

}

@AfterClass

**void** tearDown()

{

logger.info("\*\*\*\*\* Finished TC002\_Get\_Single\_Employee\_Record \*\*\*\*\*\*\*\*\*");

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Test Name:Create new record in database

URI: http://dummy.restapiexample.com/api/v1/create

Request Type: POST

Request Payload(Body): {"name":"XXXXX","salary":"XXXX","age":"XX"}

\*\*\*\*\*\*\*\*\* Validations \*\*\*\*\*\*\*\*\*\*

Response Payload(Body) : {"name":"XXXXX","salary":"XXXX","age":"XX"}

Status Code : 200

Status Line : HTTP/1.1 200 OK

Content Type : text/html; charset=UTF-8

Server Type : nginx/1.14.1

Content Encoding : gzip

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**package** com.employeeapi.testCases;

**public** **class** TC003\_Post\_Employee\_Record **extends** TestBase{

RequestSpecification httpRequest;

Response response;

String empName=RestUtils.empName();

String empSalary=RestUtils.empSal();

String empAge=RestUtils.empAge();

@BeforeClass

**void** createEmployee() **throws** InterruptedException

{

logger.info("\*\*\*\*\*\*\*\*\*Started TC003\_Post\_Employee\_Record \*\*\*\*\*\*\*\*\*\*");

RestAssured.baseURI = "http://dummy.restapiexample.com/api/v1";

httpRequest = RestAssured.given();

// JSONObject is a class that represents a simple JSON. We can add Key-Value pairs using the put method

//{"name":"John123X","salary":"123","age":"23"}

JSONObject requestParams = **new** JSONObject();

requestParams.put("name", empName); // Cast

requestParams.put("salary", empSalary);

requestParams.put("age", empAge);

// Add a header stating the Request body is a JSON

httpRequest.header("Content-Type", "application/json");

// Add the Json to the body of the request

httpRequest.body(requestParams.toJSONString());

response = httpRequest.request(Method.POST, "/create");

Thread.sleep(5000);

}

@Test

**void** checkResposeBody()

{

String responseBody = response.getBody().asString();

Assert.assertEquals(responseBody.contains(empName), **true**);

Assert.assertEquals(responseBody.contains(empSalary), **true**);

Assert.assertEquals(responseBody.contains(empAge), **true**);

}

@Test

**void** checkStatusCode()

{

**int** statusCode = response.getStatusCode(); // Gettng status code

Assert.assertEquals(statusCode, 200);

}

@Test

**void** checkstatusLine()

{

String statusLine = response.getStatusLine(); // Gettng status Line

Assert.assertEquals(statusLine, "HTTP/1.1 200 OK");

}

@Test

**void** checkContentType()

{

String contentType = response.header("Content-Type");

Assert.assertEquals(contentType, "text/html; charset=UTF-8");

}

@Test

**void** checkserverType()

{

String serverType = response.header("Server");

Assert.assertEquals(serverType, "nginx/1.14.1");

}

@Test

**void** checkcontentEncoding()

{

String contentEncoding = response.header("Content-Encoding");

Assert.assertEquals(contentEncoding, "gzip");

}

@AfterClass

**void** tearDown()

{

logger.info("\*\*\*\*\*\*\*\*\*Finished TC003\_Post\_Employee\_Record \*\*\*\*\*\*\*\*\*");

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Test Name:Update an employee record

URI: http://dummy.restapiexample.com/api/v1/update/{id}

Request Type: PUT

Request Payload(Body): {"name":"XXXXX","salary":"XXXX","age":"XX"}

\*\*\*\*\*\*\*\*\* Validations \*\*\*\*\*\*\*\*\*\*

Response Payload(Body) : {"name":"XXXXX","salary":"XXXX","age":"XX"}

Status Code : 200

Status Line : HTTP/1.1 200 OK

Content Type : text/html; charset=UTF-8

Server Type : nginx/1.14.1

Content Encoding : gzip

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**package** com.employeeapi.testCases;

**public** **class** TC004\_Put\_Employee\_Record **extends** TestBase {

RequestSpecification httpRequest;

Response response;

String empName=RestUtils.empName();

String empSalary=RestUtils.empSal();

