

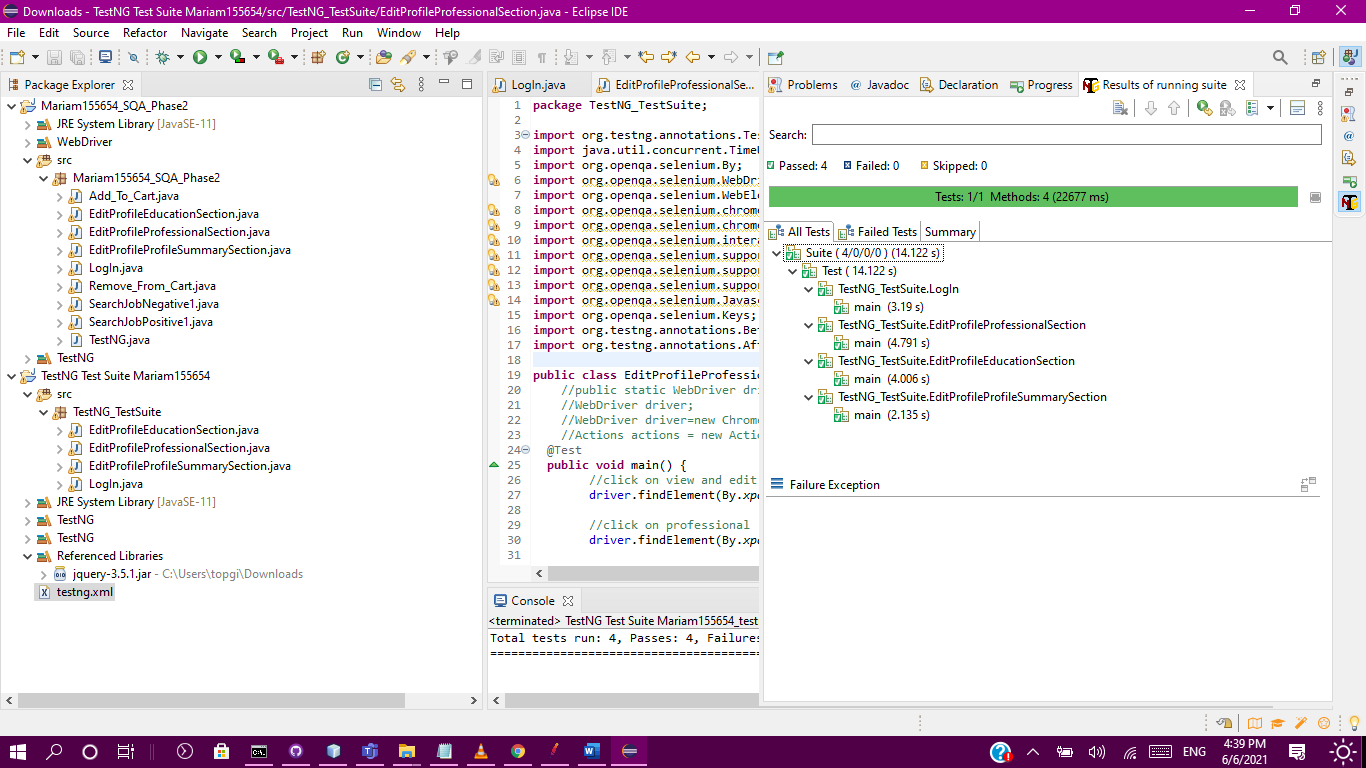
Software Quality Assurance

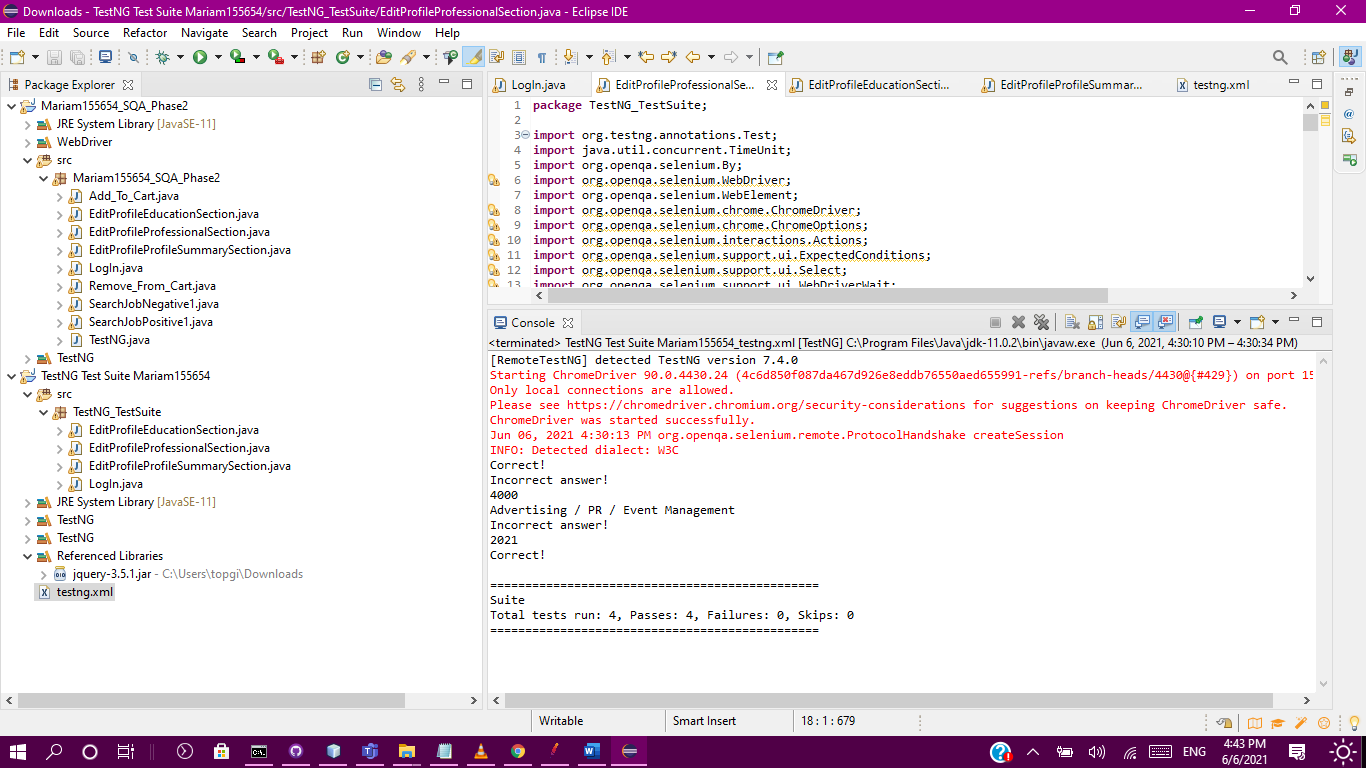
Final phase

|  |  |
| --- | --- |
| Name | ID |
| Mariam Mohamed Ibrahim | 155654 |

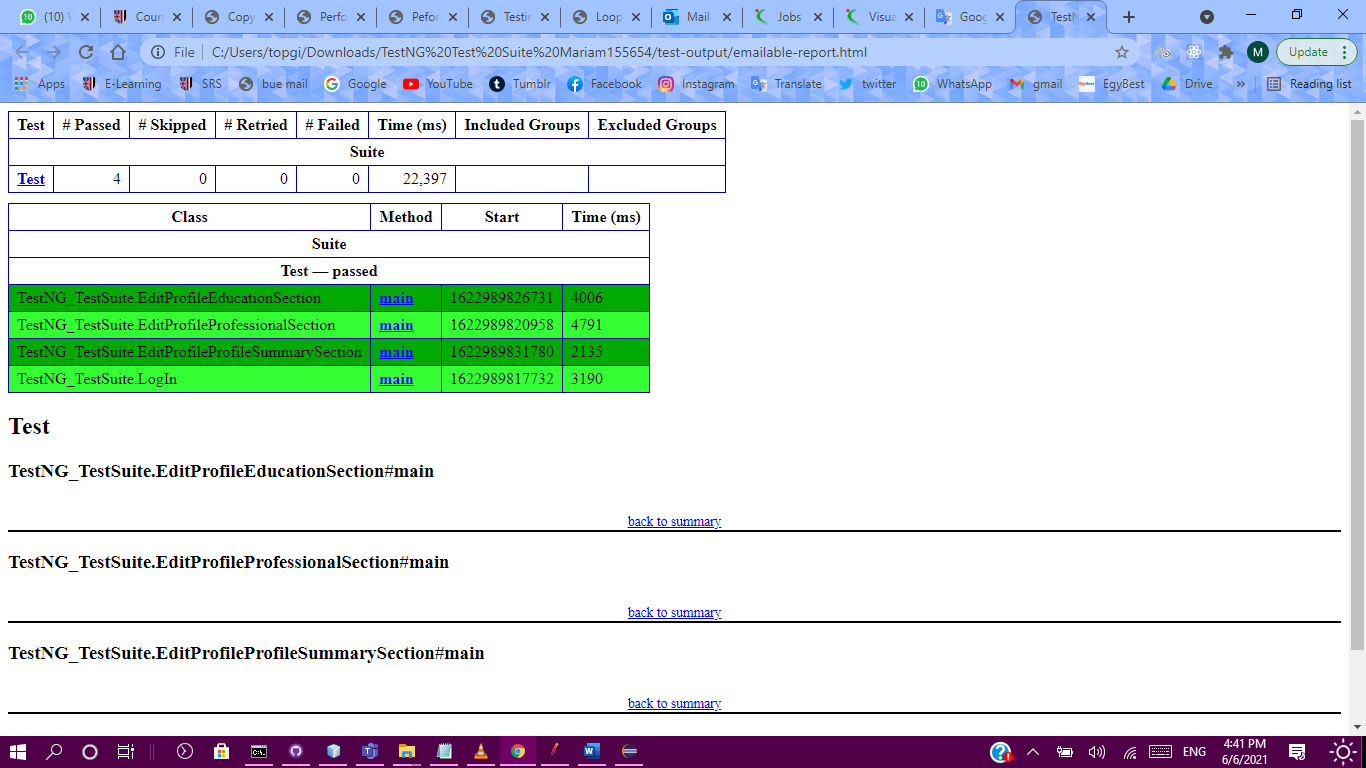
Mariam155654:

Result of running test suite in TestNG





Testing Results Report using TestNG



|  |  |
| --- | --- |
| ID | Performance\_1 |
| Test case | Alerts page access load |
| Objective | Verify that 100 users can add/view/edit or delete their alerts at same time. |
| Test scenario | 1. Create thread group. 2. Enter 100 for number of threads 3. Enter 10 for ramp up period 4. Create view results summary report 5. Create view results tree 6. Create view graph results |
| Input | * Number of threads (Users) * Ramp up period * Server URL/Path |
| Expected output | The accessing of the same alerts form page of all 100 users at the same time |
| Actual output | The accessing of the same alerts form page of all 100 users at the same time |
| Pass/Fail | Passed |

Group work:

Performance Test Case 1:

**JMeter Screenshot**

**Result Tree:**

Text

