# VISVESVARAYA TECHNOLOGICAL UNIVERSITY



**JNANASANGAMA, BELGAVI-590018 SOFTWARE TESTING REPORT (18IS62) ON**

# “FACEBOOK AUTOMATION USING SELENIUM”

Submitted in partial fulfillment of the requirements for the 6thSemester

INFORMATION SCIENCE AND ENGINEERING

**Submitted by**

**N DEEPAK 1BI19IS033**

**MIHIKA SRIVASTAVA 1BI19IS032**

**DANUSH KANCHI 1BI19IS015**

Under the guidance of Vedasree.T.K Assistant Professor Dept of ISE, BIT



# DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING BANGALORE INSTITUTE OF TECHNOLOGY

**K.R.Road, V.V.Puram, Bengaluru-560004**

|  |  |  |
| --- | --- | --- |
| **CHAPTER** | **CONTENTS** | **PAGE** |
| **1** | **INTRODUCTION** | **4** |
|  | **1.1 Testing Approach** | **5** |
|  | 1.1.1 Static, Dynamic and Passive Testing | **5** |
|  | 1.1.2 Exploratory Approach | **5** |
|  | 1.1.3 The Box Approach | **5-7** |
| **2** | **TESTING TOOLS** | **8** |
|  | **2.1 Selenium** | **8** |
|  | **2.2 Components** | **8** |
|  | 2.2.1 Selenium Core | **9** |
|  | 2.2.2 Selenium IDE | **9** |
|  | 2.2.3 Selenium RC | **10** |
|  | 2.2.4 Selenium Web Driver | **12** |
|  | 2.2.5 Selenium Grid | **13** |
|  | **2.3 TestNG** | **14** |
|  | **2.4 Framework** | **15** |
|  | **2.5 Problem definition** | **15** |

1. **IMPLEMENTATION**
   1. **Code**
2. **TESTING**
   1. **Test Report**
   2. **TestNG - Plug-in for IDE**
   3. **Snapshots**
3. **CONCLUSION**

**16**

**17-21**

**22**

**23**

**23**

**24-28**

**29**

**CHAPTER 1**

**INTRODUCTION**

Software testing is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding failures and verifying that the software product is fit for use. Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

* meets the requirements that guided its design and development,
* responds correctly to all kinds of inputs,
* performs its functions within an acceptable time,
* is sufficiently usable,
* can be installed and run in its intended environments
* Achieves the general result its stakeholder’s desire.

As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding failures due to software faults. Software testing can provide objective, independent information about the quality of software and risk of its failure to users or sponsors.

Software testing can be conducted as soon as executable software (even if partially complete) exists. The overall approach to software development often determines when and how testing is conducted. For example, in a phased process, most testing occurs after system requirements have been defined and then implemented in testable programs. In contrast, under an agile approach, requirements, programming, and testing are often done concurrently.

# TESTING APPROACH

There are many approaches available in testing. Reviews, walkthroughs, or inspections are referred to as static testing, whereas executing programmed code with a given set of test cases is referred to as dynamic testing.

## Static, Dynamic and Passive Testing

Static testing is often implicit, like proofreading, plus when programming tools/text editors check source code structure or compilers (pre-compilers) check syntax and data flow as static program analysis. Dynamic testing takes place when the program itself is run. Dynamic testing may begin before the program is 100% complete in order to test particular sections of code and are applied to discrete functions or modules. Typical techniques for these are either using stubs/drivers or execution from a debugger environment.

Static testing involves verification, whereas dynamic testing also involves validation.

Passive testing means verifying the system behavior without any interaction with the software product. Contrary to active testing, testers do not provide any test data but look at system logs and traces. They mine for patterns and specific behavior in order to make some kind of decisions. This is related to offline runtime verification and log analysis.

## Exploratory Approach

Exploratory testing is an approach to software testing that is concisely described as simultaneous learning, test design, and test execution. Cam Kaner, who coined the term in 1984, defines exploratory testing as a style of software testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the quality of his/her work by treating test related learning, test design, test execution, and test result interpretation as mutually supportive activities that run in parallel throughout the project.

## The Box Approach

Software testing methods are traditionally divided into white box and black box testing. These two approaches are used to describe the point of view that the tester takes when designing test cases. A hybrid approach called grey box testing may also be applied to software testing methodology.

**White Box Testing**

White-box testing (also known as clear box testing, glass box testing, transparent box testing and structural testing) verifies the internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing, an internal perspective of the system (the source code), as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs.

White box testing can be applied at the unit, integration, and system levels of the software testing process; it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements. Techniques used in white-box testing include:

* API Testing - testing of the application using public and private APIs (application programming interfaces)
* Code Coverage - creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)
* Fault Injection Methods - intentionally introducing faults to gauge the efficacy of testing strategies
* Mutation Testing Methods
* Static Testing Methods

Code coverage tools can evaluate the completeness of a test suite that was created with any method, including black-box testing. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested. Code coverage as a software metric can be reported as a percentage for:

* + Function coverage, which reports on functions executed
  + Statement coverage, which reports on the number of lines executed to complete the test
  + Decision coverage, which reports on whether both the True and the False branch of a given test has been executed

**Black Box Testing**

Black-box testing (also known as functional testing) treats the software as a black box, examining functionality without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it. Black-box testing methods includes, equivalence partitioning, boundary value analysis, all pairs testing, state transition tables, decision table testing, fuzz testing, model based testing, use case testing, exploratory testing, and specification-based testing.

Specification based testing aims to test the functionality of software according to the applicable requirements. This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value. Test cases are built around specifications and requirements that is what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. These tests can be functional or non-functional, though usually functional.

One advantage of the black box technique is that no programming knowledge is required. Whatever biases the programmers may have had, the tester likely has a different set and may emphasize different areas of functionality. On the other hand, black-box testing has been said to be like a walk in a dark labyrinth without a flashlight as they do not examine the source code, there are situations when a tester writes many test cases to check something that could have been tested by only one test case or leaves some parts of the program untested.

**Grey Box Testing**

Grey box testing involves having knowledge of internal data structures and algorithms for purposes of designing tests while executing those tests at the user, or black-box level. The tester will often have access to both the source code and the executable binary. Grey-box testing may also include reverse engineering to determine, for instance, boundary values or error messages. Manipulating input data and formatting output do not qualify as grey box, as the input and output are clearly outside of the black box that we are calling the system under test. This distinction is particularly important when conducting integration testing between two modules of code written by two different developers, where only the interfaces are exposed for the test.

**CHAPTER 2**

# TESTING TOOLS

Testing Tools in software testing can be defined as products that support various test activities starting from planning, requirement gathering, build creation, test execution, defect logging and test analysis. These testing tools are mainly used for testing software firmness, thoroughness, and other performance parameters. There are tons of software testing tools available in the market, and with the plethora of choices it becomes difficult to zero in on the best testing tools. In this project we use two sophisticated testing tools as follows:

**2.1 SELENIUM**

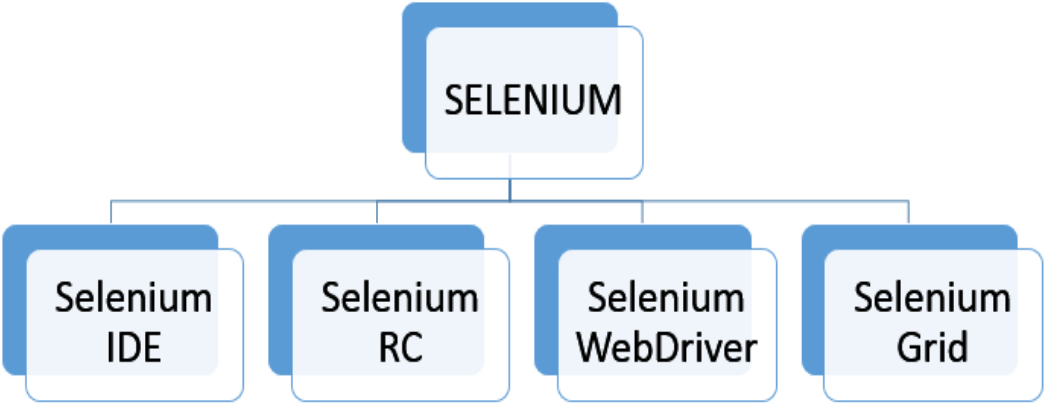
Selenium was originally developed by Jason Huggins in 2004 as an internal tool at ThoughtWorks. Huggins was later joined by other programmers and testers at ThoughtWorks, before Paul Hammant joined the team and steered the development of the second mode of operation that would later become "Selenium Remote Control" (RC). The name Selenium comes from a joke made by Huggins in an email, mocking a competitor named Mercury, saying that you can cure mercury poisoning by taking selenium supplements. The others that received the email took the name and ran with it.