String empAge=RestUtils.empAge();

@BeforeClass

**void** updateEmployee() **throws** InterruptedException

{

logger.info("\*\*\*\*\*\*\*\*\*Started TC004\_Put\_Employee\_Record \*\*\*\*\*\*\*\*\*\*");

RestAssured.baseURI = "http://dummy.restapiexample.com/api/v1";

httpRequest = RestAssured.given();

// JSONObject is a class that represents a simple JSON. We can add Key-Value pairs using the put method

//{"name":"John123X","salary":"123","age":"23"}

JSONObject requestParams = **new** JSONObject();

requestParams.put("name", empName); // Cast

requestParams.put("salary", empSalary);

requestParams.put("age", empAge);

// Add a header stating the Request body is a JSON

httpRequest.header("Content-Type", "application/json");

// Add the Json to the body of the request

httpRequest.body(requestParams.toJSONString());

response = httpRequest.request(Method.PUT, "/update/"+empID);

Thread.sleep(5000);

}

@Test

**void** checkResposeBody()

{

String responseBody = response.getBody().asString();

Assert.assertEquals(responseBody.contains(empName), **true**);

Assert.assertEquals(responseBody.contains(empSalary), **true**);

Assert.assertEquals(responseBody.contains(empAge), **true**);

}

@Test

**void** checkStatusCode()

{

**int** statusCode = response.getStatusCode(); // Gettng status code

Assert.assertEquals(statusCode, 200);

}

@Test

**void** checkstatusLine()

{

String statusLine = response.getStatusLine(); // Gettng status Line

Assert.assertEquals(statusLine, "HTTP/1.1 200 OK");

}

@Test

**void** checkContentType()

{

String contentType = response.header("Content-Type");

Assert.assertEquals(contentType, "text/html; charset=UTF-8");

}

@Test

**void** checkserverType()

{

String serverType = response.header("Server");

Assert.assertEquals(serverType, "nginx/1.14.1");

}

@Test

**void** checkcontentEncoding()

{

String contentEncoding = response.header("Content-Encoding");

Assert.assertEquals(contentEncoding, "gzip");

}

@AfterClass

**void** tearDown()

{

logger.info("\*\*\*\*\*\*\*\*\* Finished TC004\_Put\_Employee\_Record \*\*\*\*\*\*\*\*\*\*");

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Test Name: Delete an employee record

URI: http://dummy.restapiexample.com/api/v1/delete/{id}

Request Type: DELETE

Request Payload(Body): NA

\*\*\*\*\*\*\*\*\* Validations \*\*\*\*\*\*\*\*\*\*

Response Payload(Body) : {"success":{"text":"successfully! deleted Records"}}

Status Code : 200

Status Line : HTTP/1.1 200 OK

Content Type : text/html; charset=UTF-8

Server Type : nginx/1.14.1

Content Encoding : gzip

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**package** com.employeeapi.testCases;

**public** **class** TC005\_Delete\_Employee\_Record **extends** TestBase{

RequestSpecification httpRequest;

Response response;

@BeforeClass

**void** deleteEmployee() **throws** InterruptedException

{

logger.info("\*\*\*\*\*\*\*\*\*Started TC005\_Delete\_Employee\_Record \*\*\*\*\*\*\*\*\*\*");

RestAssured.baseURI = "http://dummy.restapiexample.com/api/v1";

httpRequest = RestAssured.given();

response = httpRequest.request(Method.GET, "/employees");

// First get the JsonPath object instance from the Response interface

JsonPath jsonPathEvaluator = response.jsonPath();

//Capture id

String empID=jsonPathEvaluator.get("[0].id");

response = httpRequest.request(Method.DELETE, "/delete/"+empID); //Pass ID to delete record

Thread.sleep(3000);

}

@Test

**void** checkResposeBody()

{

String responseBody = response.getBody().asString();

Assert.assertEquals(responseBody.contains("successfully! deleted Records"), **true**);

}

@Test

**void** checkStatusCode()

{

**int** statusCode = response.getStatusCode(); // Gettng status code

Assert.assertEquals(statusCode, 200);

}

@Test

**void** checkstatusLine()

{

String statusLine = response.getStatusLine(); // Gettng status Line

Assert.assertEquals(statusLine, "HTTP/1.1 200 OK");