Description automatically generated

**Summary Report:**

Graphical user interface

Description automatically generated

* **Samples:** 100 requests for the web page
* **Average:** 1343 milliseconds is required to receive this webpage for 100 users.
* **Min and max:** The minimum amount of time required to receive the page is 1155, while the max value is 1808
* **Error:** All calls were received successfully which indicates 0 error.
* **Std Dev:** 128.64 exceptional cases were found.

**Graph Results:** A picture containing graphical user interface

Description automatically generated

Throughput: 280,164/ minute, this throughput value is a good value as the value of the throughput increases the better the performance.

Performance test case 2:

|  |  |
| --- | --- |
| ID | Performance\_2 |
| Test case | Visual cv writing access load |
| Objective | Check if 300 users access visual cv writing buy page at the same time, they will all success in accessing or not. |
| Test scenario | 1. Add thread group. 2. Enter 300 in the Number of Thread (Users). 3. Enter 60 in the ramp up period (Seconds) 4. Add listener view results tree 5. Add listener view results summary report 6. Add listener view graph results 7. Add listener view results in table |
| Input | * Number of threads (Users) * Ramp up period (Seconds) * Server Name of IP * URL path * HTTP request type but we remain it GET |
| Expected output | All users passed in accessing the webpage. |
| Actual output | All users passed in accessing the webpage. |
| Pass/Fail | Passed |

JMeter screen shots:

Text

Description automatically generated

A screenshot of a computer

Description automatically generated

Graphical user interface, text

Description automatically generated

A picture containing text, monitor, screenshot, electronics

Description automatically generated

* **Samples:** 300 requests for the web page
* **Average:** 528 milliseconds is required to receive this webpage for 300 users.
* **Min and max:** The minimum amount of time required to receive the page is 330 milliseconds, while the max value is 2069 milliseconds.
* **Error:** All calls were received successfully which indicates 0 error.
* **Std Dev:** 260.88 exceptional cases were found.

