Good afternoon everyone. This is Subhrayu here.

1. Currently I am working as an Automation Test Engineer in Cognizant having total 4 years 4 months of work experience in IT industry.
2. I have hands–on experience to automate web based & desktop-based application’s test cases.
3. Coming to my recent project that is with the AIG insurance. Here, to automate a life insurance web application’s test cases, we are using selenium web driver API, JAVA with Cucumber framework to implement BDD based approach. With the BDD based approach, we are also maintaining Page Object Model design pattern over here.
4. As an Automation test engineer, my work responsibility is interpreting and converting the automatable manual test cases into our BDD based cucumber automation framework.
5. Here, all of our automation codebase are stored in GitHub repository. I am responsible for merging the pull request of a 6-member automation team through GitHub on a daily basis.
6. Main challenges in this project was that we have to validate the UI application data with the backend database data. For this, what we have done like we have divided the entire flow in 3 steps.

In the first step, we have written the scripts that used to fetch the data from the database. Here we have accessed different databases, like snowflake, Cassandra, oracle. Script is generating the DB query in the runtime depending upon the test data and executing the query in the database and finally the result sets are stored in a excel file.

In the second step, we have written the scripts that used to fetch the data from the UI of the web application and stored it in a excel file.

In the third & final step, we have written the scripts that used to compare the UI & DB excel data and create a comparison excel report. In this way, we did some rigorous comparison between UI and database and our script is capable to detect the data difference between UI & DB for more than thousand field in less than a minute.

1. Now coming to the test case execution part. Here we are using Sauce lab for test case execution. It is a cloud-based platform where we can execute our test cases in different OS & browser version and device combinations. Through the sauce lab, we are running our test cases in parallel mode also.
2. Here, we are using maven as a build tool and we are triggering the automated test cases through the maven command.
3. We are also executing our automated smoke suite through CI/CD pipeline, Jenkins on a daily basis.
4. Recently, we are integrating JIRA Xray with our existing BDD framework. As an outcome of this integration, anybody can write the automated test cases in Gherkin languages in Jira itself without any core automation technical knowledge. In the same way, anyone can check the automated test case execution result from Jira only.

Now, the entire automation flow will be looks like – QA team will write their cucumber test cases using gherkin language in Xray Jira. Then, Automation team will export that cucumber test cases from xray and the exported feature file will be placed in the automation script development suite, accordingly script will be developed and finally while we will execute our test case from our IDE/ ci-cd pipeline, the execution result will be automatically populated in the JIRA using Xray rest API.

If I focus on my Selenium + Java experience:

1. I have hands-on experience on BDD Cucumber framework. I was directly involved with the framework development team to develop the Cucumber framework. 1st 2months of our current project, I worked for developing the Cucumber framework with selenium, java, TestNG. After that, I moved to the application automation team to automate one life insurance web application’s test cases.
2. Cucumber is a tool that supports Behavior Driven Development (BDD). It offers a way to write tests that anybody can understand, regardless of their technical knowledge.
3. If I will talk about the framework. Then I can divide our Cucumber framework in 3 layers:

* **Business Layer:** Where end users (business analysts, product owners, manual Tester, QA team) can write the test cases in simple English languages. This Language used in the cucumber framework is known as Gherkin. Here we are creating feature files with extension .feature. Inside the feature file, we are creating different scenarios or test cases. Under the scenario we are defining multiple test steps in given when then format.
* Feature − It is Feature Description – actually it describes about feature under test.
* Background − Background generally has the instruction on what to setup before each scenario runs. Whenever any step/steps are required to perform in each scenario then those steps need to be placed in Background. For Instance, if a user needs to login the application before each scenario then those steps can be placed in the background section.
* Scenario / Scenario outline– Here we are mentioning the test case description which we want to automate. This will basically reflect in our cucumber automate report.
* Given – It describes the pre-requisite for the test to be executed.
* When − It defines the trigger point for any test scenario execution. That means Test actions.
* Then − Then holds the expected result for the test to be executed.
* And - It provides the logical AND condition between any two statements. AND can be used in conjunction with GIVEN, WHEN and THEN statement.
* **Business Logic Layer:** Here, against each unique test step described in the feature file, we need to define a step definition method under a java class. And also, in this layer we are maintaining POM concept i.e. page object model concept. POM is a design pattern, which is commonly used in Selenium for Automating the Test Cases. The Page object is a java class which acts as an interface for the page of your Application under test. In this Design Pattern, web pages are represented by a corresponding java class and web elements are represented by the variables of the class and all interactions are possible through the methods of the class.
* **Logic Layer:** All the logical operations are done in this layer like –
  + Parsing configuration properties file through singleton class,
  + All the necessary web driver actions
  + Extended Report creation
  + Database connection and many more things.

