Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Spring, Year: 2025), B.Sc. in CSE (Day)

Automated Login Testing using Selenium WebDriver and JUnit in Java

---

Course Title: Software Testing and Quality Assurance Lab
Course Code: CSE 454
Section: 213-D1

Students Details

Name	ID
Rabiul Hasan	213902072
MD.Mahamudul Hasan	182002058

Submission Date: 07-03-2025
Course Teacher's Name: Md. Zahidul Hasan
[For teachers use only: Don't write anything inside this box]

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

0.1 Introduction

Automation testing is a crucial part of modern software development practices. It ensures reliability, saves time, and reduces human errors during repetitive testing. This lab demonstrates the automation of a login page using Selenium WebDriver with JUnit for unit testing. The selected application is a live web app named Bistro Boss. The project covers browser-based automation of the login feature and validates successful login using URL redirection.

0.2 Objective

The main objective of this lab project is to:

- Automate the login functionality of a web application using **Selenium WebDriver** in Java.
- Validate the success of login actions by checking redirection to the homepage.
- Implement unit testing using the **JUnit framework** to ensure structured, repeatable, and reliable test execution.
- Demonstrate the integration of browser automation with testing tools to simulate real-user interactions and verify expected outcomes.
- Enhance understanding of **test automation practices** in software development, focusing on quality assurance and efficiency.

0.3 Problem Domain

Manual testing of web applications, especially for routine tasks like login verification, is both time-consuming and error-prone. Inconsistent testing may lead to undetected bugs. The problem addressed here is the need for reliable, repeatable, and efficient testing of the login functionality in a web application.

By automating the login process using Selenium, the project aims to reduce manual effort and improve test accuracy. Additionally, incorporating unit tests using JUnit ensures that tests are structured, modular, and maintainable.

0.4 Selenium Automation

Selenium WebDriver is used in this project to automate browser-based actions that simulate a real user performing a login operation. The key steps performed by the automation script are:

- Launch the Google Chrome browser using the `ChromeDriver` class.
- Navigate to the login page of the Bistro Boss web application using `driver.get()`.
- Locate the email and password input fields using `By.name()` and enter valid credentials.
- Locate and click the login button using `By.cssSelector()`.
- Wait for a few seconds to ensure the login action is completed.
- Optionally, close the browser using `driver.quit()`.

The following code snippet illustrates the core Selenium implementation:

```
WebDriver driver = new ChromeDriver();
driver.get("https://bistro-boss-final-projec-55fef.web.app/login");

WebElement emailField = driver.findElement(By.name("email"));
emailField.sendKeys("tst90@gmail.com");

WebElement passwordField = driver.findElement(By.name("pass"));
passwordField.sendKeys("691156##Mr");

WebElement loginButton = driver.findElement(By.cssSelector
("input.btn.btn-primary"));
loginButton.click();

Thread.sleep(5000);
driver.quit();
```

Selenium provides a powerful API to interact with web elements and is widely used for functional and regression testing in web applications.

0.5 Unit Testing

To ensure structured and reliable automation testing, this project uses the **JUnit** framework. JUnit is a widely-used testing framework in Java that allows developers to write repeatable tests with clear setup, execution, and teardown phases.

The unit testing implementation in this project includes:

- `@BeforeEach` — Initializes the `WebDriver` and navigates to the login page before each test.

- `@Test` — Performs the actual login test by entering credentials, clicking the login button, and verifying the resulting URL.
- `@AfterEach` — Closes the browser session after each test to ensure no leftover processes affect subsequent tests.

Below is a key snippet of the JUnit test implementation:

```
@BeforeEach
public void setUp() {
    driver = new ChromeDriver();
    driver.get("https://bistro-boss-final-projec-55fef.web.app/login");
}

@Test
public void testLogin() {
    WebElement emailField = driver.findElement(By.name("email"));
    emailField.sendKeys("tst90@gmail.com");

    WebElement passwordField = driver.findElement(By.name("pass"));
    passwordField.sendKeys("691156##Mr");

    WebElement loginButton = driver.findElement(By.cssSelector("
    input.btn.btn-primary"));
    loginButton.click();

    Thread.sleep(3000);
    String currentUrl = driver.getCurrentUrl();
    Assertions.assertEquals("https://
    bistro-boss-final-projec-55fef.web.app/", currentUrl);
}

@AfterEach
public void tearDown() {
    driver.quit();
}
```

This test checks whether the user is successfully redirected to the homepage after logging in. If the URL matches the expected value, the test passes; otherwise, it fails. Assertions help verify correct application behavior automatically.

