Introduction to Selenium and its components

History of Selenium :

A software engineer, **Jason Huggins**, at ThoughtWorks in Chicago, developed Selenium in 2004. He created the program to automate testing of a web application called JavaScriptTestRunner. He was looking for an easier way to automate testing of the application, which was repetitive and boring to do manually.

Selenium was initially used by ThoughtWorks employees, but was later made open source and renamed Selenium Core. Selenium has become a powerful tool for automating testing of web applications across different platforms.

Selenium : Selenium is a suite of tools for automating web browsers.

Selenium Components :

- **Selenium RC :** Selenium RC (Remote Control):

  - Legacy component for automating web applications.

  - Required a server-based setup.

  - Less efficient compared to newer components.

  - Not recommended for new projects.

  - Selenium WebDriver is the preferred component for modern web automation.

  Selenium RC is a legacy component of the Selenium suite that was primarily used for automating web applications. It worked on a client-server architecture:

Server: A Selenium RC server process ran on the machine under test.

Client: Client libraries for different programming languages (Java, Python, C#, etc.) interacted with the server.

How it worked:

The client library sent commands to the server.

The server executed these commands on the web browser running on the test machine.

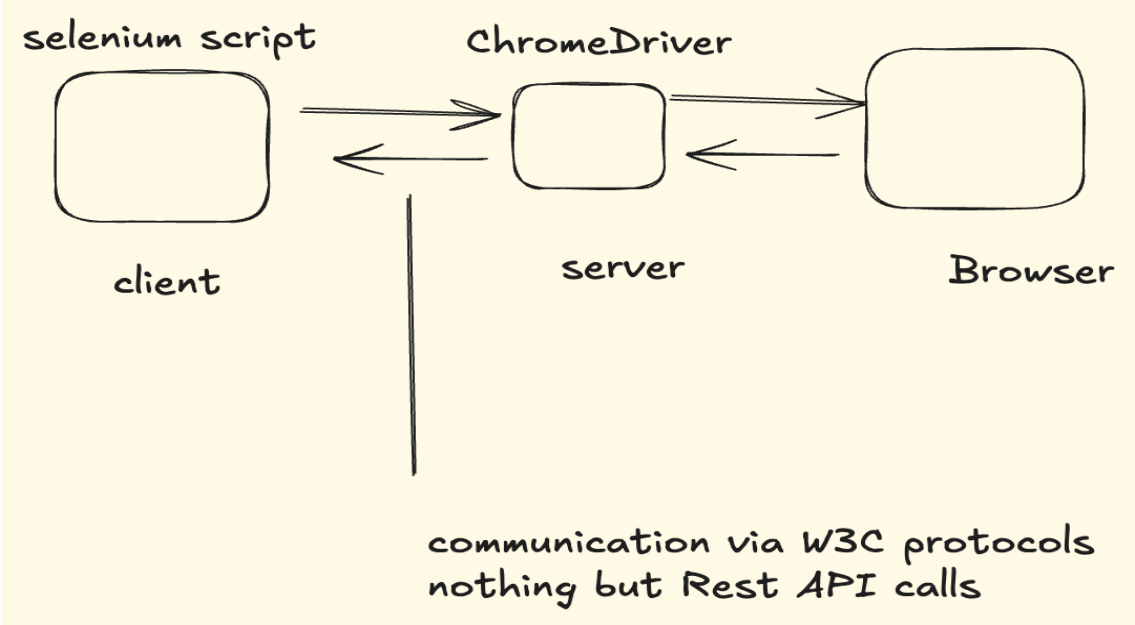
The server returned the results of the actions to the client.

- **Selenium IDE :** A browser extension for recording and playback simple tests.
- **WebDriver :** Selenium WebDriver is the primary tool for most modern test automation projects. Selenium WebDriver is the core for directly controlling web browsers
- **Selenium Grid :** Enables distributed testing across multiple machines.

Features of Selenium Grid :

1. Distributed testing: Allows for running tests on multiple machines (nodes) simultaneously.
2. Parallel execution: Enables faster test execution by distributing tests across different browsers and operating systems.
3. Hub-and-node architecture: Consists of a central hub that manages test sessions and distributes them to nodes.
4. Scalability: Can be easily scaled to handle large-scale test automation projects.