}

@Test

**void** checkContentType()

{

String contentType = response.header("Content-Type");

Assert.assertEquals(contentType, "text/html; charset=UTF-8");

}

@Test

**void** checkserverType()

{

String serverType = response.header("Server");

Assert.assertEquals(serverType, "nginx/1.14.1");

}

@Test

**void** checkcontentEncoding()

{

String contentEncoding = response.header("Content-Encoding");

Assert.assertEquals(contentEncoding, "gzip");

}

@AfterClass

**void** tearDown()

{

logger.info("\*\*\*\*\*\*\*\*\* Finished TC005\_Delete\_Employee\_Record \*\*\*\*\*\*\*\*");

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Data driven API test \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**package** rest\_Methods;

**public** **class** XLutils {

**public** **static** FileInputStream *fis*;

**public** **static** XSSFWorkbook *wb*;

**public** **static** XSSFSheet *sheet*;

**public** **static** XSSFRow *row*;

**public** **static** XSSFCell *cell*;

**public** **static** **int** getrowCount(String filePath, **int** sheetNo) **throws** Exception {

File file = **new** File(filePath);

*fis* = **new** FileInputStream(file);

*wb* = **new** XSSFWorkbook(*fis*);

*sheet* = *wb*.getSheetAt(sheetNo);

**int** rowCountNo = *sheet*.getLastRowNum();

*wb*.close();

*fis*.close();

**return** rowCountNo;

}

**public** **static** **int** getCellCount(String file, **int** sheetNo, **int** rownum) **throws** Exception {

File filee = **new** File(file);

*fis* = **new** FileInputStream(filee);

*wb* = **new** XSSFWorkbook(*fis*);

*sheet* = *wb*.getSheetAt(sheetNo);

*row* = *sheet*.getRow(rownum);

**int** cellCountNo = *row*.getLastCellNum();

*wb*.close();

*fis*.close();

**return** cellCountNo;

}

**public** **static** String getcellData(String file, **int** sheetNo, **int** rownum, **int** cellnum) **throws** Exception {

File filee = **new** File(file);

*fis* = **new** FileInputStream(filee);

*wb* = **new** XSSFWorkbook(*fis*);

*sheet* = *wb*.getSheetAt(sheetNo);

*row* = *sheet*.getRow(rownum);

*cell* = *row*.getCell(cellnum);

String data;

**try** {

DataFormatter dataFormatter = **new** DataFormatter();

String cellData = dataFormatter.formatCellValue(*cell*);

**return** cellData;

} **catch** (Exception e) {

data="";

}

*wb*.close();

*fis*.close();

**return** data;

}

}

**package** rest\_Methods;

**public** **class** DataDrivenTest {

@Test(dataProvider = "testData1")

**public** **void** dataDriventest\_addNewEmployee(String a, String b, String c, String d, String e) {

RestAssured.*baseURI* = "https://reqres.in/";

RequestSpecification request = RestAssured.*given*();

request.header("Content-type", "application/json");

JSONObject json\_object = **new** JSONObject();

json\_object.put("id", a);

json\_object.put("email", b);

json\_object.put("first\_name", c);

json\_object.put("last\_name", d);

json\_object.put("avatar", e);

request.body(json\_object.toJSONString());

Response response = request.post("api/users");

//Response response = request.get("api/users");

String prettyBody = response.asPrettyString();

System.***out***.println("prettyBody : "+prettyBody);

Assert.*assertTrue*(prettyBody.contains(a));

Assert.*assertTrue*(prettyBody.contains(b));

Assert.*assertTrue*(prettyBody.contains(c));

}

//@DataProvider(name = "testData1")

**public** String[][] dataProvider1() {

String data[][] = {

{"55", "fname1", "lname1", "email1","avatar1"},

{"56","fname2","lname2","email2","avatar2"},

{"57","fname3","lname3","email3","avatar3"}

};

**return** (data);

}

@DataProvider(name = "testData1") //apache poi: 4.1.0

**public** String[][] xldataProvider() **throws** Exception {

String filePath = "./XLfolder/API\_test\_data.xlsx";

