Worldline Testing Task – 4

Dhineshwaran C
WDGET2024090

Github Link: https://github.com/Dhineshwaran-C/WorldlineTask4.git

Data Provider with Cucumber BDD

Introduction:

This document is about a basic automated test script using Selenium WebDriver and Cucumber for testing a login functionality of a web application. It includes Java classes to handle web interactions, read data from Excel files, and define feature scenarios using Cucumber's Gherkin syntax. The test scenario verifies the login process using multiple sets of credentials.

Code Explanation:

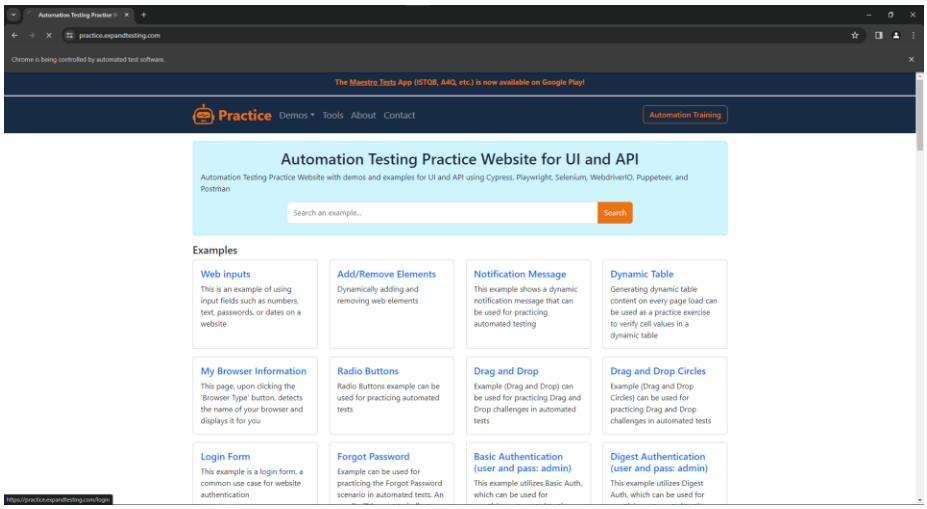
Logins class:

Open_chrome_and_open_application() Method:

This function serves as the entry point for the test script. It initializes the WebDriver, specifically for Chrome, by setting the system property to the ChromeDriver executable path. Then, it maximizes the browser window and navigates to the application URL. This step ensures that the test starts in a clean state, with the browser ready to

interact with the application under test. This action occurs when the feature file calls the 'Given User is on Home Page' step.

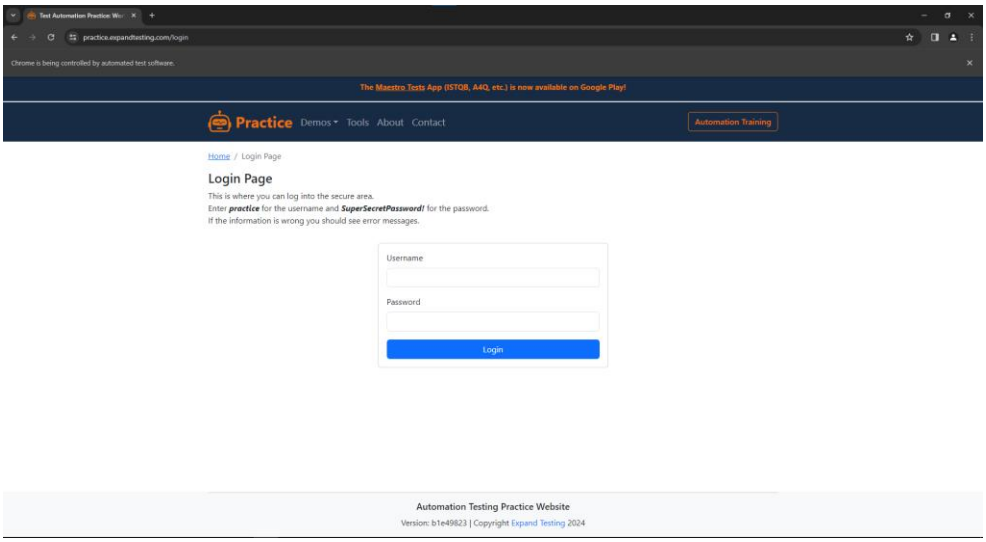
```
@Given("User is on Home Page")
public void open_chrome_and_open_application() {
    System.setProperty("webdriver.chrome.driver","G:\\worldline\\chromedriver-win64\\chromedriver-win64\\chromedriver.exe");
    driver = new ChromeDriver();
    driver.manage().window().maximize();
    driver.get("https://practice.expandtesting.com/");
    System.out.println("Open Home Page Executed");
}
```



Login_page() Method:

Upon reaching the application's home page, the script proceeds to navigate to the login page. It locates the login link element on the home page and clicks on it. Subsequently, it verifies if the URL matches the expected login page URL. This step ensures that the navigation to the login page is successful and that the correct page is loaded for further interactions. This action occurs when the feature file calls the ' When User navigate to Login Page' step.