Selenium is a portable framework for testing web applications. Selenium provides a playback tool for authoring functional tests without the need to learn a test scripting language (Selenium IDE). It also provides a test domain-specific language (Selenese) to write tests in a number of programming languages, including C#, Groovy, Java, Perl, PHP, Python, Ruby and Scala. The tests can then run against most modern web browsers. Selenium runs on Windows, Linux, and macOS. It is an open source software released under the Apache License 2.0.

# COMPONENTS

Selenium is composed of several components with each taking on a specific role in aiding the development of web application test automation

Components of Selenium is as shown in Figure 2.1



## Figure 2.1 Components of Selenium

**2.2.1 Selenium Core**

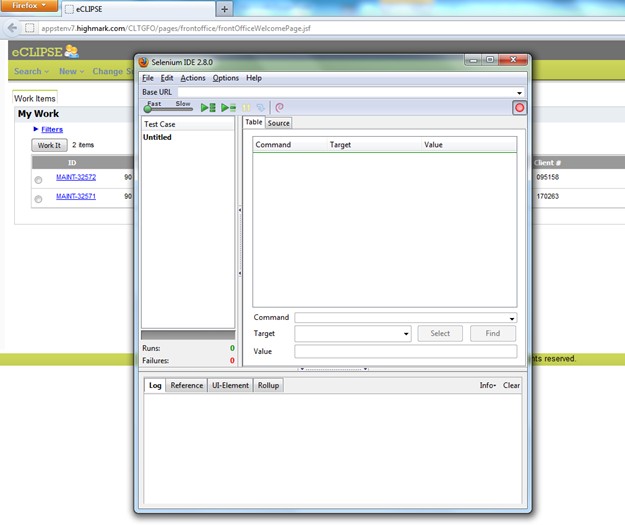
Selenium Core is a component core JavaScript library developed by Jason Huggins and Thought Works team. Selenium Core is the foundation for other components like Selenium IDE.

## Selenium IDE

Selenium IDE is a complete integrated development environment (IDE) for Selenium tests. It is implemented as Firefox Add-On and as a Chrome Extension. It allows for recording, editing and debugging of functional tests. It was previously known as Selenium Recorder. Selenium-IDE was originally created by Shinya Kasatani and donated to the Selenium project in 2006. Selenium IDE was previously little-maintained. Selenium IDE began being actively maintained in 2018.

Scripts may be automatically recorded and edited manually providing auto completion support and the ability to move commands around quickly. Scripts are recorded in Selenese, a special test scripting language for Selenium. Selenese provides commands for performing actions in a browser (click a link, select an option) and for retrieving data from the resulting pages. Figure

* 1. refers to Selenium IDE



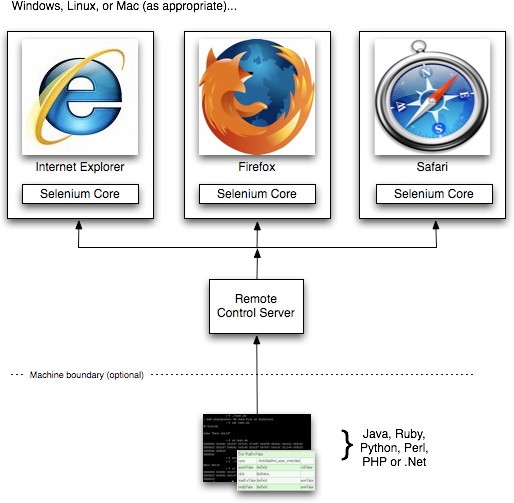
## Figure 2.2 Selenium IDE

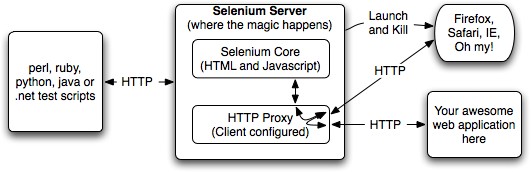
**2.2.3 Selenium RC**

Selenium Remote Control (RC) is a server, written in Java that accepts commands for the browser via HTTP. RC makes it possible to write automated tests for a web application in any programming language, which allows for better integration of Selenium in existing unit test frameworks. To make writing tests easier, Selenium project currently provides client drivers for PHP, Python, Ruby, .NET, Perl and Java.

An instance of selenium RC server is needed to launch html test case, which means that the port should be different for each parallel run. However, for Java/PHP test case only one Selenium RC instance needs to be running continuously. Selenium RC served as the flagship testing

framework of the entire project of selenium for a long - standing times and significantly Selenium RC is the first and foremost automated web testing tool that enabled users to adopt their preferred programming language. Selenium with the release of Selenium 2, Selenium RC has been officially deprecated in favor of Selenium Web Driver. Figure 2.3 refers to Selenium RC





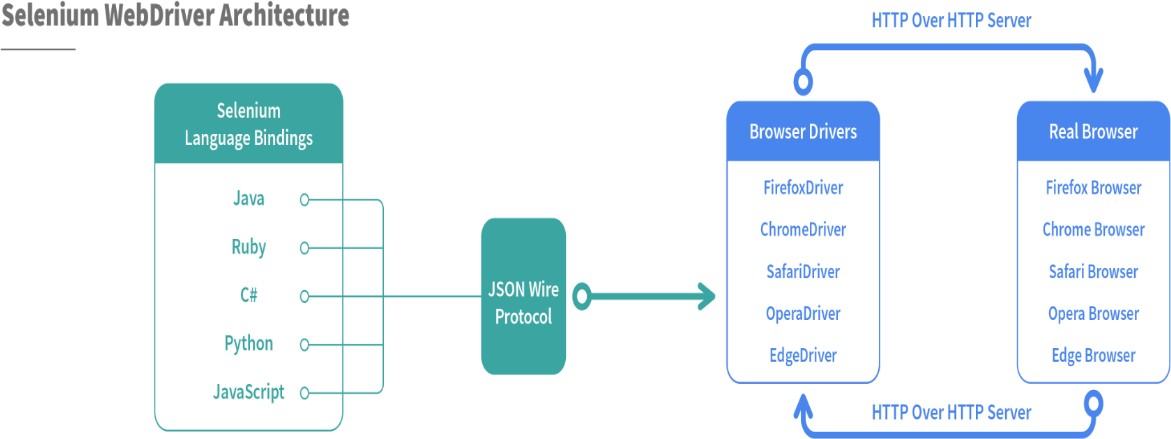
## Figure 2.3 Selenium Remote Control

**2.2.4 Selenium Web Driver**

Selenium Web Driver is the successor to Selenium RC. Selenium Web Driver accepts commands (sent in Selenese, or via a Client API) and sends them to a browser. This is implemented through a browser-specific browser driver, which sends commands to a browser and retrieves results. Most browser drivers actually launch and access a browser application (such as Firefox, Google Chrome, Internet Explorer, Safari, or Microsoft Edge), there is also an HtmlUnit browser driver, which simulates a browser using the headless browser HtmlUnit. Selenium Web Driver (Selenium 2.0) is fully implemented and supported in Python, Ruby, Java, and C#.

Selenium Grid can be used with Web Driver to execute tests on remote systems (see below). Where possible, Web Driver uses native operating system level functionality rather than browser-based JavaScript commands to drive the browser. This bypasses problems with subtle differences between native and JavaScript commands, including security restrictions

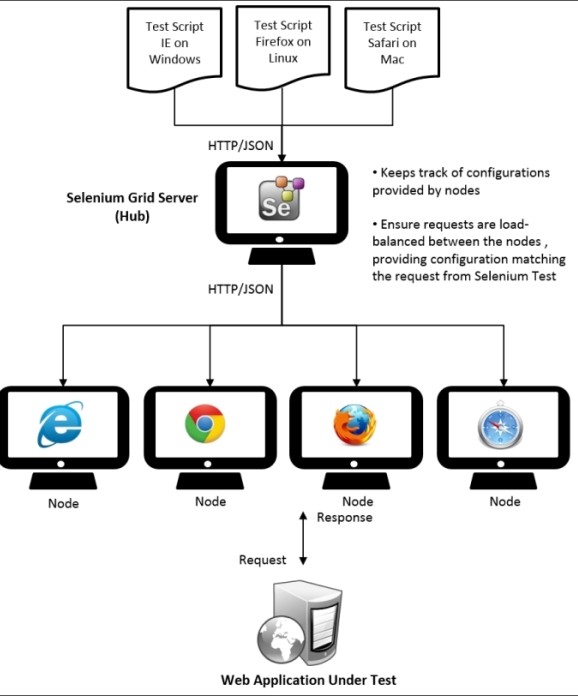
Selenium 2.0 aims to provide a basic set of building blocks from which developers can create their own domain-specific language (DSL). The Watir project in the Ruby language has a rich history of good design. Watir Web Driver is created entirely automatically, based on the Web Driver specification and the HTML specification. Figure 2.4 refers to Selenium Web Driver Architecture



**Figure 2.4 Selenium Web Driver Architecture**

# 2.2.5 Selenium Grid

Selenium Grid is a server that allows tests to use web browser instances running on remote machines. With Selenium Grid, one server acts as the central hub. Tests contact the hub to obtain access to browser instances. The hub has a list of servers that provide access to browser instances (WebDriver nodes), and lets tests use these instances. Selenium Grid allows running tests in parallel on multiple machines and to manage different browser versions and browser configurations centrally (instead of in each individual test).