**int** row = XLutils.*getrowCount*(filePath, 0);

**int** cel = XLutils.*getCellCount*(filePath, 0, 1);

String data[][] = **new** String[row][cel];

**for**(**int** i=1; i<=row; i++) {

**for**(**int** j=0; j<cel; j++) {

data[i-1][j] = XLutils.*getcellData*(filePath, 0, i, j);

}

}

**return** data;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\* Serialization and deSerialization \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**package** serialization;

**class** Test **implements** Serializable{

**int** i = 10;

**int** j = 20;

}

**public** **class** Serialization\_deSerilization {

**public** **static** **void** main(String[] args) **throws** Exception {

Test test1 = **new** Test();

//serialization

FileOutputStream fos = **new** FileOutputStream("test.txt");

ObjectOutputStream oos = **new** ObjectOutputStream(fos);

oos.writeObject(test1);

//deserialization

FileInputStream fis = **new** FileInputStream("test.txt");

ObjectInputStream ois = **new** ObjectInputStream(fis);

Test test2 = (Test) ois.readObject();

System.***out***.println(test2.i+" "+test2.j);

}

}

-------------------------------------------------------------

\*\* first let see POST and GET method simple approach.

**package** serialization;

**public** **class** Student\_API\_NOT\_Serialization {

**public** HashMap map = **new** HashMap();

// POST Request without serialization

@Test(priority=1)

**public** **void** createNewStudent() {

map.put("id", "9");

map.put("first\_name", "jawad");

map.put("last\_name", "karim");

map.put("email", "adga@gmail.com");

ArrayList <String> courseList = **new** ArrayList<String>();

courseList.add("selenium");

courseList.add("Java");

map.put("courses", courseList);

RequestSpecification httpRequest = RestAssured.*given*()

.contentType(ContentType.***JSON***)

.body(map);

Response response = httpRequest.post("https://reqres.in/api/users");

**int** status\_code = response.getStatusCode();

String stringBody = response.body().asString();

Assert.*assertEquals*(stringBody.contains("createdAt"), **true**);

}

// GET Request without serialization

@Test

**public** **void** getStudentsRecord() {

RestAssured.*given*().

when().

get("https://reqres.in/api/users").

then().

statusCode(200).

log().all();

}

}

\*\* now let see POST and GET method with serialization approach.

\*\* create a root POJO class

**public** **class** Student {

**int** id;

String email;

String first\_name;

String last\_name;

List <String> courseList;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getFirst\_name() {

**return** first\_name;

}

**public** **void** setFirst\_name(String first\_name) {

**this**.first\_name = first\_name;

}

**public** String getLast\_name() {

**return** last\_name;

}

**public** **void** setLast\_name(String last\_name) {

**this**.last\_name = last\_name;

}

**public** List<String> getCourseList() {

**return** courseList;

}

**public** **void** setCourseList(List<String> courseList) {

**this**.courseList = courseList;

}

}

**public** **class** Serialization\_test {

// POST with serialization process

@Test(priority=1)

**public** **void** postPerson\_serialization() {

Student student = **new** Student();

student.setId(7);

student.setFirst\_name("jawad");

student.setLast\_name("karim");

student.setEmail("adf@gmail.com");

ArrayList<String> courseList = **new** ArrayList<String>();

courseList.add("java");

courseList.add("selenium");

courseList.add("API");

student.setCourseList(courseList);

*given*()

.contentType(ContentType.***JSON***)

.body(student)

.when()

.post("https://reqres.in/api/users")

.then()

.assertThat().statusCode(201);

}

// GET with de\_serialization process

// for GET with de\_serialization first I need to create POJO classes for .. variables like above (Student class).

@Test(priority=2)

**public** **void** getStudent\_deserialization() {

Student student2= *get*("https://reqres.in/api/users?page=2").as(Student.**class**);

System.***out***.println(student2.getFirst\_name());

}

}

­­­

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Serialization POJO /Get \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

{

"page": 2,

"per\_page": 6,

"total": 12,

"total\_pages": 2,

"data": [

{

"id": 7,

"email": "michael.lawson@reqres.in",

"first\_name": "Michael",

"last\_name": "Lawson",

"avatar": "https://reqres.in/img/faces/7-image.jpg"

},

{

"id": 12,

"email": "rachel.howell@reqres.in",

"first\_name": "Rachel",

"last\_name": "Howell",

"avatar": "https://reqres.in/img/faces/12-image.jpg"

}

],

"support": {

"url": "https://reqres.in/#support-heading",

"text": "To keep ReqRes free, contributions are appreciated!"