This layer is already developed by the framework team.

1. There is a TestNG runner class, which is the heart of the framework. It acts as a bridge between Business & Business logic layer i.e., between feature file and step definition methods.

@CucumberOptions(  
features = {"src/test/java/com/project/testngCucumber/feature/"}  
,glue = {"com.project.testngCucumber.stepDef"}  
*//,plugin ={"pretty" , "html:OutputReport" , "json:OutputReport/feature"+".json"}*)  
public class Runner extends AbstractTestNGCucumberTests {  
public static List<String> *jsonFilesPath* = new ArrayList<>();  
@BeforeTest  
public void beforeTest(){  
String testName = Reporter.*getCurrentTestResult*().getTestContext().getCurrentXmlTest().getName();  
System.*out*.println("Test name::"+testName);  
String plugin = "pretty, html:OutputReport, json:OutputReport/"+testName+"\_"+ DateTimeUtils.*getCurrentTimeStamp*("ddMMyyyyHHmmss")+".json";  
*jsonFilesPath*.add("./"+plugin.split("json:")[1]);  
System.*setProperty*("cucumber.plugin",plugin);  
String tags = Reporter.*getCurrentTestResult*().getTestContext().getCurrentXmlTest().getParameter("tags");  
System.*setProperty*("cucumber.options","--tags '"+ tags + "'");  
}  
@AfterSuite  
public void afterSuit(){  
System.*out*.println("Json Files path: "+*jsonFilesPath*);  
new ExtendReporter(*jsonFilesPath*,"./ExtendReport","Cucumber Report");  
}  
}

*<?*xml version="1.0" encoding="UTF-8"*?>*<!DOCTYPE suite SYSTEM "<http://testng.org/testng-1.0.dtd>" *>*<suite name="RegressionTest" parallel="false">  
<listeners>  
<listener class-name="com.project.testngCucumber.report.ReportListner"></listener>  
</listeners>  
  
<test name="Assignment 01 (IE)">  
<parameter name="URL" value="<https://www.google.com/>"/>  
<parameter name="browser" value="ie"/>  
<parameter name="tags" value="@tc01 or @tc02"/>  
<classes>  
<class name="com.project.testngCucumber.runner.Runner"/>  
</classes>  
</test>  
<test name="Assignment 02 (IE)">  
<parameter name="URL" value="<https://www.google.com/>"/>  
<parameter name="browser" value="ie"/>  
<parameter name="tags" value="@tc01 or @tc02"/>  
<classes>  
<class name="com.project.testngCucumber.runner.Runner"/>  
</classes>  
</test>  
</suite>

1. So, the flow of execution will look like the following −

* Stakeholders write down the feature files.
* Step definition file will be created accordingly.
* Specify the testng xml file to run the series of test cases.
* Once we run/trigger the TestNG runner class −
* It will parse the Gherkin feature file.
* It will execute the methods written in the step definition file according to feature file statements.
* It will combine the test case result.
* It will build the test report in the specified format (which can be html/JSON).
* I am also involved to develop an excel comparison reporting part in our framework. Our requirement was to extract the different UI field value and stored in a excel file, in the same way we execute the corresponding query to the database and stored in another excel file & finally compared the both excel in order to check the data difference between UI & Database in a single excel report.