0.6 Implementation

Selenium Automation - Main Class

```
package com.mycompany.mavenproject1final;
```

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

/**
 *
 * @author DELL
 */
public class Mavenproject1final {

    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();
        driver.get("https://bistro-boss-final-projec-55fef.web.app/login");

        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        WebElement emailField = driver.findElement(By.name("email"));
        emailField.sendKeys("tst90@gmail.com");

        WebElement passwordField = driver.findElement(By.name("pass"));
        passwordField.sendKeys("691156##Mr");

        WebElement loginButton = driver.findElement(By.cssSelector
        ("input.btn.btn-primary"));
        loginButton.click();

        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        driver.quit();
    }
}
```

JUnit Test File

```
package com.mycompany.mavenproject1final;

import org.junit.jupiter.api.AfterEach;
```

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.junit.jupiter.api.Assertions;

public class Mavenproject1finalTest {

    private WebDriver driver;
    private final String BASE_URL =
        "https://bistro-boss-final-projec-55fef.web.app/login";

    @BeforeEach
    public void setUp() {
        try {
            System.out.println("STEP 1: Initializing ChromeDriver...");
            driver = new ChromeDriver();
            System.out.println("STEP 2: ChromeDriver initialized.");

            System.out.println("STEP 3: Navigating to URL...");
            driver.get(BASE_URL);
            System.out.println("STEP 4: Navigation successful.");

            Thread.sleep(2000);

        } catch (Exception e) {
            System.err.println("Exception in setUp: " + e.getMessage());
            e.printStackTrace();
            Assertions.fail("Failed in setUp(): " + e.getMessage());
        }
    }

    @Test
    public void testLogin() {
        try {
            System.out.println("STEP 5: Locating email field...");
            WebElement emailField = driver.findElement(By.name("email"));
            emailField.sendKeys("tst90@gmail.com");

            System.out.println("STEP 6: Locating password field...");
            WebElement passwordField = driver.findElement(By.name("pass"));
            passwordField.sendKeys("691156##Mr");

            System.out.println("STEP 7: Locating and clicking login button...");
            WebElement loginButton =
                driver.findElement(By.cssSelector("input.btn.btn-primary"));
```

```
            loginButton.click();

            System.out.println("STEP 8: Waiting for redirection...");
            Thread.sleep(3000);

            String currentUrl = driver.getCurrentUrl();
            Assertions.assertEquals(
                "https://bistro-boss-final-projec-55fef.web.app/", currentUrl);
            System.out.println("STEP 9: Login successful. Test passed.");

        } catch (Exception e) {
            e.printStackTrace();
            Assertions.fail("Test failed during login: " + e.getMessage());
        }
    }

    @AfterEach
    public void tearDown() {
        System.out.println("STEP 10: Tearing down WebDriver...");
        if (driver != null) {
            driver.quit();
            System.out.println("STEP 11: Driver quit successfully.");
        }
    }
}
```

0.7 OutPut

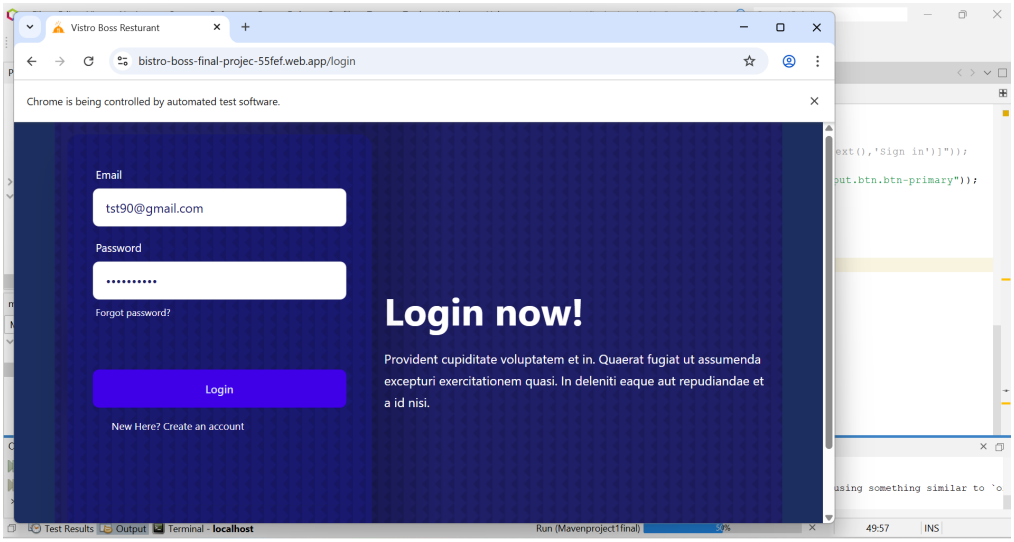

Figure 1: Selenium Automation

Figure 2: Testing Display

0.8 Conclusion

This project successfully demonstrates how Selenium WebDriver can be used to automate web application testing, and how JUnit can be integrated to perform structured unit testing. The login functionality of the Bistro Boss application was tested automatically, verifying that a user can log in and be redirected correctly. This practice significantly reduces the effort required for repetitive testing and increases accuracy.

0.9 Discussion

The automation testing project utilizing Selenium WebDriver and JUnit for the login functionality of the Bistro Boss web application demonstrates the effectiveness of modern testing tools in validating real-world web systems.

Selenium WebDriver was used to simulate user actions such as entering an email, typing a password, and clicking the login button. The use of hard-coded ‘Thread.sleep()‘ methods served for basic waiting purposes, though in a production environment, more robust solutions such as `WebDriverWait` should be used to handle dynamic content loading.

JUnit provided a structured way to define the test lifecycle, including setup and teardown processes using `@BeforeEach` and `@AfterEach`. Assertions were used to verify the redirection to the homepage after a successful login, ensuring the system behaved as expected.

Some potential limitations of the current implementation include:

- Hard-coded credentials that can pose security and maintenance issues.
- Lack of parameterization and reusable methods.
- Basic sleep-based waiting, which is less reliable than explicit waits.