The ability to run tests on remote browser instances is useful to spread the load of testing across several machines and to run tests in browsers running on different platforms or operating systems. The latter is particularly useful in cases where not all browsers to be used for testing can run on the same platform. Figure 2.5 refers to Selenium Grid working

**Figure 2.5 Selenium Grid**

# 2.3 TestNG

* TestNG is a very important framework when you are actually developing the framework from scratch level.
* TestNG provides you full control over the test cases and the execution of the test cases. Due to this reason, TestNG is also known as a testing framework.
* Cedric Beust is the developer of a TestNG framework.
* If you want to run a test case A before that as a pre-request you need to run multiple test cases before you begin a test case A. You can set and map with the help of TestNG so that pre-request test cases run first and then only it will trigger a test case A. In such way, you can control the test cases.
* TestNG framework came after Junit, and TestNG framework adds more powerful functionality and easier to use.
* It is an open source automated TestNG framework. In TestNG, NG stands for "**Next Generation**".
* TestNG framework eliminates the limitations of the older framework by providing more powerful and flexible test cases with help of easy annotations, grouping, sequencing and parametrizing.

**Diagram

Description automatically generated**

# 2.4 FRAMEWORKS

## Configuration of Selenium Web Driver in Eclipse

Writing a test case in any programming language is almost the same as writing code to develop an application. Before starting the coding for any application, we need to do some setup and configurations in the IDE, which we will use to develop. Following the same, before starting the development of automation test cases using Selenium Web Driver, we need to perform specific configurations in the Eclipse IDE***.*** It ensures it has all the dependencies which we need for the development of the test cases. So, before using the Selenium Web Driver for Java for the test case development, we need to Configure Eclipse***.***

Firstly here we have configured chrome to be the default web driver for our project. The major external JAR files to be installed in order to execute the selenium web driver in Eclipse IDE are-

* + - * client-combined-3.141.59.jar
      * client-combined-3.141.59-sources.jar
      * byte-buddy-1.8.15.jar
      * commons-exec-1.3
      * guava-25.0-jre
      * okhttp-3.11.0
      * okio-1.14.0.jar

# 2.5 PROBLEM DEFINITION

Our activity mainly focuses on testing the basic features of handling a Social media Account. In this case, we have taken the example of facebook account. This involves logging in and out of our account, verifying the account that we logged into, editing the account details such as profile pic updation, adding or updating personal details, traversing through different tabs and so on..

The main objectives of this program-

* To automate the process of logging in and out from our account.
* To handle tabs traversal.
* To test out the add and update features of the account.

**CHAPTER 3**

**IMPLEMENTATION**

## Java

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a [general-purpose](https://en.wikipedia.org/wiki/General-purpose_language) programming language intended to let [application developers](https://en.wikipedia.org/wiki/Application_developer) write once, run anywhere (WORA) meaning that [compiled](https://en.wikipedia.org/wiki/Compiler) Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to [byte code](https://en.wikipedia.org/wiki/Java_bytecode) that can run on any [Java](https://en.wikipedia.org/wiki/Java_virtual_machine) [virtual machine](https://en.wikipedia.org/wiki/Java_virtual_machine) (JVM) regardless of the underlying [computer architecture](https://en.wikipedia.org/wiki/Computer_architecture). The [syntax](https://en.wikipedia.org/wiki/Syntax_(programming_languages)) of [Java](https://en.wikipedia.org/wiki/Java_(software_platform)) is similar to [C](https://en.wikipedia.org/wiki/C_(programming_language)) and [C++,](https://en.wikipedia.org/wiki/C%2B%2B) but has fewer [low-level](https://en.wikipedia.org/wiki/Low-level_programming_language) facilities than either of them. The Java runtime provides dynamic capabilities (such as reflection and runtime code modification) that are typically not available in traditional compiled languages.

## Advantages of Java For Selenium Tool Testing

A good community of developers to create documentation and resolve issues has helped Java to become the most preferred language among the application developers. Thus, writing selenium test cases using Java has multiple benefits:

* Selenium supports Java. So, testers can leverage the active community of contributors and detailed documentation to write test cases.
* Programs written in Java are faster than other popular languages like Python.
* Java is more widely used in commercial applications as compared to other programming languages like Python and hence integrating Selenium tests it easier.

Eventually choosing the right language varies by project, organization, and individuals driving it. An essential criterion is to know the language in-depth when dealing with Selenium.

**3.1 CODE**

**package** fbtestdemo;

**import** java.util.concurrent.TimeUnit;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.JavascriptExecutor;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.testng.annotations.AfterTest;

**import** org.testng.annotations.BeforeTest;

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

**public** **class** fbdemo {

WebDriver driver;

@BeforeTest

**public** **void** Test1() **throws** InterruptedException {

System.*setProperty*("webdriver.chrome.driver", "C:\\selenium\\chromedriver\_win32\\chromedriver.exe");

driver = **new** ChromeDriver();

driver.get("http://www.facebook.com/");

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

System.***out***.println(driver.getTitle());

Thread.*sleep*(1000);

}

@Test

**public** **void** Test2() **throws** InterruptedException {

**try** {

driver.getCurrentUrl();

WebElement email = driver.findElement(By.*id*("email"));

email.sendKeys("\*\*\*\*\*\*\*\*\*");

WebElement password = driver.findElement(By.*id*("pass"));

password.sendKeys("\*\*\*\*\*\*\*\*\*\*");

WebElement login = driver.findElement(By.*name*("login"));

login.click();

Thread.*sleep*(100);

System.***out***.println("Login successful");

Thread.*sleep*(50);

}

**catch** (Exception e) {

System.***out***.println("Test2 failed");

System.***out***.println(e);

}

}

@Test

**public** **void** Test3() **throws** InterruptedException{

**try** {

Thread.*sleep*(10000);

WebElement mydetails = driver.findElement(By.*xpath*("//div[@role='navigation']"));

((JavascriptExecutor)driver).executeScript("arguments[0].removeAttribute('style')", mydetails);

WebElement account = mydetails.findElement(By.*xpath*("//div[@aria-label='Your profile']"));

account.click();

Thread.*sleep*(1000);

WebElement user = account.findElement(By.*xpath*("(//span[@dir='auto'])[11]"));

String username = user.getText();

//System.out.println(username);

**if**(username.equalsIgnoreCase("Deepak N")) {

System.***out***.println("Account Verified Successfully");

}

Thread.*sleep*(100);

user.click();

}

**catch** (Exception e) {

System.***out***.println("Test3 failed");

System.***out***.println(e);

}

}

@Test

**public** **void** Test4() **throws** InterruptedException{

**try** {

driver.getCurrentUrl();

Thread.*sleep*(5000);

JavascriptExecutor js2 = (JavascriptExecutor) driver;

js2.executeScript("window.scrollBy(0,170)", "");

Thread.*sleep*(10000);

driver.findElement(By.*xpath*("(//a[@role='tab' and @tabindex='0'])[4]")).click();

Thread.*sleep*(5000);

JavascriptExecutor js3 = (JavascriptExecutor) driver;

js3.executeScript("window.scrollBy(0,220)", "");

Thread.*sleep*(5000);

WebElement tabp=driver.findElement(By.*xpath*("(//div[@role='button' and @tabindex='0' and @aria-label='Edit'])"));

((JavascriptExecutor)driver).executeScript("arguments[0].removeAttribute('style')", tabp);

tabp.click();

Thread.*sleep*(5000);

JavascriptExecutor js4 = (JavascriptExecutor) driver;

js4.executeScript("window.scrollBy(0,180)", "");