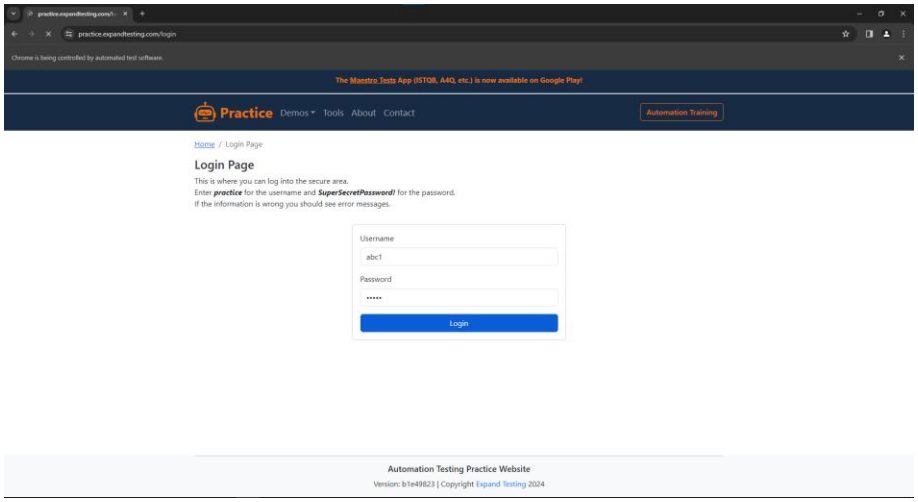
```java
@When("User navigate to Login Page")
public void login_page() {
    WebElement login = driver.findElement(By.xpath("(//a[normalize-space()='Login Form'])[1]"));
    login.click();
    String expectedURL = "https://practice.expandtesting.com/login";
    String actualURL = driver.getCurrentUrl();
    Assert.assertEquals(expectedURL,actualURL);
    System.out.println("Navigate to Login Page Executed");
}
```



Enter_username_and_password() Method:

This method facilitates the input of username and password into the login form. It retrieves the credentials from test data stored in an Excel file, using the provided user index extracted from the feature file. Upon retrieval, it inputs these credentials into the corresponding fields on the login form. This action occurs when the feature file calls the ' Then User enters username and password <userIndex>' step, with <userIndex> being taken from the examples provided at the bottom of the feature file.

```java
@Then("User enters username and password {int}")
public void enter_username_and_password(int userIndex) {
    testData = testingData();
    String usernames = (String) testData[userIndex - 1][0];
    String passwords = (String) testData[userIndex - 1][1];
    WebElement user = driver.findElement(By.name("username"));
    user.sendKeys(usernames);
    WebElement pass = driver.findElement(By.name("password"));
    pass.sendKeys(passwords);
    System.out.println("Enter Username and Password Executed");
}
```



Login_with_credentials() Method:

After entering the username and password, the script proceeds to click on the login button to initiate the login process. This action triggers the authentication process. This action occurs when the feature file calls the ' And User should get logged in' step.

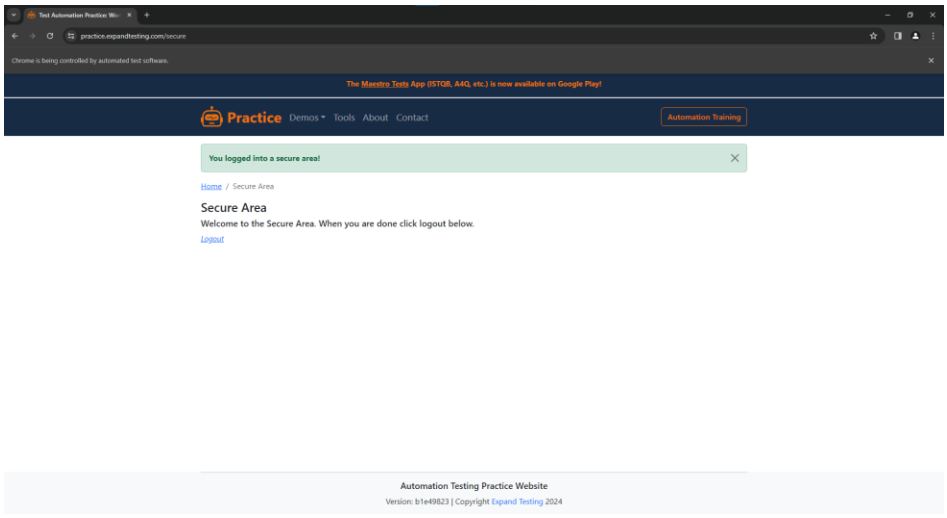
```java
@And("User should get logged in")
public void login_with_credentials()  {
    WebElement l = driver.findElement(By.xpath("(//button[normalize-space()='Login'])[1]"));
    l.click();
    System.out.println("User clicked the login button");
}
```