}

**package** deseralization\_pojo;

**public** **class** Get\_list\_users {

**private** String page;

**private** String per\_page;

**private** String total;

**private** String total\_pages;

**private** List <DataPOJO> data;

**private** SupportPOJO support;

**public** String getPage() {

**return** page;

}

**public** **void** setPage(String page) {

**this**.page = page;

}

**public** String getPer\_page() {

**return** per\_page;

}

**public** **void** setPer\_page(String per\_page) {

**this**.per\_page = per\_page;

}

**public** String getTotal() {

**return** total;

}

**public** **void** setTotal(String total) {

**this**.total = total;

}

**public** String getTotal\_pages() {

**return** total\_pages;

}

**public** **void** setTotal\_pages(String total\_pages) {

**this**.total\_pages = total\_pages;

}

**public** List<DataPOJO> getData() {

**return** data;

}

**public** **void** setData(List<DataPOJO> data) {

**this**.data = data;

}

**public** SupportPOJO getSupport() {

**return** support;

}

**public** **void** setSupport(SupportPOJO support) {

**this**.support = support;

}

@Override

**public** String toString() {

**return** "Get\_list\_users [page=" + **this**.page + ",

per\_page=" + **this**.per\_page + ", total=" + **this**.total

+ ", total\_pages="+ **this**.total\_pages + ", data="

+ **this**.data + ", support=" + **this**.support + "]";

}

}

**package** deseralization\_pojo;

**public** **class** DataPOJO {

**private** **int** id;

**private** String email;

**private** String first\_name;

**private** String last\_name;

**private** String avatar;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getFirst\_name() {

**return** first\_name;

}

**public** **void** setFirst\_name(String first\_name) {

**this**.first\_name = first\_name;

}

**public** String getLast\_name() {

**return** last\_name;

}

**public** **void** setLast\_name(String last\_name) {

**this**.last\_name = last\_name;

}

**public** String getAvatar() {

**return** avatar;

}

**public** **void** setAvatar(String avatar) {

**this**.avatar = avatar;

}

@Override

**public** String toString() {

**return** "DataPOJO [id=" + **this**.id + ", email=" + **this**.email + ", first\_name=" + **this**.first\_name + ", last\_name=" + **this**.last\_name

+ ", avatar=" + **this**.avatar + "]";

}

}

**package** deseralization\_pojo;

**public** **class** SupportPOJO {

**private** String url;

**private** String text;

**public** String getUrl() {

**return** url;

}

**public** **void** setUrl(String url) {

**this**.url = url;

}

**public** String getText() {

**return** text;

}

**public** **void** setText(String text) {

**this**.text = text;

}

@Override

**public** String toString() {

**return** "SupportPOJO [url=" + **this**.url + ", text=" + **this**.text + "]";

}

}

**package** deseralization\_pojo;

**public** **class** DeSerializingTest {

@Test

**public** **void** TestDeserialPOJO() {

RestAssured.*baseURI* = "https://reqres.in/api/users?page=2";

Get\_list\_users listUsers = RestAssured.*given*().when().get().as(Get\_list\_users.**class**);

//System.out.println("list of Users : "+listUsers.toString());

//System.out.println("list of data : "+ listUsers.getData());

//System.out.println("support data :"+ listUsers.getSupport());

List <DataPOJO> datas = listUsers.getData();

**for**(**int** i=0; i<datas.size(); i++) {

System.***out***.println(i+". data list : "+datas.get(i));

}

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** **Complex json with pojo class** **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

post response is/body : {

"instructor": "jawad",

"url": "softwaretestingandautomation.com",

"services": "softwaretesting",

"expertize": "automation",

"courses": {

"webAutomation": [

{

"courseTitle": "selenium",

"price": 3000

},

{

"courseTitle": "Java",

"price": 4000

}

],

"apiAutomation": [

{

"courseTitle": "API automation",

"price": 2000

},

{

"courseTitle": "rest assured",

"price": 2000

}

],

"mobileAutomation": [

{

"courseTitle": "Mobile Automation",

"price": 4000

}

]

