**SELENIUM WEBDRIVER**



* **WEBDRIVER is an *INTERFACE (100% abstract).***
* It is a **cross browser Web UI Functional Test Automation tool.**
* **WebDriver** is a **web automation framework/API** that allows to execute tests against different browsers.
* **Selenium** can support different type of **browsers** for automation.
* *Selenium can be integrated with* ***TestNG*** *to perform* ***Multi Browser Testing****.*

**A screenshot of a computer

Description automatically generated**

**JAVASCRIPT EXECUTOR<INTERFACE>**

CHROME<CLASS>

FF >45<CLASS>

IE<CLASS>

**WEBDRIVER<INTERFACE>**

***Selenium WebDriver Architecture***

1. **Internal Source code**

***1. RemoteWebDriver - A Class.***

***2. WebDriver and JavaScriptExecutor - both are INTERFACE***

***3. browser drivers - IEDriver/ ChromeDriver/ FirefoxDriver – Classes***

* **RemoteWebDriver** **implements** WebDriver and JavaScriptExecutor.
* **Browser driver** class **extends** RemoteWebDriver (inherits properties of RemoteWebDriver)

1. **Operational Architecture/ Internal working**

Selenium Webdriver API has **4 components**

1. **Selenium client library** – Selenium language binding

API for each language in Selenium client library.

1. **JSON wire protocol over HTTP** - REST API for data transmission between clients and HTTP Servers of each Browser Drivers.
2. **Browser Drivers** – Each browser has corresponding Browser Driver.

Communication between browsers without revealing the internal functionality of the browser.

1. **Browsers** – **Real browsers.**

* **Selenium standalone server** is needed to **execute program (code**) to **run remote Webdriver over the protocol.**
* **Selenium client library** will send the **request** via **JSON** wire protocol over **HTTP** for each **command**/ each line of program[JSON wire protocol is a Server].
* **Every statement** in the program/script/Test method will be converted as **URL** with the help of **JSON wire protocol over HTTP.**
* **URLs** will be passed to the **Browser drivers** and they **pass the request to Real Browser over HTTP and real Browsers are invoked.**

***Integration with eclipse***

Select "java\_project\_name" --> right click on "build path" --> "Configure build path"

--> select "Libraries" tab --> selec "Add External JARs"

--> select the downloaded webdriver jar file --> click OK.

\*\* It will appear under **"Referenced Libraries".**

***Cross Browser Testing***

* is a technique to **test** **web application** with different web **browsers**.
* is a type of **functional test** to check that web application works as expected in different browsers.

******

***Locators in Selenium:***

The locator can be termed as ***an address that identifies a web element uniquely within the webpage.***

***Locators are the HTML properties of a web element which tells the Selenium about the web element it needs to perform the action on.***

There is a diverse range of web elements. The most common amongst them are: **Text box / Button / Drop Down / Hyperlink / Check Box / Radio Button**

With the varied range of web elements comes a vast province of strategies/approaches to locate these web elements.

***Web Elements are the basic building blocks of a web page.***

***Locators provide a way to access the HTML elements from a web page.***

***Locator is a command that tells Selenium which GUI elements (Text Box, Buttons, Check Boxes etc) it needs to operate on.***

Selenium uses locators to find and match the elements of the webpage that it needs to interact with.

In Selenium, we can use locators to perform actions on the text boxes, links, checkboxes and other web elements.

***Exceptions in Selenium WebDriver and methods to handle them.***

*An* ***Exception*** *is an* ***event****, which occurs during the execution of a program that* ***disrupts the normal flow*** *of the program’s instructions or in simple words, any issue which makes your test case stop in between the execution.”*

When exception occurs, the normal flow of program halts & an ***exception object*** is created. The program then tries to find someone that can handle the raised exception. *The exception object contains a lot of debugging information such as method hierarchy, line number where the exception occurred, type of exception etc.*

***The process of creating the exception object and handing it over to run-time environment is called “throwing the exception”.***

***To Handle Exceptions in Selenium WebDriver: 2 ways***

1. ***Try Catch:***
2. **Throws** declaration

***Types of WebDriver exceptions***

|  |  |  |
| --- | --- | --- |
|  | ***Exception*** | ***Description*** |
|  | ***ElementClickInterceptedException*** |  |
|  | ***ElementNotInteractableException*** |  |
| 1 | ***ElementNotVisibleException*** | *Although an element is present in the DOM, it is not visible  (****cannot be interacted with****).  E.g. Hidden Elements – defined in HTML using type=”hidden”.* |
| 2 | ***ElementNotSelectableException*** | *Although an element is present in the DOM, it may be disabled  (cannot be clicked/selected).* |
| 3 | ***InvalidSelectorException*** | *Selector used to find an element does not return a WebElement.  Say XPath expression is used which is either syntactically invalid or does not select WebElement* |
| 4 | ***NoSuchElementException*** | *WebDriver is unable to identify the elements during run time, i.e. FindBy method can’t find the element* |
| 5 | ***NoSuchFrameException:*** | *WebDriver is switching to an invalid frame, which is not available.* |
| 6 | ***NoAlertPresentException*** | *WebDriver is switching to an invalid alert, which is not available.* |
| 7 | ***NoSuchWindowException*** | *WebDriver is switching to an invalid window, which is not available.* |
| 8 | ***StaleElementReferenceException*** |  |
| 9 | ***SessionNotFoundException*** | *The WebDriver is performing the action immediately after ‘quitting’ the browser.* |
| 10 | ***TimeoutException:*** | *The command did not complete in enough time.  E.g. the element didn’t display in the specified time. Encountered when working with waits.* |
| 11 | ***WebDriverException:*** | *The WebDriver is performing the action immediately after ‘closing’ the browser.* |

***Waits in Selenium:***

**Implicit Wait**

The **Implicit Wait in Selenium** is used to tell the web driver to wait for a certain amount of time before it throws a "No Such Element Exception". The default setting is 0. Once we set the time, the web driver will wait for the element for that time before throwing an exception.