Login_success() Method:

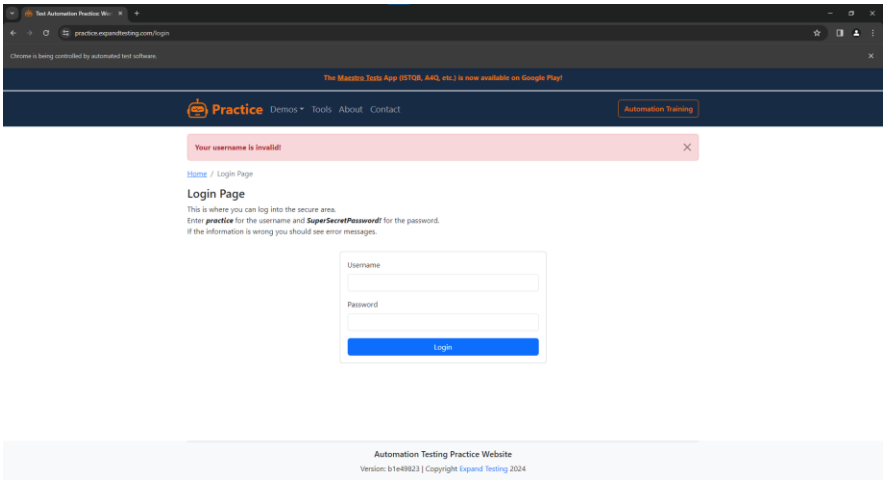
Following the attempt to log in, this method verifies if the login was successful. It does so by comparing the current URL with the expected URL of the page that authenticated users are redirected to upon successful login. If the URLs match, it indicates that the login was successful; otherwise, it concludes that the login attempt failed. This action occurs when the feature file calls the ' Then Message displayed Login Successfully or not' step.

```
@Then("Message displayed Login Successfully or not")
public void login_success() throws InterruptedException {
    Thread.sleep(2000);
    String expectedURL = "https://practice.expandtesting.com/secure";
    String actualURL = driver.getCurrentUrl();
    if (expectedURL.equals(actualURL)) {
          System.out.println("Login Successful");
       } else {
          System.out.println("Login Failed");
       }
    driver.quit();
}
```

Login Success



Login Fail



testingData() Method:

This function handles the retrieval of test data from an Excel file. It uses an instance of the ReadExcelFile class to access the Excel file and fetch the required data, such as usernames and passwords. By reading test data from an external source, this method enables data-driven testing, where multiple sets of credentials can be used to test the login functionality with different scenarios.

```
public Object[][] testingData(){
    ReadExcelFile config = new ReadExcelFile("G:\\worldline\\taskSample2.xlsx");

    int rows = config.getRowCount(0);
    int col = config.getColCount(0);

       Object[][] credentials = new Object[rows][col];

       for(int i=0;i<rows;i++) {
           for(int j=0;j<col;j++) {
               credentials[i][j] = config.getData(0, i+1, j);
           }
       }

       return credentials;
   }
```

ReadExcelFile class:

The constructor of the ReadExcelFile class initializes an instance of the XSSFWorkbook class, which represents the Excel workbook. It takes the path of the Excel file as a parameter. Inside the constructor, it attempts to create a FileInputStream from the Excel file path and initialize the XSSFWorkbook instance with it.

```
public class ReadExcelFile {
    XSSFWorkbook wb;
    XSSFSheet sheet;

    public ReadExcelFile(String excelPath) {
        try {
            File src = new File(excelPath);
            FileInputStream inp = new FileInputStream(src);
            wb = new XSSFWorkbook(inp);
        }
        catch(Exception e){
            System.out.println(e.getMessage());
        }
    }
```

getData() Method:

The getData() method retrieves data from a specific cell in the Excel sheet. It takes the sheet number, row number, and column number as parameters. It first gets the sheet from the workbook based on the provided sheet number. Then, it retrieves the row using the getRow() method and the column using the getCell() method. Finally, it extracts the string value of the cell using getStringCellValue() and returns it.

```
public String getData(int sheetnumber, int row, int column) {
    sheet = wb.getSheetAt(sheetnumber);
    String data = sheet.getRow(row).getCell(column).getStringCellValue();
    return data;
}
```

getRowCount() Method:

The getRowCount() method retrieves the total number of rows in a specified sheet. It takes the sheet number as a parameter. It uses the getLastRowNum() method to get the index of the last row containing data. Since row indexes are zero-based, adding 1 to the last row index gives the total number of rows.

```
public int getRowCount(int sheetnumber) {
    int rows = wb.getSheetAt(sheetnumber).getLastRowNum();
    return rows;
}
```

getColCount() Method:

The getColCount() method retrieves the total number of columns in a specified sheet. Similar to getRowCount(), it takes the sheet number as a parameter. It gets the first row of the sheet using getRow(0) to determine the column count. It iterates through each cell in the first row and increments the count if the cell is not blank. Finally, it returns the total number of non-blank cells, which represents the number of columns.

```
public int getColCount(int sheetnumber) {
    Row row = wb.getSheetAt(sheetnumber).getRow(0);
    int columns = 0;

    if (row != null) {
        int physicalNumberOfCells = row.getPhysicalNumberOfCells();
        for (int i = 0; i < physicalNumberOfCells; i++) {
            Cell cell = row.getCell(i);
            if (cell != null && cell.getCellType() != CellType.BLANK) {
                columns++;
            }
        }
    }
    return columns;
}
```

Feature File:

```
Feature: Login Action
    Description: This feature will test a LogIn and LogOut functionality

    Scenario: Login with valid Credentials
        Given User is on Home Page
        When User navigate to Login Page
        Then User enters username and password <userIndex>
        And User should get logged in
        Then Message displayed Login Successfully or not

    Examples:
        | userIndex |
        | 1 |
        | 2 |
        | 3 |
        | 4 |
        | 5 |
```

Test Data:

username	password
practice	SuperSecretPassword!
abc1	dfsd2
world	line
testing	training
task	Cucumber

Console Output:

```
Mar 24, 2024 7:55:23 PM cucumber.api.cli.Main run
WARNING: You are using deprecated Main class. Please use io.cucumber.core.cli.Main

Scenario: Login with valid Credentials            # src/test/java/feature/task4Feature.feature:13
Starting ChromeDriver 122.0.6261.111 (9d4c1072da62b411b351a38b9ed6214ab236aa7b-refs/branch-heads/6261@{#1015}) on port 36315
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.
ChromeDriver was started successfully.
[1711290325.090][WARNING]: This version of ChromeDriver has not been tested with Chrome version 123.
Mar 24, 2024 7:55:25 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: W3C
Open Home Page Executed
  Given User is on Home Page                      # feature.logins.open_chrome_and_open_application()
Navigate to Login Page Executed
  When User navigate to Login Page                # feature.logins.login_page()
ERROR StatusLogger Log4j2 could not find a logging implementation. Please add log4j-core to the classpath. Using SimpleLogger to log to the console...
Enter Username and Password Executed
  Then User enters username and password 1        # feature.logins.enter_username_and_password(int)
User clicked the login button
  And User should get logged in                   # feature.logins.login_with_credentials()
Login Successful
  Then Message displayed Login Successfully or not # feature.logins.login_success()

Scenario: Login with valid Credentials            # src/test/java/feature/task4Feature.feature:14
Starting ChromeDriver 122.0.6261.111 (9d4c1072da62b411b351a38b9ed6214ab236aa7b-refs/branch-heads/6261@{#1015}) on port 37603
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.
ChromeDriver was started successfully.
[1711290332.310][WARNING]: This version of ChromeDriver has not been tested with Chrome version 123.
Mar 24, 2024 7:55:32 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: W3C
Open Home Page Executed
  Given User is on Home Page                      # feature.logins.open_chrome_and_open_application()
Navigate to Login Page Executed
  When User navigate to Login Page                # feature.logins.login_page()
Enter Username and Password Executed
  Then User enters username and password 2        # feature.logins.enter_username_and_password(int)
User clicked the login button
  And User should get logged in                   # feature.logins.login_with_credentials()
Login Failed
  Then Message displayed Login Successfully or not # feature.logins.login_success()

2 Scenarios (2 passed)
10 Steps (10 passed)
0m14.285s
```