},

"linkedin": "linkedinid",

"id": "107",

"createdAt": "2022-12-29T18:15:27.000Z"

}

: COMPLEX JSON POST WITH POJO :

: create a POJO class for every block & create private variables + setter/getter

: assign data for 1st block

: assign data for 2nd block

: Store data sets in ArrayList if multiple block of data

: assign data list to Course objects (immidiate super block)

: assign all variable of Parent objects.

\*\* Parent POJO class

**package** complex\_json\_POST;

**public** **class** ParentPOJO {

**private** String instructor;

**private** String url;

**private** String services;

**private** String expertize;

**private** Courses courses;

**private** String linkedin;

**public** String getInstructor() {

**return** instructor;

}

**public** **void** setInstructor(String instructor) {

**this**.instructor = instructor;

}

**public** String getUrl() {

**return** url;

}

**public** **void** setUrl(String url) {

**this**.url = url;

}

**public** String getServices() {

**return** services;

}

**public** **void** setServices(String services) {

**this**.services = services;

}

**public** String getExpertize() {

**return** expertize;

}

**public** **void** setExpertize(String expertize) {

**this**.expertize = expertize;

}

**public** Courses getCourses() {

**return** courses;

}

**public** **void** setCourses(Courses courses) {

**this**.courses = courses;

}

**public** String getLinkedin() {

**return** linkedin;

}

**public** **void** setLinkedin(String linkedin) {

**this**.linkedin = linkedin;

}

}

\*\* Courses POJO class

**package** complex\_json\_POST;

**import** java.util.List;

**public** **class** Courses {

**private** List <WebAutomation> webAutomation;

**private** List <ApiAutomation> apiAutomation;

**private** List <MobileAutomation> mobileAutomation;

**public** List<WebAutomation> getWebAutomation() {

**return** webAutomation;

}

**public** **void** setWebAutomation(List<WebAutomation> webAutomation) {

**this**.webAutomation = webAutomation;

}

**public** List<ApiAutomation> getApiAutomation() {

**return** apiAutomation;

}

**public** **void** setApiAutomation(List<ApiAutomation> apiAutomation) {

**this**.apiAutomation = apiAutomation;

}

**public** List<MobileAutomation> getMobileAutomation() {

**return** mobileAutomation;

}

**public** **void** setMobileAutomation(List<MobileAutomation> mobileAutomation) {

**this**.mobileAutomation = mobileAutomation;

}

}

\*\* web automation POJO class

**package** complex\_json\_POST;

**public** **class** WebAutomation {

**private** String courseTitle;

**private** **int** price;

**public** String getCourseTitle() {

**return** courseTitle;

}

**public** **void** setCourseTitle(String courseTitle) {

**this**.courseTitle = courseTitle;

}

**public** **int** getPrice() {

**return** price;

}

**public** **void** setPrice(**int** price) {

**this**.price = price;

}

}

\*\* API automation POJO class

**package** complex\_json\_POST;

**public** **class** ApiAutomation {

**private** String courseTitle;

**private** **int** price;

**public** String getCourseTitle() {

**return** courseTitle;

}

**public** **void** setCourseTitle(String courseTitle) {

**this**.courseTitle = courseTitle;

}

**public** **int** getPrice() {

**return** price;

}

**public** **void** setPrice(**int** price) {

**this**.price = price;

}

}

\*\* mobile automation POJO class

**package** complex\_json\_POST;

**public** **class** MobileAutomation {

**private** String courseTitle;

**private** **int** price;

**public** String getCourseTitle() {

**return** courseTitle;

}

**public** **void** setCourseTitle(String courseTitle) {

**this**.courseTitle = courseTitle;

}

**public** **int** getPrice() {

**return** price;

}

**public** **void** setPrice(**int** price) {

**this**.price = price;

}

}

\*\* Execute test rest assured class

**package** complex\_json\_POST;

**public** **class** Test\_complex\_json {

@Test

**public** **void** executeTest() {

WebAutomation webauto1 = **new** WebAutomation();

webauto1.setCourseTitle("selenium");

webauto1.setPrice(3000);

