Selenium Features & Capabilities of the Framework

Java – It uses Java programming language.

TestNG – It uses TestNG as a testing framework. You can learn more about TestNG from our – [TestNG tutorial](https://artoftesting.com/testng-tutorial).

Maven-based – It will be maven-based. So all the dependencies will be in a POM file and the test suite can be triggered using maven commands.

Hybrid Framework – It will be a hybrid framework with a combination of the modular and data-driven framework.

Page Object Model – The framework will use the [Page Object Model design pattern](https://artoftesting.com/pageobjectmodel) in Selenium.

Page Factory – It will use [page factory implementation](https://artoftesting.com/page-factory-in-selenium) of the page object model in Selenium.

Screenshot on failure – The framework will have the capability to [capture screenshots](https://artoftesting.com/screenshotinselenium) in case of failed tests.

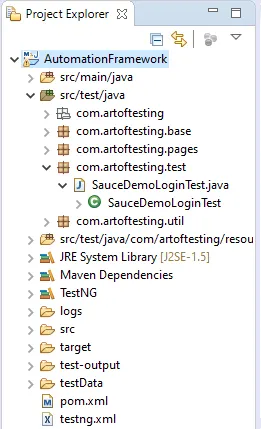
Test data in Excel – The framework will have a utility class that will read test data from an excel file.

Logging – Log4j is used for logging.

Demo site – I will be using the [Saucedemo](https://www.saucedemo.com/) website for the sample test scripts.

Structure of the framework

Let’s talk about the structure of the framework and the different files & folders in the framework.



**TestBase.java inside Base Package**

This is the base file that performs set-up and tear-down operations like –  browser configurations, implicit and explicit waits handling, cookies deletion, etc. Each test class must extend this class.

**Page Classes inside Pages Package**

The ‘pages’ package contains all the page classes. Each page class contains the web elements and actions that can be performed on those classes.

**Test Classes inside Test Package**

The ‘test’ package contains all the test classes. Each test class extends the TestBase.java class and contains the test scripts.

**Util Package**

In the util package, we can have all the utilities e.g. in this framework, I have an ExcelUtil file that reads data from an excel file and converts it into 2D array.

**Resources folder**

Inside the resources folder, we can have different configuration files like log4j2.xml file.

**Logs folder**

The logs folder contains all the log files generated while running the test scripts.

**TestData folder**

The testData folder contains the test data for the test scripts e.g. we can have credentials and other sets of test data that are used by our test scripts.

**Pom.xml file**

The pom.xml contains all the dependencies used in the project.

**TestNG.xml**

TestNG.xml file contains the testNG configurations using which we can run a particular method, group, or test class and at the same time configure the tests to run in parallel.

**Selenium 4 features:**

1. WebDriver became W3C (World Wide Web Consortium) Standardization: standardization of web technologies, making the framework more stable and reducing complexities across browsers
2. Enhanced Selenium Grid : comes with Docker support, support IPv6 addresses and HTTPS protocol, grid instances can be written in TOML (Tom’s Obvious, Minimal Language)
3. Upgraded Selenium IDE
4. Relative Locators in Selenium 4 : To left of, To right of, Above, Below, Near (50px)

Ex:

import static org.openqa.selenium.support.locators.RelativeLocator.with;

WebElement submitButton = driver.findElement(By.id("submit"));

WebElement cancelButton = driver.findElement(**with**(By.tagName("button")).**toLeftOf**(submitButton));

1. Improved Documentation
2. Support for Chrome Debugging Protocol

While Selenium 4 provides direct access to the Chrome DevTools Protocol (CDP), it is highly encouraged that you use the WebDriver Bidi (Bi Directional) APIs instead.