Thread.*sleep*(5000);

driver.findElement(By.*xpath*("(//div[@role='menuitem' and @tabindex='0'])[4]")).click();

JavascriptExecutor js5 = (JavascriptExecutor) driver;

js5.executeScript("window.scrollBy(0,320)", "");

Thread.*sleep*(1500);

driver.findElement(By.*xpath*("//div[@aria-label='Save']")).click();

System.***out***.println("Profile pic updated");

Thread.*sleep*(1500);

JavascriptExecutor js6 = (JavascriptExecutor) driver;

js6.executeScript("window.scrollBy(0,-450)", "");

Thread.*sleep*(500);

}

**catch** (Exception e) {

System.***out***.println("Test4 failed");

System.***out***.println(e);

}

}

@Test

**public** **void** Test5() **throws** InterruptedException{

**try** {

driver.getCurrentUrl();

Thread.*sleep*(5000);

WebElement edit = driver.findElement(By.*xpath*("//div[@aria-label='Edit profile']"));

edit.click();

Thread.*sleep*(5000);

JavascriptExecutor js2 = (JavascriptExecutor) driver;

js2.executeScript("window.scrollBy(0,550)", "");

Thread.*sleep*(5000);

WebElement hobbies = edit.findElement(By.*xpath*("//div[@aria-label='Add Hobbies']"));

//Creating the JavascriptExecutor interface object by Type casting

JavascriptExecutor js = (JavascriptExecutor)driver;

//Perform Click on WebElement using JavascriptExecutor

js.executeScript("arguments[0].click();", hobbies);

Thread.*sleep*(5000);

**for**(**int** i=2;i<=4;i++)

{

String path = "(//div[@aria-pressed='false'] [@role='button'] [@tabindex='0'])"+"["+i+"]";

hobbies.findElement(By.*xpath*(path)).click();

}

hobbies.findElement(By.*xpath*("(//div[@aria-label='Save'] [@role='button'] [@tabindex='0'])")).click();

System.***out***.println("Hobbies added");

Thread.*sleep*(1000);

JavascriptExecutor js3 = (JavascriptExecutor) driver;

js3.executeScript("window.scrollBy(0,-550)", "");

//close the window

WebElement close = driver.findElement(By.*xpath*("(//div[@aria-label='Close'] [@role='button'] [@tabindex='0'])"));

JavascriptExecutor js5= (JavascriptExecutor)driver;

js5.executeScript("arguments[0].click();", close);

Thread.*sleep*(3000);

JavascriptExecutor js4 = (JavascriptExecutor) driver;

js4.executeScript("window.scrollBy(0,-550)", "");

}

**catch** (Exception e) {

System.***out***.println("Test5 failed");

System.***out***.println(e);

}

}

@Test

**public** **void** Test6() **throws** InterruptedException{

Thread.*sleep*(5000);

WebElement mydetails = driver.findElement(By.*xpath*("(//div[@aria-label='logout']"));

mydetails.click();

Thread.*sleep*(1000);

WebElement logout = mydetails.findElement(By.*xpath*("(//div[@role='button'] [@tabindex='0'])[14]"));

}

@Test

**public** **void** Test7() **throws** InterruptedException{

**try** {

Thread.*sleep*(5000);

driver.findElement(By.*xpath*("(//a[@role='tab' and

@tabindex='0'])[1]")).click();

Thread.*sleep*(5000);

WebElement mydetails = driver.findElement(By.*xpath*("(//div[@aria-label='Your profile' and @role='button' and @tabindex='0'])"));

((JavascriptExecutor)driver).executeScript("arguments[0].removeAttribute( )", mydetails);

mydetails.click();

Thread.*sleep*(2000);

WebElement logout = mydetails.findElement(By.*xpath*("(//div[@role='button'] [@tabindex='0'])[9]"));

logout.click();

System.***out***.println("Logged out successfully");

}

**catch** (Exception e)

{

System.***out***.println("Test7 failed");

System.***out***.println(e);

}

}

@Test

**public** **void** Test8() **throws** Exception {

Thread.*sleep*(1000);

WebElement email = driver.findElement(By.*id*("email"));

email.sendKeys("randomstproject123");

WebElement password = driver.findElement(By.*id*("pass"));

password.sendKeys("testinglogin123");

WebElement login = driver.findElement(By.*name*("login"));

login.click();

String opurl= driver.getCurrentUrl();

Thread.*sleep*(2000);

**if**(opurl!="https://www.facebook.com/?sk=welcome")

**throw** **new** Exception("Login unsuccessful because of invalid email/password ");

}

@AfterTest

**public** **void** Test9() {

driver.quit();

}

}

**CHAPTER 4**

**TESTING**

Testing is vital for the success of any software. No system design is ever perfect. Testing is carried out in two phases. First phase is during the software engineering that is during the module creation. Second phase is after the completion of software. This is system testing which verifies that the whole set of programs hanged together. Software Testing is the process used to help identify the correctness, completeness, security, and quality of developed computer software.

**Table 4.1** Testing validations

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST CASE** | **INPUT** | **OUTPUT / BEHAVIOUR** | **REMARKS** |
| 1 | Open the Project executable file | The project launches the chrome webpage of the specified URL | Respective webpage is launched.  TEST CASE PASSED |
| 2 | Valid inputs provided for source, destination and appropriate date in calendar. | Chrome automates the webpage and loads the inputs appropriately | TEST CASE PASSED |
| 3 | Clicking on the profile name | Retrieving the name of the account opened for verification purpose | TEST CASE PASSED |
| 4 | Clicking on the profile button and changing the profile picture | Profile gets changed successfully | TEST CASE PASSED |
| 5 | Providing hobby details by selecting with different predefined list of values | Hobbies gets added successfully | TEST CASE PASSED |
| 6 | A different destination is loaded without changing the locator x-path of the web element. | The console shows the test failure method with exceptions in connecting with path loaded in the code. | TEST CASE FAILED |

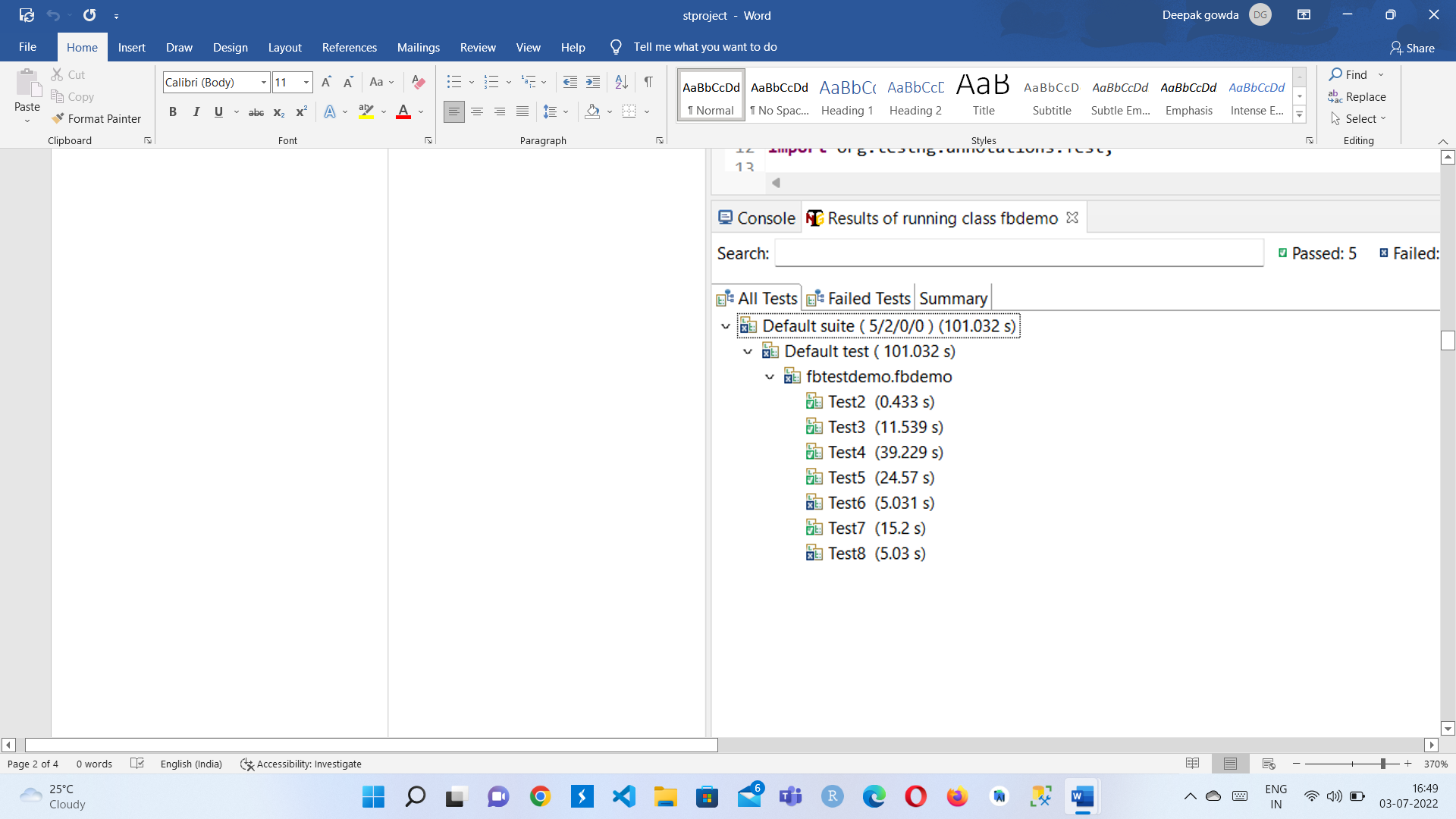
|  |  |  |  |
| --- | --- | --- | --- |
| 7 | Selecting the log out button from the profile setting menu | The correct webpage is displayed after execution and the result is recorded positive. | TEST CASE PASSED |
| 8 | A different destination is loaded without changing the locator x-path of the web element. | The correct webpage is displayed after execution with appropriate error messages. | TEST CASE FAILED |
| 9 | Close the webpage by calling the quit function of the webdriver | Webpage is closed without any disruption in the connection. | TEST CASE PASSED |

