ASSIGNMENT 2

# Name: Faizan Ahmad

# Roll No: 221434448

# CSCS 351 Section: A

# Submitted to Professor Dr. Saad Bin Saleem

# Selenium Installation

The process of installing Selenium involves 3 steps, namely:

1. Install Java
2. Install Eclipse IDE
3. Install Selenium WebDriver

**Install Java**

Follow below steps to complete your Java installation.

Go to the Java Downloads Page and click on the option for Java Platform (JDK).

In the next page, select the Accept License Agreement radio button, accept it and click the download link against your matching system configuration.

You can run the installer once the download is over and follow onscreen instructions.

Go to start and search for ‘System’

Click on ‘System’

Click on ‘Advanced system settings’

Click on ‘Environment Variables’ under ‘Advanced’ tab.

Next thing that you have to do is to configure your environment variables. Let’s see how to do that. Here, you have to edit the path of the system variable.

Under ‘Variable value’, at the end of the line, enter the following path – %JAVA\_HOME%bin;

Now, you can click ‘OK’ and you are done.

Now to cross-check the installation, just run following command in cmd – java -version. It should display the installed version of Java in your system.

**Install Eclipse**

Follow the below steps to configure Eclipse on your system:

Navigate to the following URL – https://www.eclipse.org/downloads/ and select the download link depending on your system architecture – (32 Bit or 64 Bit) and download it.

Once the download is over, extract the zipped file and save it to any directory. The root folder is the eclipse.

Open the folder and launch eclipse.exe

Now, the last step is to install Selenium. Let’s see how to install selenium and configure it in your system.

**Install Selenium Web Driver**

Open the browser and navigate to http://www.seleniumhq.org.

Click ‘Download’ menu and choose ‘Download version’.

The download should get started for ‘selenium-server-standalone-x.y.z.jar’. Save this JAR in “C:Selenium”.

Next, you have to download Selenium Java Client. In the same downloads page, scroll down and you will find a section called Selenium Client and WebDriver Language. Now, click on Download link under Java.

Extract the zip file and save it in your selenium folder.

Next, you have to download Chrome Driver for Google Chrome and Gecko Driver if you are using Mozilla Firefox.

In the same downloads page, scroll further down and you will find third-party drivers section. There you will find options for downloading gecko driver and chrome driver. In the next page, you will find the latest version available. You can choose that and download based on your system configuration.

You can simply download the driver, extract the zip file and save it in any location. Similarly, you can download the latest version of gecko driver and save it if you are using the Mozilla browser.

Next, you can simply create a new project in Eclipse IDE and add all the external selenium jar files from selenium lib directory.

First, launch Eclipse and create new project.

Let’s create a class file and give public access modifier.

Next, you should add referenced libraries to your project. Click on your project -> Build Path -> Configure Build Path and add external JARs.

package Edureka;

Now you are all set to execute the first Selenium program. You can run the below script,

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.WebDriverWait;

public class FirstSeleniumScript {

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

System.setProperty("webdriver.chrome.driver", "C:Selenium-java-edurekachromedriver\_win32chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.manage().window().maximize();

driver.manage().deleteAllCookies();

driver.manage().timeouts().pageLoadTimeout(40, TimeUnit.SECONDS);

driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);

driver.get("https://login.yahoo.com/");

driver.findElement(By.xpath("//input[@id='login-username']")).sendKeys("edureka@yahoo.com");

}

}

# Conducting API testing and writing configuration used for API testing

1. Create a Maven Project in your IDE. We are using Eclipse.
2. Open your POM.xml

**Code structure**

Add the below dependency to your POM.xml:

<dependency>

<groupId>io.rest-assured</groupId>

<artifactId>json-path</artifactId>

<version>4.2.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.rest-assured</groupId>

<artifactId>xml-path</artifactId>

<version>4.2.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.rest-assured</groupId>

<artifactId>json-schema-validator</artifactId>

<version>4.2.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.rest-assured</groupId>

<artifactId>rest-assured-all</artifactId>

<version>4.2.0</version>

<scope>test</scope>

</dependency>

### **Troubleshooting:**

In case you see errors and not sure if the dependencies got downloaded well,

1. Perform a maven build to import all dependencies.
2. Still, you see errors, then do a maven clean followed by a maven install, and it should build without any errors.
3. You can add the below lines in your java class and see no compile errors are present.

import io.restassured.RestAssured.\*;

import io.restassured.matcher.RestAssuredMatchers.\*;

import org.hamcrest.Matchers.\*;

## **First script**

Given().

param("x", "y").

header("z", "w").

when().

Method().