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.devtools.DevTools;

public void performanceMetricsExample() {

ChromeDriver driver = new ChromeDriver();

DevTools devTools = driver.getDevTools();

devTools.createSession();

devTools.send(Performance.enable(Optional.empty()));

List<Metric> metricList = devTools.send(Performance.getMetrics());

driver.get("https://google.com");

driver.quit();

for(Metric m : metricList) {

System.out.println(m.getName() + " = " + m.getValue());

}

}

//Remote driver

public void performanceMetricsExample() {

ChromeOptions chromeOptions = new ChromeOptions();

WebDriver driver = **new RemoteWebDriver(new URL("<grid-url>"), chromeOptions);**

driver **= new Augmenter().augment(driver);**

DevTools devTools = **((HasDevTools) driver).**getDevTools();

devTools.createSession();

devTools.send(Performance.enable(Optional.empty()));

List<Metric> metricList = devTools.send(Performance.getMetrics());

driver.get("https://google.com");

driver.quit();

for(Metric m : metricList) {

System.out.println(m.getName() + " = " + m.getValue());

}

}

1. Better Window/Tab Management in Selenium 4

driver.switchTo().newWindow(WindowType.WINDOW);

driver.switchTo().newWindow(WindowType.TAB);

1. Deprecation of Desired Capabilities

Use:

Firefox – FirefoxOptions

Chrome – ChromeOptions

Internet Explorer (IE) – InternetExplorerOptions

Microsoft Edge – EdgeOptions

Safari – SafariOptions

1. Modifications in Action Class

moveToElement(onElement).click() changed to click(WebElement)

moveToElement(onElement).clickAndHold() to clickAndHold(WebElement)

contextClick(WebElement)

contextClick(WebElement)

release() moved to action class from interactions.ButtonReleaseAction class

1. Capture screenshot of specific web element:

WebElement logo=driver.Findelement (By.xpath(“//div[@id=’divLogo’]//img”));

File file=**logo.getScreenshotAs(OutputType.FILE);**

File destFile =new File(“logo.png”);

FileUtils.copyFile(file,destfile);

1. TimeUnits is deprecated everywhere, use Duration class
2. Deprecation of FindsBy

**Shared capabilities**

browserName : If the specified browser is not installed at the remote end, the session creation will fail

browserVersion : optional, if ask for Chrome version 75 on a system that only has 80 installed, the session creation will fail

pageLoadStrategy : chromeOptions.setPageLoadStrategy(PageLoadStrategy.NORMAL);

normal complete Used by default, waits for all resources to download

eager interactive DOM access is ready, but other resources like images may still be loading

none Any Does not block WebDriver at all

platformName : This identifies the operating system at the remote-end

acceptInsecureCerts : This capability checks whether an expired (or) invalid TLS Certificate is used while navigating during a session

If the capability is set to false, an insecure certificate error will be returned as navigation encounters any domain certificate problems. If set to true, invalid certificate will be trusted by the browser.

unhandledPromptBehavior : Specifies the state of current session’s user prompt handler. Defaults to dismiss and notify state

dismiss, accept, dismiss and notify, accept and notify, ignore

setWindowRect : This command acts as setter to getWindowRect command which accepts width, height, x, y as optional arguments

maximized, minimized, normal, fullscreen

strictFileInteractability : This new capability indicates if strict interactability checks should be applied to input type=file elements

As strict interactability checks are off by default, there is a change in behaviour when using Element Send Keys with hidden file upload controls

Proxy :

Proxy proxy = new Proxy();

proxy.setHttpProxy("<HOST:PORT>");

ChromeOptions options = new ChromeOptions();

options.setCapability("proxy", proxy);

IEOptions:

ignoreProtectedModeSettings: Whether to skip the Protected Mode check while launching a new IE session.

InternetExplorerOptions options = new InternetExplorerOptions();

options.introduceFlakinessByIgnoringSecurityDomains();

WebDriver driver = new RemoteWebDriver(options);

GetTitle: driver.getTitle();

Get current URL: driver.getCurrentUrl();

Navigation:

driver.get("https://selenium.dev"); //wait for page load, no browser history, cant use navigation back

driver.navigate().to("https://selenium.dev"); // no wait for page load, has browser history, can use back after navigation

driver.navigate().back();

driver.navigate().forward();

driver.navigate().refresh();

Alerts:

Alert alert = wait.until(ExpectedConditions.alertIsPresent());

String text = alert.getText();

alert.accept();

alert.dismiss();

Confirm:

wait.until(ExpectedConditions.alertIsPresent());

Alert alert = driver.switchTo().alert();

String text = alert.getText();

alert.dismiss();

Prompts:

Alert alert = wait.until(ExpectedConditions.alertIsPresent());

alert.sendKeys("Selenium");

alert.accept();

Cookies:

// Adds the cookie into current browser context

Cookie cookie1 = new Cookie("test2", "cookie2");

driver.manage().addCookie(new Cookie("key", "value"));

// Get cookie details with named cookie 'foo'

Cookie cookie1 = driver.manage().getCookieNamed("foo");

// Get All available cookies

Set<Cookie> cookies = driver.manage().getCookies();

// delete a cookie with name 'test1'

driver.manage().deleteCookieNamed("test1");

driver.manage().deleteCookie(cookie1);

// deletes all cookies

driver.manage().deleteAllCookies();

Same-Site Cookie Attribute : It is introduced to prevent CSRF (Cross-Site Request Forgery) attacks.

As of now this feature is landed in chrome(80+version), Firefox(79+version) and works with Selenium 4 and later versions.

Strict : When the sameSite attribute is set as Strict, the cookie will not be sent along with requests initiated by third party websites

Lax : When you set a cookie sameSite attribute to Lax, the cookie will be sent along with the GET request initiated by third party website.

Cookie cookie = new Cookie.Builder("key", "value").sameSite("Strict").build();

Cookie cookie1 = new Cookie.Builder("key", "value").sameSite("Lax").build();

driver.manage().addCookie(cookie);

driver.manage().addCookie(cookie1);

System.out.println(cookie.getSameSite());

System.out.println(cookie1.getSameSite());

Frames:

//Store the web element

WebElement iframe = driver.findElement(By.cssSelector("#modal>iframe"));

//Switch to the frame

driver.switchTo().frame(iframe);

//Now we can click the button in frame

driver.findElement(By.tagName("button")).click();

//Using the ID

driver.switchTo().frame("buttonframe");

//Or using the name instead

driver.switchTo().frame("myframe");

// Switches to the second frame

driver.switchTo().frame(1);

// Return to the top level

driver.switchTo().defaultContent();

Windows:

//Store the ID of the original window

String originalWindow = driver.getWindowHandle();

//Check we don't have other windows open already

assert driver.getWindowHandles().size() == 1;

//Click the link which opens in a new window

driver.findElement(By.linkText("new window")).click();

//Wait for the new window or tab

wait.until(ExpectedConditons.numberOfWindowsToBe(2));

//Loop through until we find a new window handle

for (String windowHandle : driver.getWindowHandles()) {

if(!originalWindow.contentEquals(windowHandle)) {

driver.switchTo().window(windowHandle);

break;

}

}

//Wait for the new tab to finish loading content

wait.until(ExpectedConditons.titleIs("Selenium documentation"));

// Opens a new tab and switches to new tab

driver.switchTo().newWindow(WindowType.TAB);

// Opens a new window and switches to new window

driver.switchTo().newWindow(WindowType.WINDOW);

//Close the tab or window

driver.close();

//Switch back to the old tab or window

driver.switchTo().window(originalWindow);

//close all windows and quit browser

driver.quit();

Get window size:

Dimension size = driver.manage().window().getSize();

int width1 = size.getWidth();

int height1 = size.getHeight();

Set window size:

driver.manage().window().setSize(new Dimension(1024, 768));

Get position of window: Fetches the coordinates of the top left coordinate of the browser window.

Point position = driver.manage().window().getPosition();

int x1 = position.getX();

int y1 = position.getY();

Set position:

driver.manage().window().setPosition(new Point(0, 0));

//Maximize

driver.manage().window().maximize();

//Minimize

driver.manage().window().minimize();

//Full Screen

driver.manage().window().fullscreen();

TakeScreenshot:

//Entire Screen

File scrFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(scrFile, new File("./image.png"));

//Element only

WebElement element = driver.findElement(By.cssSelector("h1"));

File scrFile = element.getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(scrFile, new File("./image.png"));

JavasScript execute:

//Creating the JavascriptExecutor interface object by Type casting

JavascriptExecutor js = (JavascriptExecutor)driver;

//Button Element

WebElement button =driver.findElement(By.name("btnLogin"));

//Executing JavaScript to click on element

js.executeScript("arguments[0].click();", button);

//Get return value from script

String text = (String) js.executeScript("return arguments[0].innerText", button);

//Executing JavaScript directly

js.executeScript("console.log('hello world')");

Print Page:

printer = (PrintsPage) driver;