**Syntax**:

driver.manage().timeouts().implicitlyWait(TimeOut, TimeUnit.SECONDS);

Implicit wait will accept 2 parameters, the first parameter will accept the time as an integer value and the second parameter will accept the time measurement in terms of SECONDS, MINUTES, MILISECOND, MICROSECONDS, NANOSECONDS, DAYS, HOURS, etc.

Explicit Wait

is used to tell the Web Driver to wait for certain conditions (Expected Conditions) or maximum time exceeded before throwing "ElementNotVisibleException" exception.

 it can be applied only for specified elements. It gives better options than implicit wait as it waits for dynamically loaded Ajax elements.

Setting Explicit Wait is important in cases where there are certain elements that naturally take more time to load. If one sets an implicit wait command, then the browser will wait for the same time frame before loading every web element. This causes an unnecessary delay in executing the test script.

**Object Instantiation for WebDriverWait class**

WebDriverWait wait = **new** WebDriverWait(driver,TimeOut);

*wait.until(ExpectedConditions.visibilityOfElementLocated(Webelement element)));*

*element. click();*

The following are the Expected Conditions that can be used in Explicit Wait

alertIsPresent()

elementSelectionStateToBe()

elementToBeClickable()

elementToBeSelected()

frameToBeAvaliableAndSwitchToIt()

invisibilityOfTheElementLocated()

invisibilityOfElementWithText()

presenceOfAllElementsLocatedBy()

presenceOfElementLocated()

textToBePresentInElement()

textToBePresentInElementLocated()

textToBePresentInElementValue()

titleIs()

titleContains()

visibilityOf()

visibilityOfAllElements()

visibilityOfAllElementsLocatedBy()

visibilityOfElementLocated()

**Fluent wait**

The Fluent Wait command defines the maximum amount of time for Selenium WebDriver to wait for a certain condition to appear. It also defines the frequency with which WebDriver will check if the condition appears before throwing the “**ElementNotVisibleException**”.

Fluent Wait looks for a web element repeatedly at regular intervals until timeout happens or until the object is found.

Fluent Wait commands are most useful when interacting with web elements that can sometimes take more time than usual to load. This is largely something that occurs in Ajax applications.

While using Fluent Wait, it is possible to set a default polling period as needed. The user can configure the wait to ignore any exceptions during the polling period.

Wait wait = new FluentWait(WebDriver reference)

.withTimeout(timeout, SECONDS)

.pollingEvery(timeout, SECONDS)

.ignoring(Exception.class);

**Difference between Implicit and Explicit Wait**

|  |  |
| --- | --- |
| **Implicit Wait** | **Explicit Wait** |
|  |  |
| Applies to all elements in a test script. | Applies only to specific elements as intended by the user. |
| No need to specify “ExpectedConditions” on the element to be located | Must always specify “ExpectedConditions” on the element to be located |
| Most effective when used in a test case in which the elements are located with the time frame specified in implicit wait | Most effective when used when the elements are taking a long time to load. Also useful for verifying property of the element such as visibilityOfElementLocated, elementToBeClickable,elementToBeSelected |

***DesiredCapabilities***

***A class in org.openqa.selenium.remote package which extends MutableCapabilities class and this is used to set a series of key value pairs.***

***Handle SSL certificate in Selenium:***

**SSL Certificate Error Handling in Firefox**

ProfilesIni prof = new ProfilesIni()

FirefoxProfile ffProfile= prof.getProfile ("myProfile")

ffProfile.setAcceptUntrustedCertificates(true)

ffProfile.setAssumeUntrustedCertificateIssuer(false)

WebDriver driver = new FirefoxDriver (ffProfile)

## SSL Certificate Error Handling in Chrome

DesiredCapabilities handlSSLErr = DesiredCapabilities.chrome ()

handlSSLErr.setCapability (CapabilityType.ACCEPT\_SSL\_CERTS, true)

WebDriver driver = new ChromeDriver (handlSSLErr);

## SSL Certificate Error Handling in IE

driver.navigate ().to ("javascript:document.getElementById('overridelink').click()");

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability(CapabilityType.ACCEPT\_SSL\_CERTS, true);

System.setProperty("webdriver.ie.driver","IEDriverServer.exe");

WebDriver driver = new InternetExplorerDriver(capabilities);

***Scenarios which are automated using Selenium***

1. Browser Launch / maximize / navigate to URL / Static wait / Driver – close
2. No Of Links and Names
3. Radio Button / checkbox
4. Drop Down

Select drpCountry = new Select(driver.findElement(By.name("")));

drpCountry.selectByVisibleText("");

1. DoubleClick

Actions actions = new Actions(driver);

WebElement elementLocator = driver.findElement(By.id("ID"));

actions.doubleClick(elementLocator).perform();

1. Right Click

Actions actions = new Actions(driver);

WebElement elementLocator = driver.findElement(By.id("ID"));

actions.contextClick(elementLocator).perform();

1. Mouse Hover

Actions action = new Actions(webdriver);

WebElement we = webdriver.findElement(By.xpath(“ "));

action.moveToElement(we).build().perform();