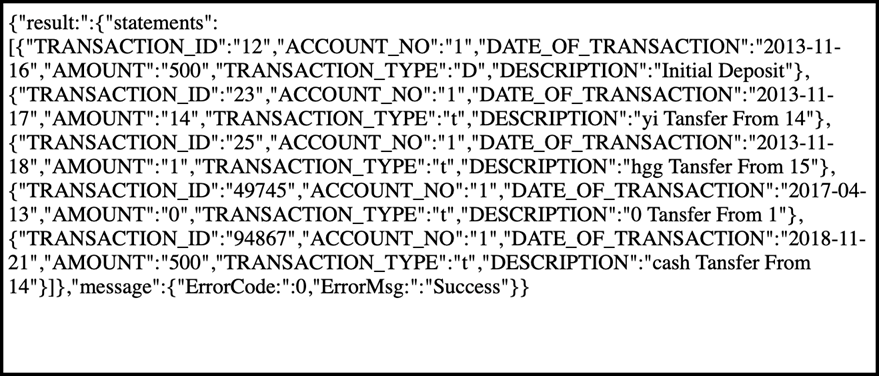
Then().

statusCode(XXX).

body("x, ”y", equalTo("z"));

|  |  |
| --- | --- |
| Code | Explanation |
| Given() | ‘Given’ keyword, lets you set a background, here, you pass the request headers, query and path param, body, cookies. This is optional if these items are not needed in the request |
| When() | ‘when’ keyword marks the premise of your scenario. For example, ‘when’ you get/post/put something, do something else. |
| Method() | Substitute this with any of the CRUD operations(get/post/put/delete) |
| Then() | Your assert and matcher conditions go here |

Now that you have the setup and some background to the syntax, let’s create our first test.



In case you get an error on the browser when you try to get a response for the request,

1. See if you have used Https or Http. Your browser might have settings to not open insecure websites.
2. See if you have any proxy or firewall blocks your browser from opening websites.

**Note**: You did not use any headers here, no body, and no cookie. It was a URL and also you are getting content from the API and not posting or updating any existing content, so that makes it a GET call. Remember this to understand our first test better.

### **The Objective of the test:**

The goal of the script is to print the same output on your IDE console as what you received on the browser.

Let us code this with the below steps:

### Getting the response Body

**Step 1)** Create a class named as “myFirstRestAssuredClass”

**Step 2)** Create a method called “getResponseBody”

**Step 3)** Similar to the structure learned earlier of given, when and then, type the below code

given(). -> No headers required, no query or path param.

when(). -> No specific condition setup

then(). -> No specific assertions required

log(). all() -> Once all the response is fetched, log response, headers, essentially everything that the request returns to you.

public static void getResponseBody(){

given().when().get("Account\_No=1").then().log()

.all();

}

Now notice that the URL used is long and less readable, if you look closely, you will notice that 3 query parameters are being used which are

1. Customer\_ID
2. Password
3. Account\_No

Rest Assured, helps us pass every part(query, path, header param) separately, making the code more readable and easy to maintain. Also, we can parameterize the data from an external file as required.

For using query param, we go back to our definition of the syntax and see that all of them are passed as a part of given.

public static void getResponseBody(){

given().queryParam("CUSTOMER\_ID","68195")

.queryParam("PASSWORD","1234!")

.queryParam("Account\_No","1")

.when().get().then().log()

.body();

}

Note that we used “body” instead of “all”; this helps us to extract only the body of the response.

### **Getting the response status code**

The next method that we script will be to get the status code and also put an assertion to validate the same.

**Step 1)** Create a method called getResponseStatus()

**Step 2)** Use the same request structure used above. Copy and paste it.

**Step 3)** Instead of logging it, we use the ‘getStatusCode’ inbuilt method of Rest Assured to fetch the status code value

**Step 4)** In order to assert that your status code is 200, we use the keywords – assertThat().statusCode(expectedCode)

**Note:** URL is a variable used for simplicity. URL holds the entire API request URL

public static void getResponseStatus(){

int statusCode= given().queryParam("CUSTOMER\_ID","68195")

.queryParam("PASSWORD","1234!")

.queryParam("Account\_No","1") .when().get().getStatusCode();

System.out.println("The response status is "+statusCode);

given().when().get(url).then().assertThat().statusCode(200);

}

**Business Need**

One of the basic rules of automation is that we have to put checkpoints so that the test proceeds only if all the required conditions are met. In API testing, the most basic validation is to check if the status code of the request is in 2XX format.

**Note:**

1. 200 is a successful response for this scenario. At times, you need the request to fail as well, and then you might use 4XX or 5XX. Do try to change the status code by supplying invalid parameters and check.
2. When we assert a condition, there will be no printing on the console unless there is an error.

## **Script to fetch different parts of a response**

Fetching response body and response status code is already covered in the above segment. It is worthy to note that to fetch different parts of the response, the keyword ‘extract’ is very important.

### **Header**

Rest Assured is a very straightforward language, and fetching headers is just as simple. The method name is headers(). Like before, we will create a standalone method to do the same.

public static void getResponseHeaders(){

System.out.println("The headers in the response "+

get(url).then().extract()

.headers());

}

Please note that ‘given().when()’ is skipped here, and the code line starts from get(), this is because there is no precondition or verification made here to hit the request and get a response. In such cases, it’s optional to use.