PrintOptions printOptions = new PrintOptions();

printOptions.setPageRanges("1-2");

Pdf pdf = printer.print(printOptions);

String content = pdf.getContent();

Elements:

class name Locates elements whose class name contains the search value (compound class names are not permitted)

css selector Locates elements matching a CSS selector

id Locates elements whose ID attribute matches the search value

name Locates elements whose NAME attribute matches the search value

link text Locates anchor elements whose visible text matches the search value

partial link text Locates anchor elements whose visible text contains the search value. If multiple elements are matching, only the first one will be selected.

tag name Locates elements whose tag name matches the search value

xpath Locates elements matching an XPath expression

Relative Locators : Selenium 4 onwards

Selenium uses the JavaScript function getBoundingClientRect() to determine the size and position of elements on the page, and can use this information to locate neighboring elements.

WebElement fruit = driver.findElement(By.cssSelector("#fruits .tomatoes"));

WebElement fruits = driver.findElement(By.id("fruits"));

//Multiple matches

List<WebElement> plants = driver.findElements(By.tagName("li"));

for (WebElement element : elements) {

System.out.println("Paragraph text:" + element.getText());

}

//Find element by element

WebElement element = driver.findElement(By.tagName("div"));

List<WebElement> elements = element.findElements(By.tagName("p"));

//Get Active element, focused element

String attr = driver.switchTo().activeElement().getAttribute("title");

Interactions:

WebElement ele = driver.findElement(By.name("q"));

//Click

Ele.click();

//Enter text or upload file

Ele.sendKeys(“input text or path to file”);

//Clear

Ele.clear();

//Submit, removed in Sel 4

Ele.submit()

//check if isDisplayed

Ele.isDisplayed();

//check if enabled

Ele.isEnabled();

//check if selected

ele.isSelected();

//getTagName

ele.getTagName();

//get size and position

Rectangle res = ele.getRect();

// Rectangle class provides getX,getY, getWidth, getHeight methods

System.out.println(res.getX());

//get CSS value

ele.getCssValue("color");

//get text content

Ele.getText();

//get attributes

List<String> attrs = Ele.getAttributes();

Ele.getAttribute(“attr”);

//Select list

WebElement selectElement = driver.findElement(By.id("selectElementID"));

Select selectObject = new Select(selectElement);

selectObject.selectByIndex(1);

selectObject.selectByIndex(1);

selectObject.selectByVisibleText("Bread");

// Return a List<WebElement> of options that have been selected

List<WebElement> allSelectedOptions = selectObject.getAllSelectedOptions();

// Return a WebElement referencing the first selection option found by walking down the DOM

WebElement firstSelectedOption = selectObject.getFirstSelectedOption();

// Return a List<WebElement> of options that the <select> element contains

List<WebElement> allAvailableOptions = selectObject.getOptions();

selectObject.deselectByIndex(1);

selectObject.deselectByValue("value1");

selectObject.deselectByValue("value1");

selectObject.deselectByValue("value1");

//check if multi selectable

Boolean doesThisAllowMultipleSelections = selectObject.isMultiple();

Remote Webdriver:

ChromeOptions chromeOptions = new ChromeOptions();

chromeOptions.setCapability("browserVersion", "67");

chromeOptions.setCapability("platformName", "Windows XP");

WebDriver driver = new RemoteWebDriver(new URL("<remote web server url>"), chromeOptions);

driver.get("http://www.google.com");

driver.quit();

//upload local file to remote server

driver.setFileDetector(new LocalFileDetector());

driver.get("http://sso.dev.saucelabs.com/test/guinea-file-upload");

WebElement upload = driver.findElement(By.id("myfile"));

upload.sendKeys("/Users/sso/the/local/path/to/darkbulb.jpg");

**Driver manager:**

First method :

import io.github.bonigarcia.wdm.WebDriverManager;

WebDriverManager.chromedriver().setup();

ChromeDriver driver = new ChromeDriver();

Second method :

System.setProperty("webdriver.chrome.driver","/path/to/chromedriver");

ChromeDriver driver = new ChromeDriver();

Waits:

Explicit wait:

WebElement firstResult =

new WebDriverWait(driver,Duration.ofSeconds(10)).until(ExpectedConditions.elementToBeClickable(ele)));

WebElement foo =

new WebDriverWait(driver, Duration.ofSeconds(3)).until(driver -> driver.findElement(By.name("q")));

Implicit wait:

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

FluentWait:

// Waiting 30 seconds for an element to be present on the page, checking

// for its presence once every 5 seconds.

Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)

.withTimeout(Duration.ofSeconds(30))

.pollingEvery(Duration.ofSeconds(5))

.ignoring(NoSuchElementException.class);

WebElement foo = wait.until(new Function<WebDriver, WebElement>() {

public WebElement apply(WebDriver driver) {

return driver.findElement(By.id("foo"));

}

});

Actions Class:

//pause between action

WebElement clickable = driver.findElement(By.id("clickable"));

new Actions(driver)

.moveToElement(clickable)

.pause(Duration.ofSeconds(1))

.clickAndHold()

.pause(Duration.ofSeconds(1))

.sendKeys("abc")

.perform();

**Keyboard action:**

Actions ac = new Actions(driver);

ac.keyDown(Keys.SHIFT)

.sendKeys(“a”)

.keyUp(Keys.SHIFT)

.sendKeys(textField, "Selenium!")

.perform();

//Copy paste

Keys cmdCtrl = platformName.is(Platform.MAC) ? Keys.COMMAND : Keys.CONTROL;

WebElement textField = driver.findElement(By.id("textInput"));

new Actions(driver)

.sendKeys(textField, "Selenium!")

.sendKeys(Keys.ARROW\_LEFT)

.keyDown(Keys.SHIFT)

.sendKeys(Keys.ARROW\_UP)

.keyUp(Keys.SHIFT)

.keyDown(cmdCtrl)

.sendKeys("xvv")

.keyUp(cmdCtrl)

.perform();

Mouse Action:

//click and hold

new Actions(driver)

.clickAndHold(ele)

.perform();

//Click and release

new Actions(driver)

.click(ele)

.perform();

There are a total of 5 defined buttons for a Mouse:

0 — Left Button (the default)

1 — Middle Button (currently unsupported)

2 — Right Button

3 — X1 (Back) Button

4 — X2 (Forward) Button

//context click, right click

new Actions(driver)

.contextClick(ele)

.perform();

//Back click

PointerInput mouse = new PointerInput(PointerInput.Kind.MOUSE, "default mouse");

Sequence actions = new Sequence(mouse, 0)

.addAction(mouse.createPointerDown(PointerInput.MouseButton.BACK.asArg()))

.addAction(mouse.createPointerUp(PointerInput.MouseButton.BACK.asArg()));

((RemoteWebDriver) driver).perform(Collections.singletonList(actions));

//forward click

PointerInput mouse = new PointerInput(PointerInput.Kind.MOUSE, "default mouse");

Sequence actions = new Sequence(mouse, 0)

.addAction(mouse.createPointerDown(PointerInput.MouseButton.FORWARD.asArg()))

.addAction(mouse.createPointerUp(PointerInput.MouseButton.FORWARD.asArg()));

((RemoteWebDriver) driver).perform(Collections.singletonList(actions));

//double click

new Actions(driver)

.doubleClick(ele)

.perform();

//move to element

new Actions(driver)

.moveToElement(hoverable)

.perform();

//offset from element

WebElement tracker = driver.findElement(By.id("mouse-tracker"));

new Actions(driver)

.moveToElement(tracker, 8, 11)

.perform();

//offset from viewport

PointerInput mouse = new PointerInput(PointerInput.Kind.MOUSE, "default mouse");

Sequence actions = new Sequence(mouse, 0)

.addAction(mouse.createPointerMove(Duration.ZERO, PointerInput.Origin.viewport(), 8, 12));

((RemoteWebDriver) driver).perform(Collections.singletonList(actions));

//offset from current position

new Actions(driver)

.moveByOffset(13, 15)

.perform();

//Drag and drop

WebElement draggable = driver.findElement(By.id("draggable"));

WebElement droppable = driver.findElement(By.id("droppable"));

new Actions(driver)

.dragAndDrop(draggable, droppable)

.perform();

//Drag drop offset

WebElement draggable = driver.findElement(By.id("draggable"));

Rectangle start = draggable.getRect();

