**NAME : ANUSUYA .B**

**COURSE : ADVANCED CERTIFICATION PROGRAM IN**

**SOFTWARE TESTING**

**ENROLLMENT**

**NUMBER : EBEONO223747633**

**BATCH : 2022-8955**

**TRAINER : LEENA THORAT**

**WEBSITE LINK :**

**https://www.lambdatest.com/selenium-playground/**

CONTENT

1. Introduction to Website
2. System Specifications
3. Software Specification
4. Hardware Specification
5. Tested Features
6. Setting CromeDriver Path
7. Window Maximize
8. URL Access
9. Getting Title
10. Logo Verification
11. Hyperlinks Count
12. URL validation
13. Send keys & Validation
14. Click And Hold & Validation
15. Popup & Validation
16. Drop Down & Validation
17. By Visible Text
18. By Value
19. Multiple Selection
20. Drop Down Count
21. Check Box
22. Conclusion
23. **INTRODUCTION TO WEBSITE**

LambdaTest is a cloud-based testing platform that offers a

range of services for testing web applications, includes Cross-Browser Testing, Selenium Grid, Real-time Testing, Test Automation, Visual Testing, Issue Tracker Integration, Collaboration, and Sharing. It provides a seamless and efficient solution for ensuring the compatibility and functionality of web applications across various browsers and platforms. With features such as parallel test execution, real-time interaction, automated testing, visual comparison, and integration with issue tracking systems, LambdaTest empowers teams to streamline their testing processes and deliver high-quality web applications.

,

**B. SYSTEM SPECIFICATIONS**

1. **HARDWARE CONFIGURATION**

* System : Toshiba
* Hard Disk : 40
* Monitor : 14’ Colour Monitor
* Mouse : Optical Mouse
* RAM : 4GB
* Keyboard : 101 Keys

1. **SOFTWARE SPECIFICATION**

* Operating System : Windows 10
* Coding Language : Java
* Automation Technology : Selenium WebDriver
* Tools : Eclipse IDE
* Build-in Tool : Maven

**C. TESTED FEATURES**

1. **SETTING CROMDRIVER PATH**

The **System.setProperty()** method is used to set the value of a specific system property. The path specified should point to the location where the ChromeDriver executable file is stored on your machine.

1. **WINDOW MAXIMIZE**

The line of code **driver.manage().window().maximize()** is used to maximize the browser window when working with Selenium WebDriver.

This code is called on the **WebDriver** instance to manage the window of the browser being automated. The **manage()** method returns the **Options** interface, which provides various options to customize the browser window. In this case, the **window()** method is called to access the window-related options.

1. **URL ACCESS**

The **get()** method is called on the WebDriver instance to initiate the navigation to the provided URL. The **get()** is used to instruct the WebDriver to navigate to a specific URL, allowing you to interact with and automate actions on the corresponding web page.

1. **GETTING TITLE**

The method getTitle() retrieves the title of the

current web page, and **System.out.println("Page Title : "+title);** prints the title to the console.

1. **LOGO VERIFICATION**

This code block uses an **if-else** statement to check if

The **Logo** element displayed on the webpage**.** The **isDisplayed()** method is called on the Logo element, which returns a boolean value indicating whether the element is currently visible on the page. If the logo is displayed, the code prints "Logo is presented" to the console.

Otherwise, if the logo is not displayed, it prints "Logo is NOT presented".

1. **HYPERLINKS COUNT**

In Java, **List** is an interface that represents an ordered collection of elements. And finds all the web elements on the webpage that are represented by the **<a> tag**, which typically corresponds to links. The code **link.size()** returns the number of elements found, indicating the total number of links on the webpage.

1. **URL VALIDATION**

The **getCurrentUrl()** method is called on the driver

object to retrieve the current URL of the web page. The obtained URL is stored in the variable **currenturl**. The code then uses an if statement to compare the **currenturl** with the **expectedurl** using the **equals()** method. This checks if the current URL matches the expected URL.

If the comparison evaluates to true, meaning the current URL matches the expected URL, the message **"validation passed"** is printed to the console.

If the comparison evaluates to false, indicating a mismatch between the current URL and the expected URL, the message **"validation failed"** is printed.

1. **SEND KEYS & VALIDATION**

The **sendKeys()** method is called on the element located by the **ID** "user-message" to enter the **givenvalue** into the input field.

The **click()** method is called on the element located by the **ID** "showInput" to simulate a button click.

The code then retrieves the text value from the element located by the **ID** "message" using the **getText()** method and stores it in the variable gettingvalue.

An if statement is used to compare **givenvalue** with **gettingvalue** using the **equals()** method to check if they are the same.