# TEST REPORT

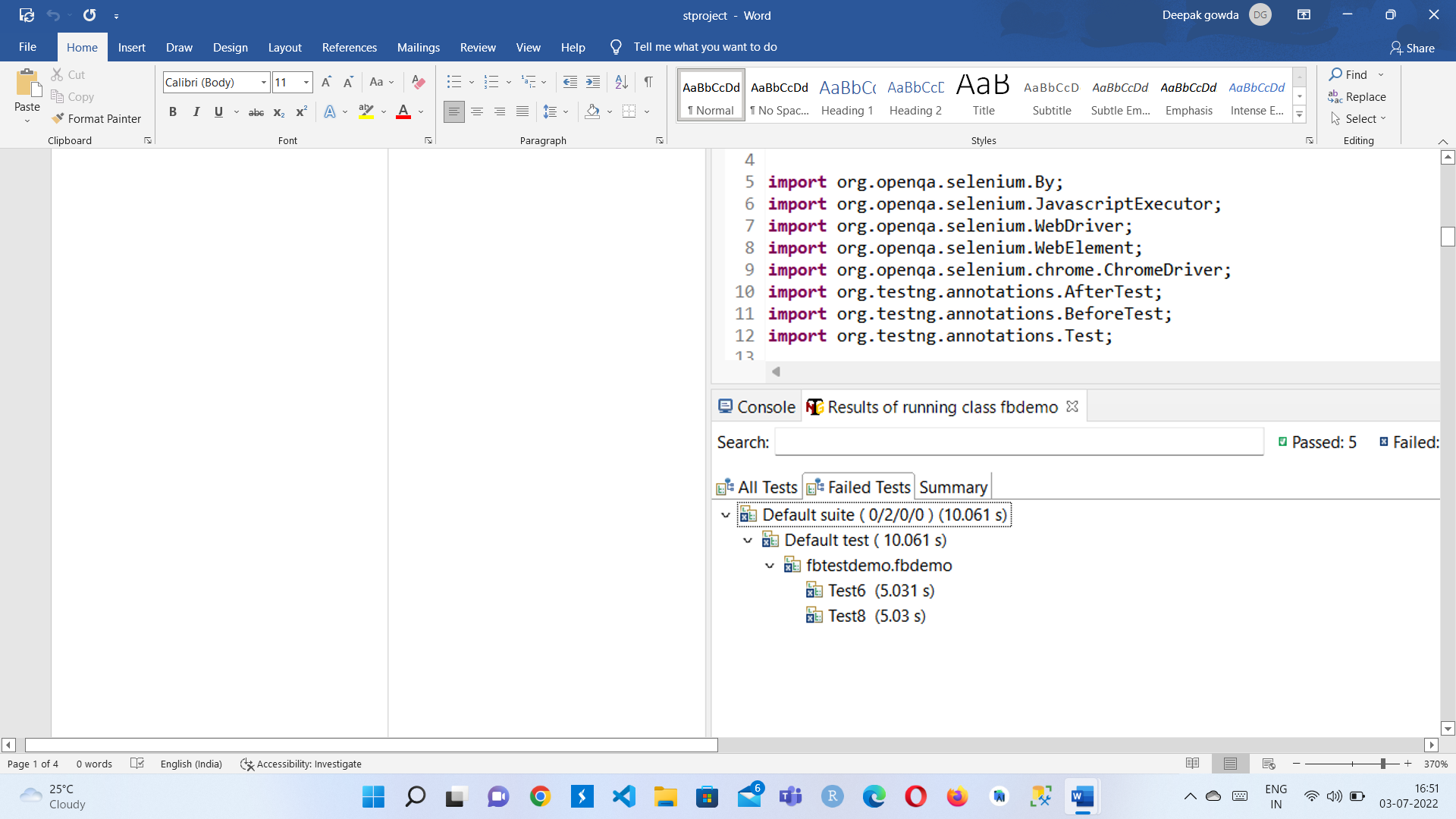
**Number of TC’s executed: 9 Number of TC’s passed: 7 Number of TC’s failed: 2 Number of defects raised: 0**

# TESTNG PLUG - IN FOR IDE

TestNG is an automation testing framework in which NG stands for "Next Generation". Suppose, you have five test cases, one method is written for each test case (Assume that the program is written using the main method without using testNG). This is not possible without using TestNG. Annotations used in the testing are very easy to understand ex: @BeforeMethod, @AfterMethod, @BeforeTest, @AfterTest. WebDriver has no native mechanism for generating reports.