Rectangle finish = driver.findElement(By.id("droppable")).getRect();

new Actions(driver)

.dragAndDropBy(draggable, finish.getX() - start.getX(), finish.getY() - start.getY())

.perform();

Pen touch: Chromium only, Sel 4.2

while a mouse has 5 buttons, a pen has 3 equivalent button states:

0 — Touch Contact (the default; equivalent to a left click)

2 — Barrel Button (equivalent to a right click)

5 — Eraser Button (currently unsupported by drivers

WebElement pointerArea = driver.findElement(By.id("pointerArea"));

PointerInput pen = new PointerInput(PointerInput.Kind.PEN, "default pen");

PointerInput.PointerEventProperties eventProperties = PointerInput.eventProperties()

.setTiltX(-72)

.setTiltY(9)

.setTwist(86);

PointerInput.Origin origin = PointerInput.Origin.fromElement(pointerArea);

Sequence actionListPen = new Sequence(pen, 0)

.addAction(pen.createPointerMove(Duration.ZERO, origin, 0, 0))

.addAction(pen.createPointerDown(0))

.addAction(pen.createPointerMove(Duration.ZERO, origin, 2, 2, eventProperties))

.addAction(pen.createPointerUp(0));

((RemoteWebDriver) driver).perform(Collections.singletonList(actionListPen));

Scroll:

//scroll to element

new Actions(driver)

.scrollToElement(ele)

.perform();

//scroll by given amount

WebElement footer = driver.findElement(By.tagName("footer"));

int deltaY = footer.getRect().y;

new Actions(driver)

.scrollByAmount(0, deltaY)

.perform();

//Scroll from an element with an offset

WebElement footer = driver.findElement(By.tagName("footer"));

WheelInput.ScrollOrigin scrollOrigin = WheelInput.ScrollOrigin.fromElement(footer, 0, -50);

//WheelInput.ScrollOrigin scrollOrigin = WheelInput.ScrollOrigin.fromViewport(10, 10);

new Actions(driver)

.scrollFromOrigin(scrollOrigin,0, 200)

.perform();

Bi-Directional API : BiDi API

The traditional WebDriver model of strict request/response commands will be supplemented with the ability to stream events from the user agent to the controlling software via WebSockets, better matching the evented nature of the browser DOM.

Because it’s a bad idea to tie your tests to a specific version of a specific browser, the Selenium project recommends using WebDriver BiDi wherever possible.

Example to get metrics from chrome devtools using (CDP):

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.devtools.DevTools;

public void performanceMetricsExample() {

ChromeDriver driver = new ChromeDriver();

DevTools devTools = driver.getDevTools();

devTools.createSession();

devTools.send(Performance.enable(Optional.empty()));

List<Metric> metricList = devTools.send(Performance.getMetrics());

driver.get("https://google.com");

driver.quit();

for(Metric m : metricList) {

System.out.println(m.getName() + " = " + m.getValue());

}

}

Colors:

import org.openqa.selenium.support.Color;

private final Color HEX\_COLOUR = Color.fromString("#2F7ED8");

private final Color RGB\_COLOUR = Color.fromString("rgb(255, 255, 255)");

private final Color RGB\_COLOUR = Color.fromString("rgb(40%, 20%, 40%)");

private final Color RGBA\_COLOUR = Color.fromString("rgba(255, 255, 255, 0.5)");

private final Color RGBA\_COLOUR = Color.fromString("rgba(40%, 20%, 40%, 0.5)");

private final Color HSL\_COLOUR = Color.fromString("hsl(100, 0%, 50%)");

private final Color HSLA\_COLOUR = Color.fromString("hsla(100, 0%, 50%, 0.5)");

private final Color TRANSPARENT = Color.fromString("transparent");

Color loginButtonColour = Color.fromString(driver.findElement(By.id("login")).getCssValue("color"));

assert loginButtonBackgroundColour.asHex().equals("#ff69b4");

assert loginButtonBackgroundColour.asRgba().equals("rgba(255, 105, 180, 1)");

assert loginButtonBackgroundColour.asRgb().equals("rgb(255, 105, 180)");

Working with multi-threads in parallel:

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import io.github.bonigarcia.wdm.WebDriverManager;

public class WebDriverFactoryStaticThreadLocal {

private static ThreadLocal<WebDriver> driver = new ThreadLocal<WebDriver>();

public static void setDriver() {

WebDriverManager.chromedriver().setup();

driver.set(new ChromeDriver());

}

public static WebDriver getDriver()

{

return driver.get();

}

public static void closeBrowser()

{

driver.get().close();

driver.remove();

}

}

//Using the factory

public class TestTwoWithStaticWebDriver {

@BeforeClass

public void setUp() throws InterruptedException

{

WebDriverFactoryStaticThreadLocal.setDriver();

System.out.println("Browser setup by Thread "+Thread.currentThread().getId()+" and Driver reference is : "+WebDriverFactoryStaticThreadLocal.getDriver());

}

Grid:

Grid Roles:

Standalone: Standalone is also the easiest mode to spin up a Selenium Grid. By default, the server will be listening on http://localhost:4444, and that’s the URL you should point your RemoteWebDriver tests. The server will detect the available drivers that it can use from the System PATH.

java -jar selenium-server-<version>.jar standalone

Hub and Node(s):

Hub : has components Router, Distributor, Session Map, New Session Queue, Event Bus

java -jar selenium-server-<version>.jar hub

Node :

java -jar selenium-server-<version>.jar node

Distributed : On Distributed mode, each component needs to be started on its own. This setup is more suitable for large Grids.

Step1 : Event Bus: serves as a communication path to other Grid components in subsequent steps.

java -jar selenium-server-<version>.jar event-bus

Step 2 : Session Map: responsible for mapping session IDs to the Node where the session is running.

java -jar selenium-server-<version>.jar sessions

Step 3 : New Session Queue: adds the new session request to a queue, then the distributor processes it.

java -jar selenium-server-<version>.jar sessionqueue

Step 4 : Distributor: Nodes register to it, and assigns a Node for a session request.

java -jar selenium-server-<version>.jar distributor --sessions http://localhost:5556 --sessionqueue http://localhost:5559 --bind-bus false

Step 5 : Router: the Grid entrypoint, in charge of redirecting requests to the right component

java -jar selenium-server-<version>.jar router --sessions http://localhost:5556 --distributor http://localhost:5553 --sessionqueue <http://localhost:5559>

Step 6 : Node(s)

java -jar selenium-server-<version>.jar node

Running Tests :

ChromeOptions chromeOptions = new ChromeOptions();

chromeOptions.setCapability("browserVersion", "100");

chromeOptions.setCapability("platformName", "Windows");

// Showing a test name instead of the session id in the Grid UI

chromeOptions.setCapability("se:name", "My simple test");

// Other type of metadata can be seen in the Grid UI by clicking on the

// session info or via GraphQL

chromeOptions.setCapability("se:sampleMetadata", "Sample metadata value");

WebDriver driver = new RemoteWebDriver(new URL("http://gridUrl:4444"), chromeOptions);

driver.get("http://www.google.com");

driver.quit();

Configuring with TOML:

java -jar selenium-server-<version>.jar <component> --config /path/to/file/<file-name>.toml

[node]

detect-drivers = false

hub = “http://localhost:4444”

[[node.driver-configuration]]

max-sessions = 100

display-name = "Firefox Nightly"

stereotype = "{\"browserName\": \"firefox\", \"browserVersion\": \"93\", \"platformName\": \"MAC\", \"moz:firefoxOptions\": {\"binary\": \"/Applications/Firefox Nightly.app/Contents/MacOS/firefox-bin\"}}"

[[node.driver-configuration]]

display-name = "Chrome Beta"

stereotype = "{\"browserName\": \"chrome\", \"browserVersion\": \"94\", \"platformName\": \"MAC\", \"goog:chromeOptions\": {\"binary\": \"/Applications/Google Chrome Beta.app/Contents/MacOS/Google Chrome Beta\"}}"

[[node.driver-configuration]]

display-name = "Chrome Dev"

stereotype = "{\"browserName\": \"chrome\", \"browserVersion\": \"95\", \"platformName\": \"MAC\", \"goog:chromeOptions\": {\"binary\": \"/Applications/Google Chrome Dev.app/Contents/MacOS/Google Chrome Dev\"}}"

webdriver-executable = '/path/to/chromedriver/95/chromedriver'