maven is a tool that can be used for building and managing any Java-based project. Maven does a lot of helpful task like-

1. We can easily build a project using maven tool.
2. We can add jars and other dependencies of the project easily using the help of maven.
3. Maven is very helpful for a project while updating central repository of JARs and other dependencies.
4. Using maven command mvn test we can execute our test script from the command line. Using this, we also run our testcases from Jenkins also.
5. **HOW DO YOU PARSE DATA SHEET IN YOUR FRAMEWORK?**

We configured 1 excel file with multiple sheets to store all the test data.

Used apache poi for parsing the excel file.

We used HashMap to digest entire workbook. Defined the hashmap as Map<String, Map<String, List<String>>>. Here key is a string and value is another HashMap. In key, we store sheetname, corresponding value is another HashMap where key is row number or any column’s value i.e. String and value is an ArrayList which is a collection of all column’s values for this specific row.

We make this excel parser class as a singleton class.

private Map<String, Map<String, List<String>>> data2 = new HashMap<>();

**public** **void** readXL() **throws** IOException {

File f1 = **new** File(xlsPath);

FileInputStream f2 = **new** FileInputStream(f1);

XSSFWorkbook wr = **new** XSSFWorkbook(f2);

**for** (**int** i = 0; i < wr.getNumberOfSheets(); i++) {

XSSFSheet sh = wr.getSheetAt(i);

**for** (**int** row = 1; row <= sh.getLastRowNum(); row++) {

XSSFRow rw = sh.getRow(row);

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

**for**(**int** columnIndex =0 ; columnIndex<rw.getLastCellNum() ; columnIndex++) {

rowData.add(getValueAsString(rw.getCell(columnIndex)));

}

data2.putIfAbsent(sh.getSheetName(), **new** HashMap<>());

Map<String,List<String>> sheetData = data2.get(sh.getSheetName());

sheetData.put(rowData.get(0), rowData);

}

}

wr.close();

}

To use this data,

**public** String getConfData(String sheetName, String key) {

List<String> listValue = data2.get(sheetName).get(key);

**return** listValue.get(1);

}

**public** String getConfData(String sheetName, String key,**int** columnIndex) {

List<String> listValue = data2.get(sheetName).get(key);

**return** listValue.get(columnIndex);

}

1. **CHALLENGES IN AUTOMATION TESTING USING SELENIUM**

I can say my achievement is that I come up with a solution to my team to overcome different challenges during automation project. I want to describe about few challenges which I & my team faced in my previous projects and how we overcome those.

TIMEOUT OR SYNCHRONIZATION ISSUE BETWEEN AUTOMATION SCRIPT & APPLICATION:

It is one of the most common challenges in Selenium test automation project even in every automation project. If we don’t handle this issue carefully, most of testcases might fail. From my previous experiences, I can say that more than 50% of scripts fail due to improper synchronization while executing automation testing. Though this problem can be avoided using smart waits like Implicit waits, Fluent waits, and explicit waits that are available within Selenium. Again, we can write our own custom method to handle the sync issues.

I have implemented one solution for this synchronization issue in my previous project. Problem was some time may be due to application slowness or internet connectivity issue or something else, some action which were performed by scripts were not reflecting in the application and as a results testcases are failing. This issue was mainly observed while clicking on a particular element and setting some values in an input field. So, my solution was using loop, do the actions repeatedly until some conditions are getting true. So, for click action, we will click continuously with a fixed time interval until some object is visible and for setting the text in the input field, set the text in the input field continuously until the value in the input field is getting equal with the text which we are providing. After implementing this solution, sometimes we observed that this loop is getting infinite loop when the application got hanged/feezed. So, to overcome this we implemented another exit criteria for this loop when application got hanged, that is limiting the loop iteration count. We normally made it 10. So, it will try to do the action for 10 times max. If then also application will not response, then we failed the testcase with proper details.