A screenshot of a computer

Description automatically generated with medium confidence

Throughput: 298.77/ minute, this throughput value is a good value as the value of the throughput increases the better the performance.

Standard Deviation: 260.88 which is the amount of error of the average and how close the request response time was to the average. This Standard Deviation is good. Since it shows the amount of deviation away from the average.

A computer screen capture

Description automatically generated with low confidence

60 seconds/300 threads. there is a 0.2 seconds delay in between each thread

Performance test case 3:

|  |  |
| --- | --- |
| ID | Performance\_3 |
| Test case | Visit Job By Location emirates page |
| Objective | Check if 300 users access Visit Job By Location specifically the emirates page without the website failing. |
| Test scenario | 1. Add thread group. 2. Enter 300 in the Number of Thread (Users). 3. Enter 80 in the ramp up period (Seconds) 4. Add listener view results tree 5. Add listener view results summary report 6. Add listener view graph results 7. Add listener view results in table |
| Input | * Number of threads (Users) * Ramp up period (Seconds) * Server Name of IP * URL path * HTTP request type but we remain it GET |
| Expected output | All users passed in accessing the webpage. |
| Actual output | All users passed in accessing the webpage. |
| Pass/Fail | Passed |

A picture containing text, screenshot, monitor, electronics

Description automatically generatedA picture containing text, monitor, screenshot, screen

Description automatically generated

Samples: 300 http get requests

Average: 1774 milliseconds for 300 users to access the webpage at once

Min 438 milliseconds at least were required for the user to receive a response for their http request

Standard Deviation: 1137.5 which is the amount of error of the average and how close the request response time was to the average. This Standard Deviation is quite high

Error: 0 which means that all users received a response.

Max: 4347 milliseconds maximum were required for the user to receive a response

A picture containing text, screenshot, monitor, electronics

Description automatically generated

The throughput of the page is 222.156/minute which is good since the website can accept 222.156 requests per 300 users.

Deviation: 1137 milliseconds which is high since it shows the amount of deviation away from the average. This high deviation shows that the website does not have high performance.

A picture containing text, monitor, screenshot, electronics

Description automatically generated

80 seconds/300 threads. there is a 0.27 seconds delay in between each thread.

Summary report for each tool:

**Selenium IDE**

What the tool does: it is a tool used for testing functionalities on websites. It tests the steps of the test case and when the steps are tested, the tester can make sure that the test case produced the correct output using commands that are build-in inside the IDE. After the test case steps are tested, the user can run it multiple times. It also can be used to create test suites and footprints and test them as well. It does not require much experience since all that it requires is recording and clicking buttons. The commands are clear as well and the documentation is simple and easy to read. Is the easiest of them. Simply you click on recording button and start performing the actions on the website then stop the record and make assertions of elements and texts. It was easy to debug.

What you did with the tool: we used it to test functionalities on Naukri gulf website. we also created test suites and footprints.

**Selenium Web Driver**

It is the hardest test tool and most time consuming. It has many problems and errors. You should write java code of every step you make in the testing.

What is this tool used for: this too is more comprehensive than Selenium IDE. However, its core functionality is the same. it tests functionalities on websites as well as testing test suites and foorprints. However, it requries more experience with code. To implement the Webdriver code, eclipse IDE was used. after the steps of the functionality are tested, the code of the webdriver makes sure that the test case outputs the desired output using asserts using functions implemented inside the webdriver library.

**TestNG**

It is more organized than web driver in dividing the parts of testing which are before testing and during testing and after testing. Also, it is more useful than selenium web driver because of the testing result report it generates after testing. Also easier in making test suite than selenium web driver.

What this tool is used for: it tests the features of the website. if the feature runs successfully and outputs the desired output, the TestNG will store the results in an html file. if the test case fails, it still shows the results in an html file. It also shows were the test case failed and if a test suite is being tested, it shows which function of the test suite cased it to fail and which passed. it is a comprehensive way of knowing which test cases worked.