WebAutomation webauto2 = **new** WebAutomation();

webauto2.setCourseTitle("Java");

webauto2.setPrice(4000);

List <WebAutomation> webList = **new** ArrayList<WebAutomation>();

webList.add(webauto1);

webList.add(webauto2);

ApiAutomation api1 = **new** ApiAutomation();

api1.setCourseTitle("API automation");

api1.setPrice(2000);

ApiAutomation api2 = **new** ApiAutomation();

api2.setCourseTitle("rest assured");

api2.setPrice(2000);

List <ApiAutomation> apiList = **new** ArrayList<ApiAutomation>();

apiList.add(api1);

apiList.add(api2);

MobileAutomation mobile1 = **new** MobileAutomation();

mobile1.setCourseTitle("Mobile Automation");

mobile1.setPrice(4000);

List <MobileAutomation> mobileList = **new** ArrayList<MobileAutomation>();

mobileList.add(mobile1);

Courses courses = **new** Courses();

courses.setWebAutomation(webList);

courses.setApiAutomation(apiList);

courses.setMobileAutomation(mobileList);

ParentPOJO parent = **new** ParentPOJO();

parent.setInstructor("jawad");

parent.setUrl("softwaretestingandautomation.com");

parent.setServices("softwaretesting");

parent.setExpertize("automation");

parent.setCourses(courses);

parent.setLinkedin("linkedinid");

Response response =

*given*()

.auth().none()

.header("Content-Type", "application/json")

.contentType(ContentType.***JSON***)

.when()

.body(parent)//.log().all()

.post("https://reqres.in/api/users");

System.***out***.println("post response is : "+ response.asPrettyString());

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Swagger API pet POST \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

response : {

"id": 102212,

"category": {

"id": 20121,

"name": "create a dog record"

},

"name": "new pet",

"photoUrls": [

"http://photoUrl",

"http://adghhdlj.com"

],

"tags": [

{

"id": 3031,

"name": "american dog"

},

{

"id": 3032,

"name": "Persian dog"

}

],

"status": "added new pet"

}

PASSED: API\_test

// pojo class.......

**package** swaggerAPI;

**public** **class** SwaggerPost {

**private** **int** id;

**private** CategoryPOJO Category;

**private** String name;

**private** List <String> photoUrls;

**private** List <TagsPOJO> tags;

**private** String status;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** CategoryPOJO getCategory() {

**return** Category;

}

**public** **void** setCategory(CategoryPOJO category) {

Category = category;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** List<String> getPhotoUrls() {

**return** photoUrls;

}

**public** **void** setPhotoUrls(List<String> photoUrls) {

**this**.photoUrls = photoUrls;

}

**public** List<TagsPOJO> getTags() {

**return** tags;

}

**public** **void** setTags(List<TagsPOJO> tags) {

**this**.tags = tags;

}

**public** String getStatus() {

**return** status;

}

**public** **void** setStatus(String status) {

**this**.status = status;

}

}

// pojo class.......

**package** swaggerAPI;

**public** **class** CategoryPOJO {

**private** **int** id;

**private** String name;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

// pojo class......

**package** swaggerAPI;

**public** **class** TagsPOJO {

**private** **int** id;

**private** String name;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

// test class.......

**package** swaggerAPI;

**public** **class** SwaggerAPI\_test {

@Test

**public** **void** API\_test() {

CategoryPOJO category = **new** CategoryPOJO();

category.setId(20121);

category.setName("create a dog record");

List <String> photoUrlList = **new** ArrayList <String>();

photoUrlList.add("http://photoUrl");

photoUrlList.add("http://adghhdlj.com");

TagsPOJO tag1 = **new** TagsPOJO();

tag1.setId(3031);

tag1.setName("american dog");

TagsPOJO tag2 = **new** TagsPOJO();

tag2.setId(3032);

tag2.setName("Persian dog");

List <TagsPOJO> tagList = **new** ArrayList<TagsPOJO>();

tagList.add(tag1);

tagList.add(tag2);