If the comparison evaluates to true, meaning the given and obtained input values match, the message "Value matched" is printed to the console.

If the comparison evaluates to false, indicating a mismatch between the given and obtained input values, the message "value non-matching" is printed.

1. **CLICK AND HOLD & VALIDATION**

An **Actions** object named action is created using the **Actions(driver)** constructor to perform advanced user interactions. The code locates the slider element using the **XPath**.

The clickAndHold() method of the action object is called with the slider element as the parameter to simulate clicking and holding the slider.

The **moveByOffset()** method of the action object is called to move the slider by a specified offset. In this case, it moves the slider 202 pixels horizontally and 0 pixels vertically.

The **release()** method of the action object is called to release the slider.

The **build().perform()** method is called on the action object to execute the sequence of actions.

The code finds the element with the ID "rangeSuccess" and assigns it to the value variable.

The text value of the value element is retrieved using the **getText()** method and stored in the **rangeText** variable.

An if statement is used to check if the **rangeText** is equal to "95", indicating that the slider validation is successful.

If the condition is true, the message "Slider validation successful" is printed to the console.

If the condition is false, the message "Slider validation failed" is printed to the console.

1. **POPUP & VALIDATION**

It sets the implicit wait timeout using the **implicitlyWait(Duration.ofSeconds(10))** method of the **driver.manage().timeouts()** object**.** This means that the driver will wait for a maximum of 10 seconds when searching for an element before throwing an exception if the element is not found.

The code finds the element with the name "**name**", and assigns it to the **first\_name** variable.

The **getAttribute("validationMessage")** method is called on the first\_name element to retrieve the value of the "validationMessage" attribute. This attribute typically contains the validation message associated with the input field.

The expected validation message "Please fill out this field." is assigned to the **expectingText** variable.

The **Assert.assertEquals(pleasefilloutthisfield, expectingText)** statement is used to compare the actual validation message (**pleasefilloutthisfield**) with the expected validation message (**expectingText**).

If the validation messages match, the message "IF USER PUT SUBMIT WITHOUT FILLING THE FIELD:] [validationMessage " is printed to the console, indicating that the validation passed.

If the validation messages do not match, the assertion will fail, and an assertion error will be thrown.

This code segment sets an implicit wait timeout, finds an input field element, retrieves its validation message, compares it with an expected message, and prints the validation message to the console. It can be used to verify if a validation message is correctly displayed when a required field is not filled out before submitting a form.

1. **DROP DOWN**

1. **BY VISIBLE TEXT**

It selects an option from a dropdown menu

using the **selectByVisibleText(** ) method of the dropdown object.

This method selects the option that matches the visible text "India" in the dropdown.

The selected option's text is then retrieved using the **getFirstSelectedOption().getText()** method of the dropdown object. This method returns the visible text of the currently selected option.

The retrieved text is assigned to the **visibleText** variable. Finally, the selected option's text is printed to the console.

1. **BY VALUE**

It selects an option from a dropdown menu using **the selectByValue()** method of the dropdown object. This method selects the option that has the attribute value set to "IN".

The selected option's text is then retrieved using the **getFirstSelectedOption().getText**() method of the dropdown object. This method returns the visible text of the currently selected option.

The retrieved text is assigned to the **Byvalue** variable. Finally, the selected option's text is printed to the console.

1. **MULTIPLE SELECTION**

The code checks whether the dropdown element

allows multiple selections or not. It uses the **isMultiple()** method of the dropdown object, which returns a boolean value indicating whether multiple selections are allowed.

If the dropdown allows multiple selections, the code prints "DropDown element allows multiple selections at a time".

If the dropdown does not allow multiple

selections, the code prints “DropDown element does NOT allow multiple selections at a time”.

1. **DROP DOWN COUNT**

It retrieves all the options within a dropdown menu

Using the **getOption()** method of the dropdown object. This method returns a list of **WebElement** objects representing each option in the dropdown.

The size of the list, which represents the number of options in the dropdown, is obtained using the **size()** method and assigned to the **listcoun**t variable.

Finally, the number of countries in the dropdown is printed to the console.

1. **CHECK BOX**

It locates a checkbox element using its **ID** "isAgeSelected**".** It enters a **loop** that iterates twice, from 0 to 1

It prints whether the checkbox is checked or not by using **System.out.println("The CheckBox is Checked " + checkbox.isSelected()).**

1. **CONCLUSION**

The LambdaTest code demonstrates the usage of

Selenium WebDriver to automate testing scenarios on the Lambda Test website. By following the providing instructions and understanding the code structure, users can execute the code and observe the results. This code can serve as reference for implementing automated testing using Selenium WebDriver in Java projects.