**JMeter**

Very useful to know how much loads the website can tolerate. Very easy and fast. Clear results. It is putting every detail. It tests the performance of the website, it generates the view results tree which presents each user access information, summary report which summaries the whole process and tell the average time it takes to access all these users to the webpage, minimum and maximum time, standard deviation and the error percentage, graph results which presenting the results in graph, view results in table which tell if each user pass or fail. We should input the users’ number, the website URL and path, the ramp up period.

Issues and difficulties that the group have faced while working with any of the tools:

**Selenium IDE**

* So slow in choosing option from dropdown list.
* “Time out” error appeared several times.
* Problem in hovering.
* Some commands selenium IDE add it like “mouse over” is making problems/failures in testing.
* Much easier than selenium web driver.
* Slower in executing testing than selenium web driver.
* All test cases passed.
* Deals better with the problems of bad implementation of the website.
* Implementing test cases in selenium IDE is way faster than implementing it in Selenium web driver.

**Selenium Web Driver**

* Add to cart and remove from cart test cases failed because of selenium web driver poor quality and bad implementation of the website, because of “**no such element: unable to locate element**” error.
* Edit profile test suite, sometimes it is executed correctly in the run and sometimes it fails.
* Using old version of eclipse caused errors in mouse commands.
* Selenium web driver could not deal with the way that naukrigulf make the drop-down lists. Since this website did not make the drop-down menu in a normal way which the options have <select> tag. Until I thought of using mouse commands to solve this problem.
* I noticed that “enter” key mouse command do not work properly so after choosing option from menu and clicking enter I make it click F2 key three times and click anywhere in the form just to make the list don’t appear anymore because it is intercepting the rest of the fields and make “**element intercepted**” error.
* I just wanted to test if the mouse commands work properly so I tested F2 button (which is decreasing brightness in my laptop) to test if after clicking it three times it will lower the brightness or no, but it did not, so I removed it and noticed that “**element intercepted**” error appeared. So, I kept it. But sometimes, the error is still existing, so I thought of clicking the mouse anywhere in the form.
* By using this way of choosing from dropdown menu, sometimes web driver actually chooses what I wanted, and sometimes it chooses other option.
* Test suites should be done by making all the classes in the test suite without main and just functions and each class will call the driver function of the class that should be executed before it, and last class at the test suite have a main.
* Sometimes, when I compare between the string I expected and the value of an element, title of the page, or text of an element, even though they are the same strings, web driver can not realize that and appear that the test case is failed/incorrect.

**Errors at the Selenium web driver:**

* “**no such element: unable to locate element**” error appeared several times in different test cases. In some test cases I solved this error by using wait time, I kept increasing the wait time until the error is solved, sometimes, I solved it by waiting until the element is visible. In “add to cart” and “remove from cart” it couldn’t be solved, although I tried to solve it by all the ways, I tried finding element by other things than xpath, like cssSelector, ID, and full xpath, wait till the page is loaded, wait till the element is visible, scroll down to some pixels, scroll down till finding the element, choosing another buttons, wait time to small number of seconds, wait time to big number of seconds.
* I solved this error (“**no such element: unable to locate element**”) in search for job classes by using:

WebDriverWait wait = new WebDriverWait(driver, 10);

wait.until( ExpectedConditions.visibilityOfElementLocated(By.xpath("/html/body/div[1]/div[2]/main/div[1]/div[1]/div")));

* In “**element intercepted**” error, I solved it by after choosing options from the dropdown menu, I click anywhere in the form so that the list does not appear anymore.
* In “**element not interactable**” error, I solved it by using the full xpath not only the xpath.
* In “**element is not in the page**” error, different than no such element error, it happens in test suites, I solved it by waiting until the page is loaded after finishing test case and begin new test case.

**TestNG**

* Difficulty in importing the TestNG in Eclipse. It cannot be imported by the way that it is wrote in the lab pdf.
* Steps of making test suite is not clear.
* The website needs to be configured in a specific way so that the TestNG test case will work.
* Sometimes it skips test cases in test suite.
* In arranging which test cases should be executed first in the test suite, even though we specify it before executing the xml, the xml is generated with the wrong order.
* Error appears saying jQuery is missing.
* Even though the button of "others" (and any buttons for reason of deletion) that are displayed to delete an alert work in every other testing tool, it never works in testing and shows error that the element is intractable.
* It doesn't always navigate with the account logged in even though the log in test case is passed every time.

**JMeter**

* After testing more than 100 users to access the website at the same time using it, we could not access the website for a while, the website blocked our IPs.