PARALLEL EXECUTION IN CUCUMBER FRAMEWORK:

In general, while we are running the cucumber test cases, we go to the Junit runner class and simply run this class as Junit test. Once a requirement comes from business analyst team to the automation team that testcases need to be executed parallelly. Requirement was to run all the Junit runner class located under a specific package parallelly. To implemented this, we faced lot of difficulties and finally after long R & D, we come up with a perfect solution.

Our solution was simple:

* We used java reflection concept here. (Reflection is an API which is used to examine or modify the behavior of methods, classes, interfaces at runtime. Java Reflection makes it possible to inspect classes, interfaces, fields and methods at runtime, without knowing the names of the classes, methods etc. It is also possible to instantiate new objects, invoke methods and get/set field values using reflection.)
* Using reflection we have created a method which takes package name as an input where all the junit classes were located and this method gives a class[] as output in the run time. That class array contains all the junit runner class which has @CucumberOption annotation. (I didn’t remember the exact implementation of this method, trying to share the concept which we were implemented).
* After getting this class[] from this method, created a for loop which will iterated for each and every junit class entry.
* Under this for loop, for every junit runner class, we created a thread and run this junit runner class.
* In order to trigger the junit class, we have a class Junit core. So, we just need to create an object of this class and call the method run().

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

Class[] classes = **null**;

classes = *getCucumberClasses*("com.selenium.cucumber.runWay");

List<Thread> threadList = **new** ArrayList<>();

**for** (Class class2 : classes) {

String jsonReportPath = "./"+*getCucumberOptionsAnnotationValue*(class2,class2.getAnnotationsByType(CucumberOptions.**class**),"json:").substring(5);

*jsonReportFiles*.add(jsonReportPath);

Thread t1 = **new** Thread(**new** Runnable() {

@Override

**public** **void** run() {

JUnitCore jUnitCore = **new** JUnitCore();

Result result = jUnitCore.run(class2);

System.***out***.printf("\*\*\*\*" + class2.getName() + "\*\*\*\*\*\nTest Run: %s, Failed: %s%n", result.getRunCount(), result.getFailureCount());} }, class2.getName());

threadList.add(t1); t1.start();}

**for**(Thread thread : threadList) {

**try** {thread.join();

} **catch** (InterruptedException e) {e.printStackTrace();}

}

**new** ExtendReporter(*jsonReportFiles*, ".\\Reports\\ExtendedReport", "Cucumber Framework, By Subhrayu");

}

Smart locators

Locators are the core part of any scripting and we need to keep on enhancing our XPath and CSS for script stability, because if [XPaths](http://learn-automation.com/how-to-write-dynamic-xpath-in-selenium/)are not proper then chances are very high that script might fail in upcoming releases. To overcome this, we need to handle dynamic elements carefully with dynamic XPath or using different functions, such as contains, ends with, starts-with, etc.

1. **WHAT IS YOUR EXPECTATIONS FROM THIS JOB?**

That’s a good question. Thank you for asking.

Actually, I’m looking for the opportunity to expand my learning in Automation testing. I want to utilize my skills and experience which I gained from my earlier work. And want to work in a team. So that, my work responsibility gives me a scope to enhance my knowledge and skills in order to cope with the latest technological changes.

Can you tell me exactly what I would be expected to do (what will be my work responsibilities) if I will get an opportunity to work with you?

Will I be working with any existing team?

Can you please suggest if I have any room for improvement?

1. **WHAT IS FRAMEWORK ARCHITECHTURE?**

There is a TestNG runner class, which is the heart of the framework. It acts as a bridge between Business & Business logic layer i.e., between feature file and step definition methods. This runner class basically a java class which extends AbstractTestNGCucumberTests class and we have to mention different properties of cucumber under @CucumberOptions annotation like feature, Glue, plugin. And here we are maintaining one configuration xml file through which we are configuring different parameters.