Introduction to Web Automation

It's time to get hands-on with Automation!

To do so, we present you with a brief introduction to automating website tests and Selenium, the tool you will be using to do this. Let's take a look!

https://youtu.be/8QeTh_7L4O4

Sounds awesome, right? Try this for yourself now:

Practice

In this practice, we are going to create our first automation framework using Selenium. To achieve this, start by creating a maven java project including the latest version of Selenium Dependency.

Follow this tutorial to install Selenium: **https://www.selenium.dev/documentation/webdriver/getting_started/install_library/**.

Make sure to also have Google Chrome installed and to download the respective ChromeDriver for your Chrome version, you can use the following link: **https://www.selenium.dev/documentation/webdriver/getting_started/install_drivers/**

Now we are almost ready to start!

Check the following link to get familiarized with basic Selenium actions. Create in your project an Example class and implement the code displayed in this article. **https://www.selenium.dev/documentation/webdriver/getting_started/first_script/**

After the first example is correctly implemented, we are going to create our own Selenium scripts.

To do this, create a new class with a public static void main method. Use the Selenium methods in the previous example to implement the following actions on the main method:

Navigate to the webpage (**https://wikipedia.org**)

1. Navigate to the webpage (**https://wikipedia.org**)
2. Write in the search box the text "Microsoft"
3. Click on the search button aside the search box
4. Wait for the new webpage to load
5. Get the title of the new webpage and check if it matches with "Microsoft", print a message in the console indicating whether it matches or not.
6. Get all the anchor links on the web page (hint: find by HTML tag "a"), and for each one of them, print on the console its respective text.

Note: For steps b, c, d, e, you can find the elements using the selenium Class By and its method CSS. To get the CSS locator of an element on a web page, right click on it and click inspect, then, the HTML code of the web page will be displayed. Right-click on the highlighted element and select copy > selector (only with chrome)

Make sure to execute the main class and make the required adjustments for the script to properly work.

Web Locators and Web Elements

How can you locate elements on a web page?

To automate tests, the first thing you need to know is how to identify the elements you want to be part of your scripts.

Learn how you can locate them by taking a look at this video:

https://youtu.be/0_uNYCYlKcQ

Practice

Part I

Enter this link **https://flukeout.github.io/** and check the graphical examples and the HTML code. Write the required selectors to locate only the required items. Make sure to complete all 32 levels.

Part II

Identify and write on a document all CSS selectors required to execute the following scenarios on **https://demoblaze.com/** website:

- Successful Sign Up
- Navigating to About Us Section and playing the video there
- Selecting and completing the buying process of any laptop on the website.

Make sure to group on the document the respective selectors according to the page or view they are located on. The CSS must be as efficient and maintainable as possible.

Driver Actions and Waits

Now... what if you need to emulate a waiting period when you automate? Well, that's when the driver's actions and awaits come in hand!

https://youtu.be/Q52dXvdNx2I

Practice

You are tasked with automating the login functionality of the **https://www.saucedemo.com/v1/** web application.

The login process involves entering a username and password, and then clicking on the login button. However, there might be delays in the rendering of the login page

or in the loading of the elements, so you need to implement appropriate wait strategies to handle these scenarios.

Start by creating a new Java Selenium project, and navigate to the web app URL. Identify and declare as WebElements all required elements to correctly execute the login process. Once you have the project configured and running, implement the following waiting strategies on different tests.

a. Implicit Wait:

- Set an implicit wait for 10 seconds using the WebDriver instance.
- Find the necessary elements using the locators and perform actions like sending keys to the username and password fields.
- Click on the login button.
- Observe how the implicit wait handles the delay between element searches and actions.

b. Explicit Wait with Expected Conditions:

- Set an explicit wait using the WebDriverWait class.
- Implement a wait condition for the visibility of the username input field.
- Once the element is visible, send keys to the username field.
- Implement a wait condition for the visibility of the password input field.
- Once the element is visible, send keys to the password field.
- Implement a wait condition for the element to be clickable for the login button.
- Once the button is clickable, click on it.
- Observe how the explicit wait waits for the specific conditions before performing actions.

c. Fluent Wait:

- Implement a fluent wait using the Wait and FluentWait classes.
- Customize the wait conditions and polling interval according to the web page behavior..
- Use fluent wait to wait for the presence of the username input field.
- Once the element is present, send keys to the username field.
- Use fluent wait to wait for the presence of the password input field.
- Once the element is present, send keys to the password field.
- Use fluent wait to wait for the element to be clickable for the login button.
- Once the button is clickable, click on it.
- Observe how fluent wait provides flexibility in setting up wait conditions.