SwaggerPost sp = **new** SwaggerPost();

sp.setId(102212);

sp.setCategory(category);

sp.setName("new pet");

sp.setPhotoUrls(photoUrlList);

sp.setTags(tagList);

sp.setStatus("added new pet");

RequestSpecification request = RestAssured.*given*();

request.header("Content-Type", "application/json");

request.contentType(ContentType.***JSON***);

request.body(sp);

Response response = request.post("https://petstore.swagger.io/v2/pet");

System.***out***.println("response : "+response.asPrettyString());

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* validation and assertion \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

System.***out***.println("id : "+ response.body().path("id"));

System.***out***.println("category-id : "+ response.body().path("category.id"));

System.***out***.println("category-name : "+ response.body().path("category.name"));

System.***out***.println("name : "+ response.body().path("name"));

System.***out***.println("photoUrls : "+ response.body().path("photoUrls[0]"));

System.***out***.println("photoUrls : "+ response.body().path("photoUrls[1]"));

System.***out***.println("tags id1 : "+ response.jsonPath().get("tags[0].id"));

System.***out***.println("tags name1 : "+ response.jsonPath().get("tags[0].name"));

System.***out***.println("tags id1 : "+ response.jsonPath().getInt("tags[1].id"));

System.***out***.println("tags name1 : "+ response.jsonPath().getString("tags[1].name"));

System.***out***.println("status : "+ response.jsonPath().get("status"));

**int** respID = response.body().path("id");

**int** categoryID = response.body().path("category.id");

String categoryName = response.body().path("category.name");

String respName = response.body().path("name");

String catUrl = response.body().path("photoUrls[0]");

String dogUrl = response.body().path("photoUrls[1]");

Int total\_url = response.body().path("photoUrls.size()");

Assert.*assertEquals*(respID, 789);

Assert.*assertEquals*(categoryID, 456);

Assert.*assertEquals*(categoryName, "create a dog record");

Assert.*assertEquals*(respName, "new pet");

Assert.*assertEquals*(catUrl, "http://photoUrl");

Assert.*assertEquals*(dogUrl, "http://adghhdlj.com");

Assert.*assertEquals*(response.jsonPath().get("tags.size()"), 6);

Assert.*assertEquals*(response.jsonPath().get("tags[0].id"), 3031);

Assert.*assertEquals*(response.jsonPath().get("tags[0].name"), "american dog");

Assert.*assertEquals*(response.jsonPath().get("tags[1].id"), 3032);

Assert.*assertEquals*(response.jsonPath().get("tags[1].name"), "Persian dog");

Assert.*assertEquals*(response.jsonPath().get("status"), "added new pet");

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Schema validation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

: add the ‘json-schema-validator’ dependency to POM

: Once you get the response then copy the response

: Google – type Json to Json Schema – paste the response – execute – copy Schema

: Create a file ‘name.json’ under resources folder – paste the schema into the file.

**public** **class** Get\_with\_params {

@Test

**public** **void** test\_Query\_path\_parameters() {

//RestAssured.baseURI = "https://reqres.in/api/users?page=2";

RestAssured.*given*()

.pathParam("myPath", "users")

.queryParam("page", 2)

.queryParam("id", 4)

.when()

.get("https://reqres.in/api/{myPath}")

.then()

.statusCode(200)

// Json schema validation

.assertThat().body(JsonSchemaValidator.*matchesJsonSchemaInClasspath*("schema.json"))

.log().all();

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* REST API Http Response Codes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

200: OK. Everything worked as expected.

201: A resource was successfully created in response to a POST request. The Location header contains the URL pointing to the newly created resource.

204: The request was handled successfully and the response contains no body content (like a DELETE request).

304: The resource was not modified. You can use the cached version.

400: Bad request. This could be caused by various actions by the user, such as providing invalid JSON data in the request body etc.

401: Authentication failed.

403: The authenticated user is not allowed to access the specified API endpoint.

404: The requested resource does not exist.

405: Method not allowed. Please check the Allow header for the allowed HTTP methods.

415: Unsupported media type. The requested content type or version number is invalid.

422: Data validation failed (in response to a POST request, for example). Please check the response body for detailed error messages.

429: Too many requests. The request was rejected due to rate limiting.

500: Internal server error. This could be caused by internal program errors.