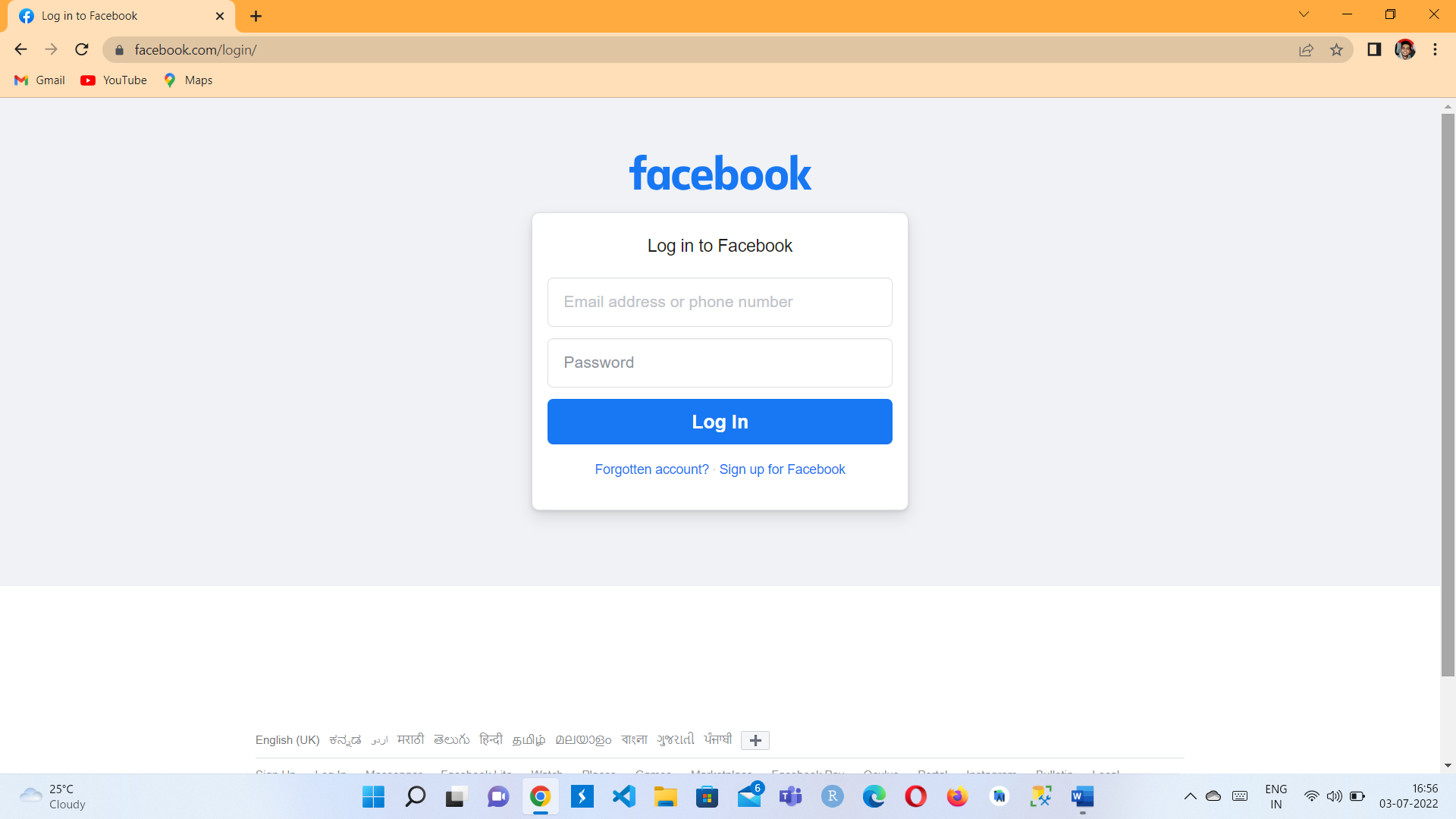
**Figure 4.2.1 TestNG all test cases**



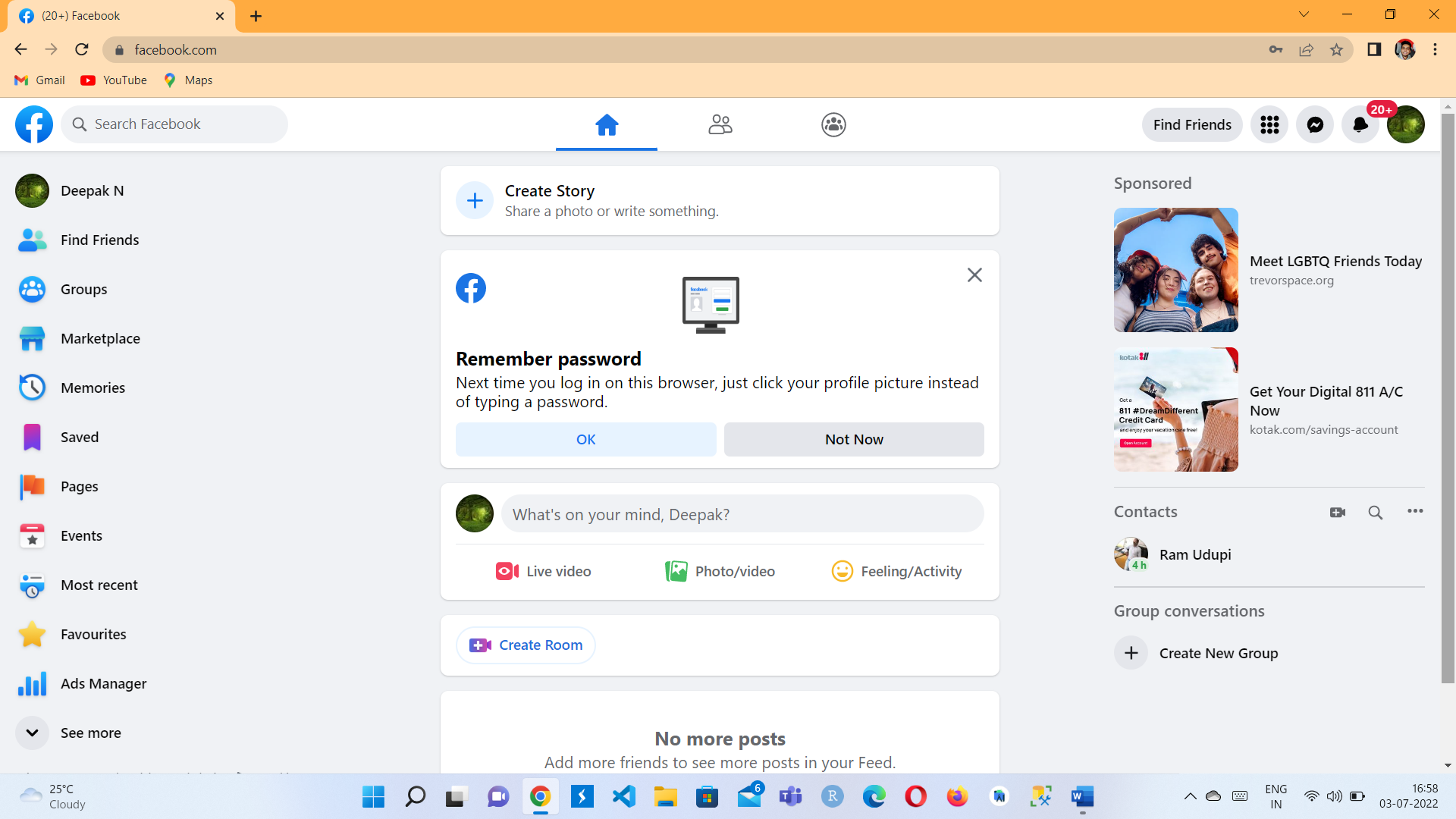
**Figure 4.2.2 TestNG failed test cases**

# 4.3 SNAPSHOTS

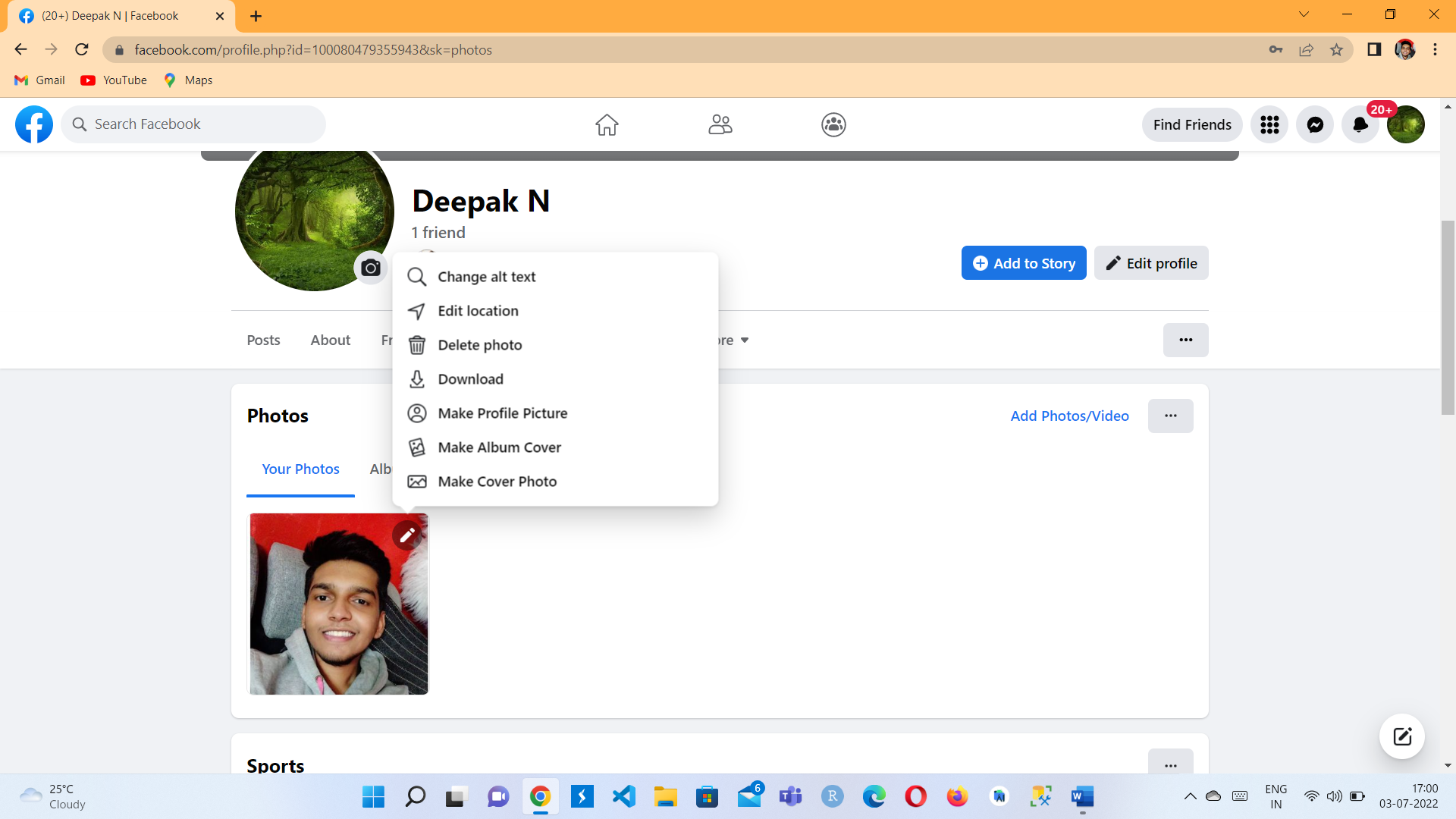
The regular output for a webpage(Facebook) is shown in the fig below-