On all the tests defined, identify success/failure situations after clicking the “Log in” button and capture them using web elements and conditionals. Then print messages according to the test results. Use the driver to close the browser after all tests are executed.

On all test scenarios using different wait strategies, identify any differences in wait times, error handling, or other relevant observations.

This exercise will help you gain hands-on experience in implementing wait strategies in Selenium using Java. It will enhance your understanding of how waits can be used to handle delays and synchronization issues while automating web applications.

Assertions

https://youtu.be/XHNmf1AaE_s

Practice

You are automating the search functionality of **https://www.youtube.com/** web application. The search process involves entering a search query, clicking on the search button, and validating the search results. You need to implement TestNG assertions to verify the correctness of the search results.

Create a new Selenium - Java - TestNG project, and configure it to launch and test the required webpage on chrome browser. Implement on your framework the following search scenarios on different TestNG test methods:

- Search for TestNG on the web page and validate each of the test results, using TestNG assertions to compare the actual search results with the expected results. For example, you can assert that the search results contain specific keywords or that the number of results is as expected.

- Implement the same logic from the previous test but changing TestNG search value for Selenium search value. Adapt the test results assertions as needed.
- Implement a new Test validating that the results from the "Selenium" Search are different from the "TestNG" Search, use multiple assertions on this test.
- Add a test which will fail, expecting the web app to launch the Log in form right after searching for "Log In" on the search section. Make sure to create assertions for all required elements on the login form"

Run the test suite and observe the TestNG reports, which will indicate the success or failure of each test method and assertion. Analyze any failures or errors to identify and resolve issues in your automation code.

This exercise will provide you with hands-on experience in implementing TestNG assertions in a Java Selenium web automation project. It will help you gain confidence in using assertions to validate the expected results and ensure the correctness of your automation tests.

Additional Resources

https://www.lambdatest.com/blog/asserts-in-testng/
https://www.blazemeter.com/blog/ui-test-automation

POM / Page Factory

https://youtu.be/MO9E9OLqyzc

Practice

You are automating the **https://www.saucedemo.com/** web application.

Create a new Selenium - Java - TestNG project, and configure it to launch and test the required webpage on Chrome browser. Implement on your framework the following scenarios, each one as a different test:

- Purchase a product: Follow the complete buy flow of the page, selecting a random product, adding it to the cart, adding the personal data, and checking that you are successfully arriving at the "Thank you for your purchase" page.
- Removing elements of the shopping cart: Add 3 different elements to the shopping cart, enter the cart page, remove them, and check the shopping cart is empty.
- Logout: try to log out and check if you are correctly redirected to the login page.

For this implementation, follow the rules described next:

- Page Object Model should be correctly implemented, reusing certain elements and defining base pages as needed for common methods and elements.
- Page factory is required.
- Before annotations should be used to manage preconditions, any other required annotations can be implemented as needed.

Alerts, iFrames and Dropdowns

https://youtu.be/quJNMRBnwVQ

Practice

You are automating the **https://www.globalsqa.com/demo-site/frames-and-windows/** web application.

Create a new Selenium - Java - TestNG project, and configure it to launch and test the required webpage on Chrome browser. Implement on your framework the following process, using page objects as needed.

1. Click on the “Open new tab” tab.
2. Click the blue “Click Here” button to open the new tab.
3. Switch to the just opened new tab and click on the Iframe tab.
4. On the “Trainings” dropdown, inside the iframe, select software testing.
5. Click on the “Manual testing” training.
6. Check with the required assertions, the manual testing training page is correctly displayed.
7. On the right sidebar, on the “Miscellaneous” Section, click on the “AlertBox” link.
8. Close the current tab and switch to the initial tab on the browser.
9. Navigate to **https://demo.automationtesting.in/Alerts.html** webpage.
10. Click the “Alert with OK & Cancel” tab.
11. Click on the button to display the confirm box.
12. Click on the Cancel button on the Alert box.
13. Assert the “You Pressed Cancel” Text is displayed below the button.