**Business Need:**

Quite a few times, you would need to use the authorization token, or a session cookie for the subsequent request, and mostly, these details are returned as headers of the response.

### **Response Time**

To get the time needed to fetch the response from the backend or other downstream systems, Rest Assured provides a method called ‘timeIn’ with a suitable timeUnit to get the time taken to return the response.

public static void getResponseTime(){

System.out.println("The time taken to fetch the response "+get(url)

.timeIn(TimeUnit.MILLISECONDS) + " milliseconds");

}

**Business Need:**

A very important feature of testing APIs is their response time, to measure the performance of the application. Note that the time taken for your call may take more or less time depending on your internet speed, the performance of the API at that time, server load, and other factors impacting the time.

### **Content-Type**

You can get the content-Type of the response returned using the method is “contentType ()”.

public static void getResponseContentType(){

System.out.println("The content type of response "+

get(url).then().extract()

.contentType());

}

**Business Need:**

At times getting the content-type is essential for ensuring there are no security gaps for any cross-origin threats or just to ensure the content passed is as per the standards of the API.

### **Fetch Individual JSON Element**

From the given response, you are asked to calculate the total amount, you need to fetch every amount and sum it up.

**Steps:**

**Step 1)** The amount field is within an array with Key “statements” which is in turn in the list with key “result”

**Step 2)** Rest Assured, provides a mechanism to reach the values in the API using “path”

**Step 3)** The path to reach amounts is “result.statements.AMOUNT”. Think of it like Xpath in selenium.

**Step 4)** Fetch all amounts in a collection, and then loop for all values to calculate the sum

public static void getSpecificPartOfResponseBody(){

ArrayList<String> amounts = when().get(url).then().extract().path("result.statements.AMOUNT") ;

int sumOfAll=0;

for(String a:amounts){

System.out.println("The amount value fetched is "+a);

sumOfAll=sumOfAll+Integer.valueOf(a);

}

System.out.println("The total amount is "+sumOfAll);

}

Note: Since the amount value is in string data type, we convert to integer and use it for summation.

## **Summary:**

* Rest Assured is a group of java libraries which enables us to automate Rest API testing
* Rest Assured is Java-based, and knowledge of core Java suffices for learning it
* It helps fetch values of request and response from complicated JSON structures
* The API request can be customized with a variety of header, query, path param, and any session or cookies to be set.
* It helps set assert statements and conditions.
* While Rest Assured is very helpful when the response is JSON type, it’s methods may not work seamlessly if content type id HTML or plain text.

# Right Code for API testing

import java.util.ArrayList;

import static io.restassured.RestAssured.\*;

import static java.util.concurrent.TimeUnit.MILLISECONDS;

public class myFirstRestAssuredClass {

final static String

public static void main(String args[]) {

getResponseBody();

getResponseStatus();

}

//This will fetch the response body as is and log it. given and when are optional here

public static void getResponseBody(){

given().when().get(url).then().log()

.all();

given().queryParam("CUSTOMER\_ID","68195")

.queryParam("PASSWORD","1234!")

.queryParam("Account\_No","1") .when().get().then().log().body();

}

public static void getResponseStatus(){

int statusCode= given().queryParam("CUSTOMER\_ID","68195")

.queryParam("PASSWORD","1234!")

.queryParam("Account\_No","1")

.when().get().getStatusCode();

System.out.println("The response status is "+statusCode);

given().when().get(url).then().assertThat().statusCode(200);

}

}

public static void getResponseBody(){

given().queryParam("CUSTOMER\_ID","68195")

.queryParam("PASSWORD","1234!")

.queryParam("Account\_No","1")

.when().get().then().log()

.body();

}

public static void getResponseStatus(){

int statusCode= given().queryParam("CUSTOMER\_ID","68195")

.queryParam("PASSWORD","1234!")

.queryParam("Account\_No","1") .when().get().getStatusCode();

System.out.println("The response status is "+statusCode);

given().when().get(url).then().assertThat().statusCode(200);

}

public static void getResponseHeaders(){

System.out.println("The headers in the response "+

get(url).then().extract()

.headers());

}

public static void getResponseTime(){

System.out.println("The time taken to fetch the response "+get(url)

.timeIn(TimeUnit.MILLISECONDS) + " milliseconds");

}

public static void getResponseContentType(){

System.out.println("The content type of response "+

get(url).then().extract()

.contentType());

}

public static void getSpecificPartOfResponseBody(){

ArrayList<String> amounts = when().get(url).then().extract().path("result.statements.AMOUNT") ;

int sumOfAll=0;

for(String a:amounts){

System.out.println("The amount value fetched is "+a);

sumOfAll=sumOfAll+Integer.valueOf(a);

}

System.out.println("The total amount is "+sumOfAll);

}