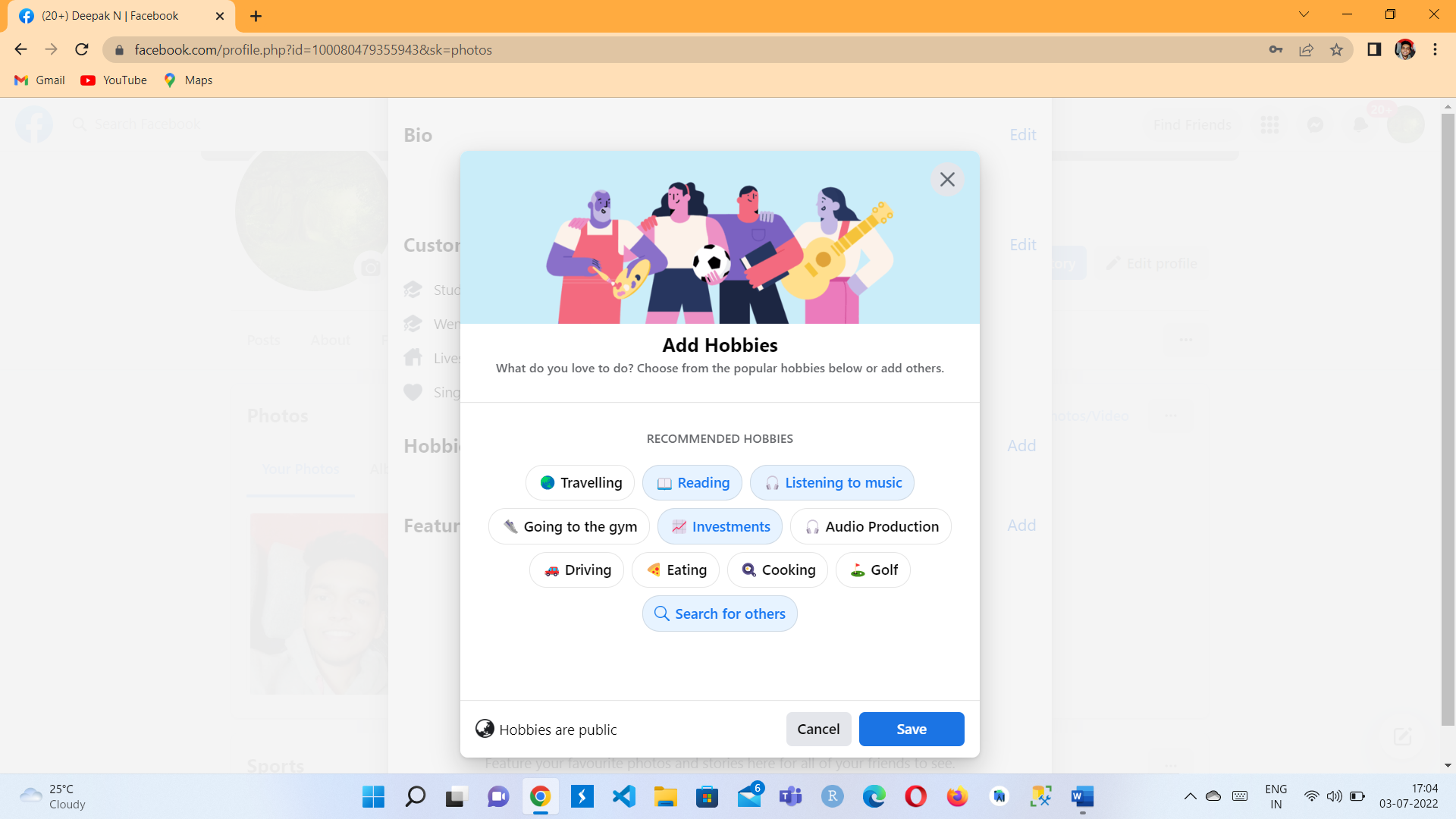
**Fig 4.3.1** Loading source



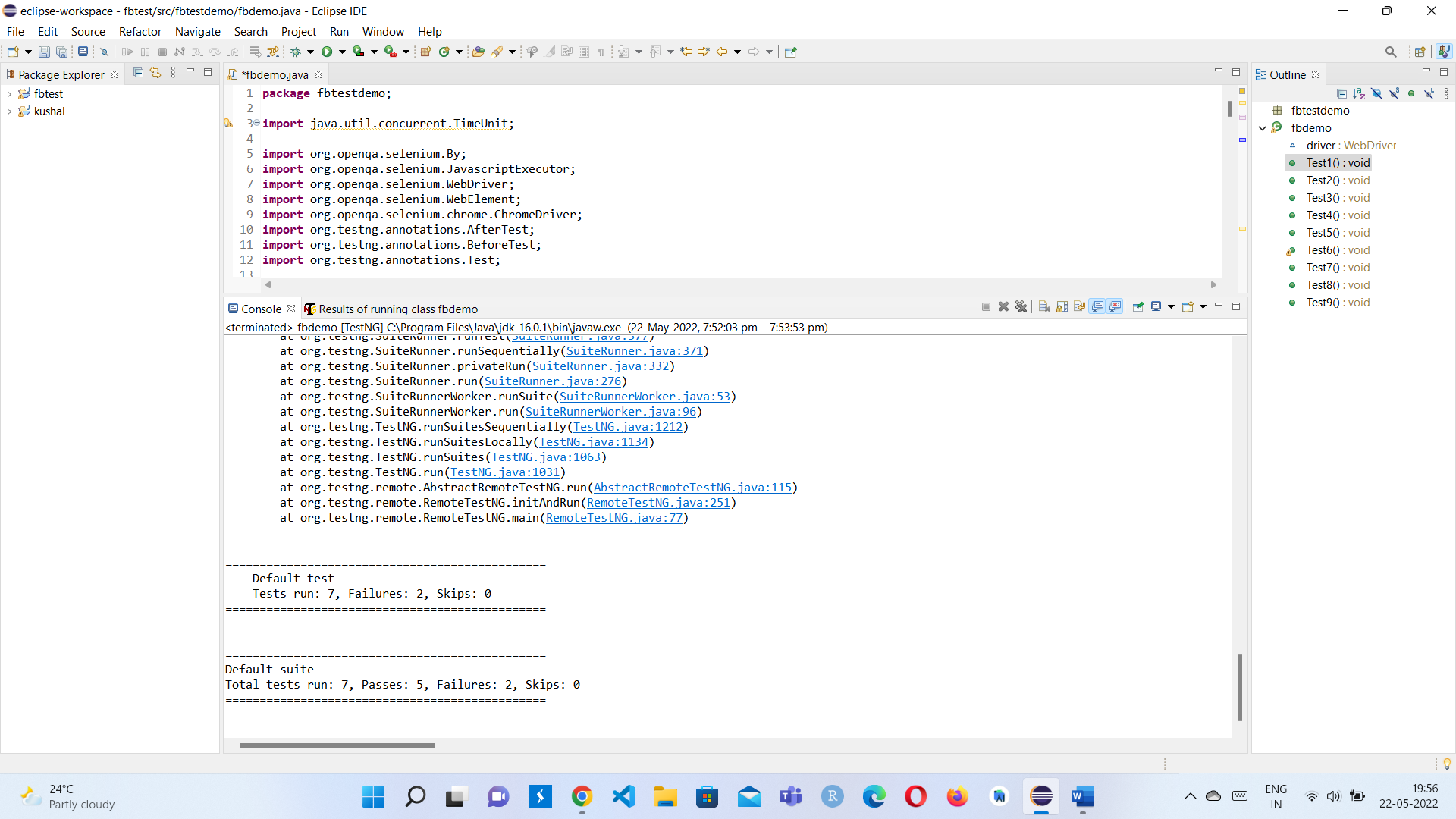
**Fig 4.3.2** Logging into the account



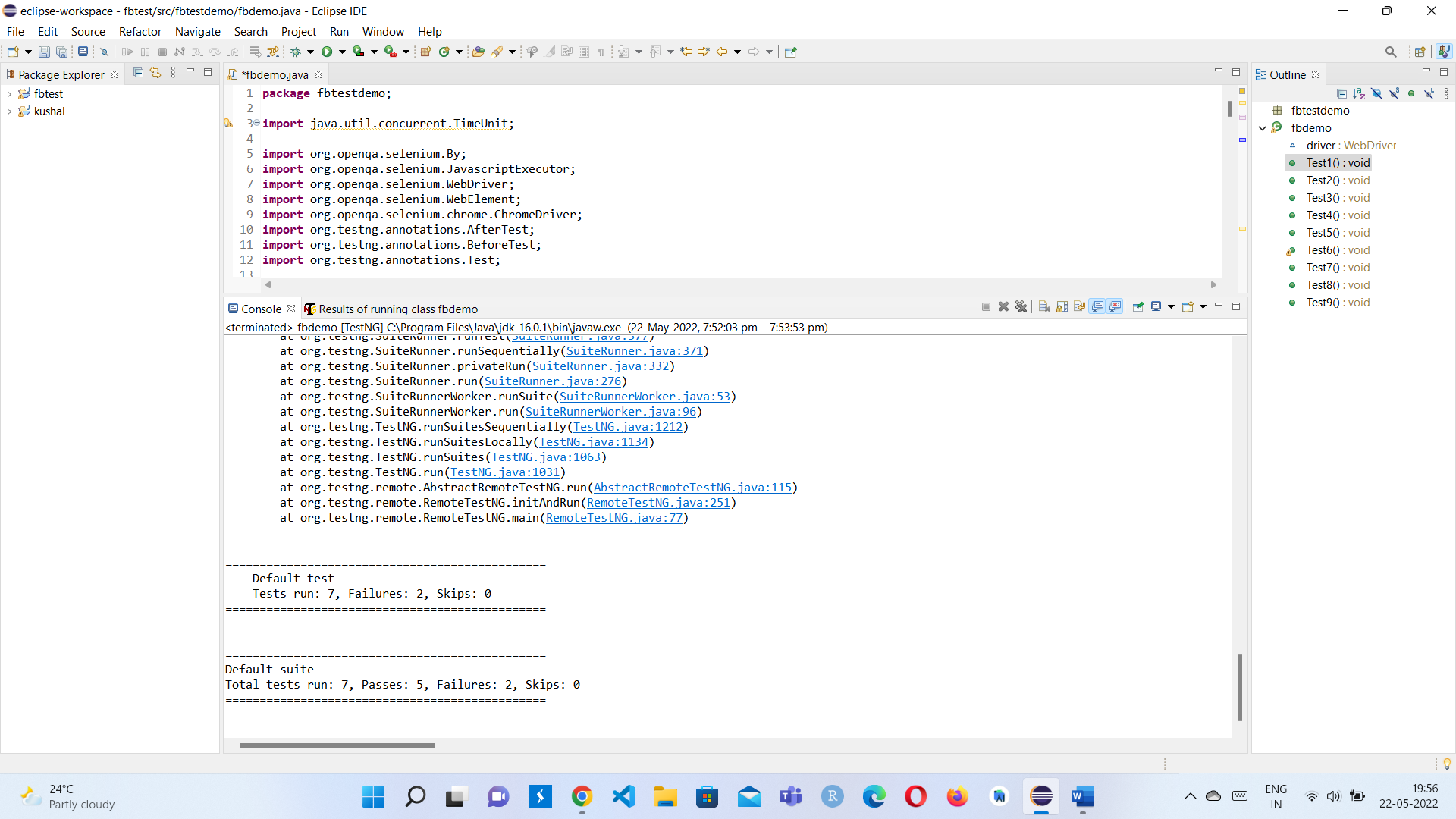
**Figure 4.3.3** Profile updating



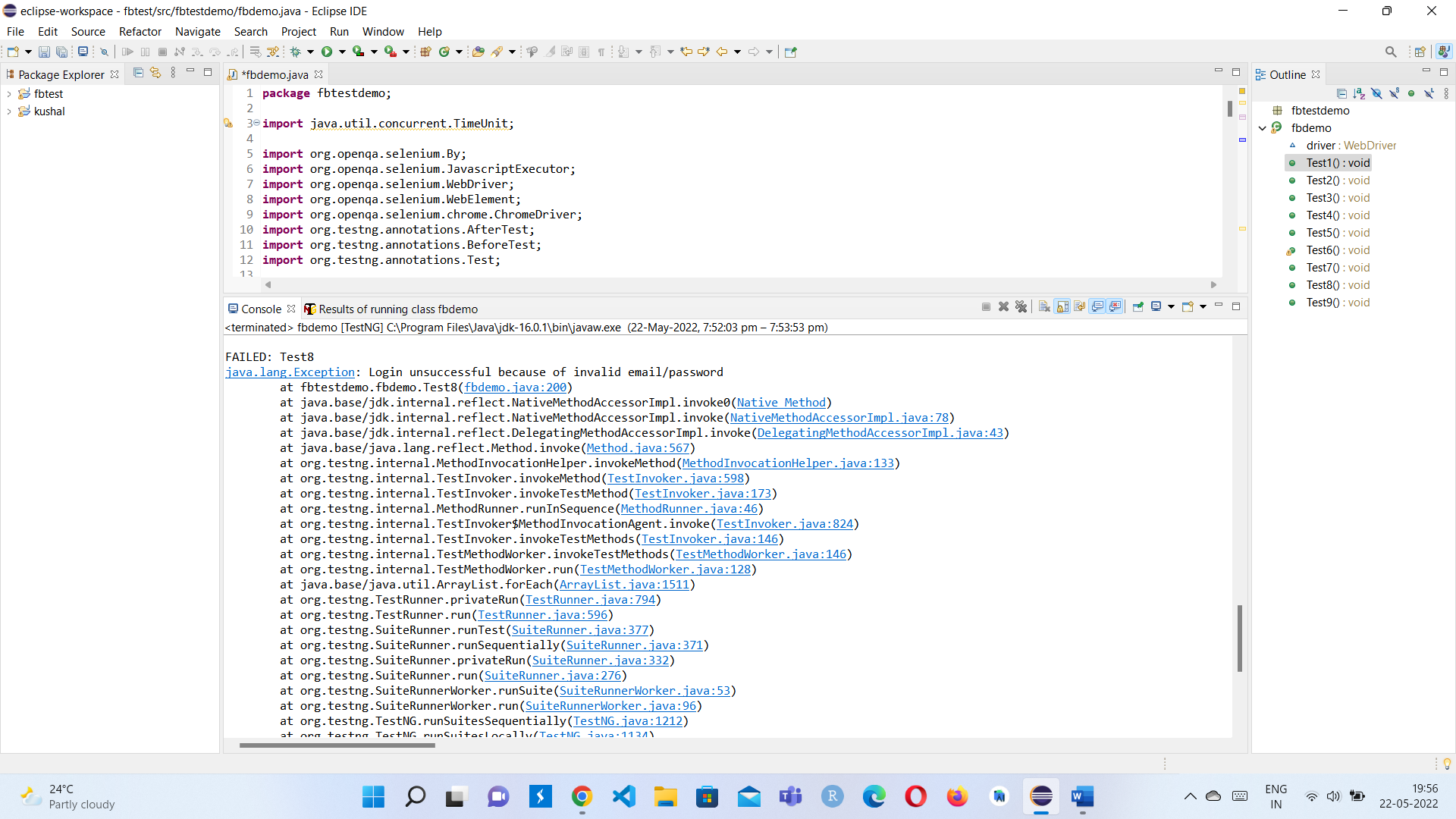
**Figure 4.3.4** After the editing profile is clicked, we are adding hobbies



**Figure 4.3.5** Test suite for Facebook automation



**Figure 4.3.6** TC6 where invalid x-path of the resultant webpage is entered into the code and hence the test case is failed



**Figure 4.3.7** Errors for TC8 where logging with wrong credentials is done, hence an error occurs in the code where it does not direct to the required webpage

**CHAPTER 5**

**CONCLUSION**

We can conclude by stating that the automation process is fast and appropriate process to enhance the testing ability of the code. We are able to find out the loopholes in the process and come to a conclusion about how many test cases can be satisfied using the testing tool. The more number of test cases solved helps out in clearing all the anomalies or exceptions where we get to know the root cause and be able to track it down. In this scenario we are able to test the efficiency of Facebook features which are used in the web pages and hence the issues can be sorted out easily, swiftly and correctly.