Selenium WebDriver Architecture :



Sample code to launch a chrome browser and load a demo web application and fetch the visible text from that demo website.

```
WebDriver driver = new ChromeDriver();

driver.get("https://demo.evershop.io/account/login");

Thread.sleep(3000);

WebElement loginText =
driver.findElement(By.xpath("//h1[@class='text-center']"));

String textFromUI = loginText.getText();

System.out.println(textFromUI);
```

Understanding Page Elements & Locators in Selenium

There are several different types of locators that can be used, including:

- **ID**: This locator type uses the unique ID attribute of an element to locate it on the page.
- **Name**: This locator type uses the name attribute of an element to locate it on the page.
- **Class name**: This locator type uses the class attribute of an element to locate it on the page.
- **Tag name**: This locator type uses the HTML tag name of an element to locate it on the page.
- **Link text**: This locator type uses the text of a link to locate it on the page.
- **Partial link text**: This locator type uses part of the text of a link to locate it on the page.

- **CSS selector**: This locator type uses a CSS selector to locate an element on the page.
- **Xpath:** This locator type uses an XPath expression to locate an element on the page.
- When writing test scripts with Selenium, you can use a combination of these locator types to accurately and reliably locate elements on the page.

With some examples :

1. **ID**: This locator uses the unique id attribute of an element to locate it. For example, if the HTML for an element on the page looks like this: ... , you can use the ID locator "#some-id" to find this element.

2. Name: This locator uses the name attribute of an element to locate it. For example, if the HTML for an element on the page looks like this: <input name="username">, you can use the Name locator "username" to find this element.

3. Class Name: This locator uses the class attribute of an element to locate it. For example, if the HTML for an element on the page looks like this: <div

class="some-class"> , you can use the Class Name locator ".some-class" to find this element.

4. Link Text: This locator uses the visible text of a link element to locate it. For example, if the HTML for a link on the page looks like this: ABCD, you can use the Link Text locator "ABCD" to find this element.

5. Partial Link Text: This locator is similar to the Link Text locator, but it only matches a portion of the link text. For example, using the Partial Link Text locator "ABCD" would match a link with the text "Welcome to ABCD".

6. CSS Selector: This locator uses a CSS selector to locate an element. CSS selectors are strings that specify how to find an element on a page based on its HTML structure. For example, if the HTML for an element on the page looks like this: ... , you can use the CSS selector "div.some-class#some-id" to find this element.

7. XPath: This locator uses an XPath expression to locate an element. XPath is a language for navigating and selecting elements in an XML document (including HTML documents). It allows you to specify complex, hierarchical patterns for locating elements on a page. For example, if you want to find all the <p> elements that are descendants of the element with the ID "some-id", you could use the XPath expression "//div[@id='some-id']/p" to find these elements.

These are the main types of locators that are used in Selenium. Which one you use will depend on the specific elements you are trying to locate on the page, and the HTML structure of the page itself.

Some examples we practiced in class :

```
//htmltag[@attribute='value'] -> element



//input[@id='value']  -> valid
```

```
//*[@id="dummy"] -> valid



//  //div[@class='inventory_item_name ']

//  (//div[@class='inventory_item_name '])[1] -> first element from list using index based.

//   //div[text()='Sauce Labs Backpack'] -> identify using visible text

//   (//div[@class='inventory_item_name '])[last()]  -> get the last element from list.

//    //div[contains(text(),'T-Shirt')]

//    //*[contains(@id,'img_link')]

//    //input[@name='login-button' and @id='login-button']

//    //input[@name='login-button' or @id='login-button']

//    //div[@id='root']//input  -> descendant concept

// //input[@name='login-button']/..   -> child to parent

//  //input[@name='login-button']/parent::form  - child to parent

//  //input[@name='login-button']/preceding::div[1]   - to find the preceding element
```

```
//  //input[@name='login-button']/../div[1]/following::div[1]
```

Frequently asked Interview questions :

1. What is the 1st line of script i.e WebDriver driver = new ChromeDriver() ?
2. Explain the Selenium Architecture?
3. How Browser is launched automatically? What is happening behind the scenes?
4. Mention the locators of Selenium WebDriver?
5. Which is the best locator in terms of Performance in selenium WebDriver?
6. Explain all locators and when we use them.
7. How many types of Xpaths are present?
8. Difference between Relative and Absolute Xpath?
9. Which Xpath is better? Relative or absolute.
10. Write a syntax for relative and absolute Xpath.
11. What do you mean by single slash ‘/’ and double slash ‘//’ in Xpath?
12. What are Xpath functions or Xpath Axes and mentions some of them.
13. What is a CSS selector and can you write a simple css selector to find an element?