**1. Why Selenium**

It’s an Open Source automation tool/API/Framework used for web based testing.

Open Source

Supports all browsers, languages and OS

Easy to integrate multiple tools like TestNG and Maven.

Handy to use

API contains -- Classes, Functions, Interfaces, Lib

**2. What type of test you have automated**

Depends on the project. Functional test cases like

1. Smoke Scenario
2. Regression scenario
3. End-to-end user

**3. What are the challenges you have faced?**

* 1- Sync issues.
* 2- Switching between windows
* 3- Integration with multiple tools (TestNG, Maven)

**4. What is your role in current org**

* Understanding the test and automate them.
* Write test cases and pages
* Keep script stable so that we can get good Return of investment. (Try to run multiple times – we can say its stable).
* Adding new library based on applications.

**5. How many test cases you can automate in one day?**

Very tricky. It depends on the project

* 1- It totally depends on scenario which you are testing.
* 2- It depends on which kind of framework you are using.
* 3- It depends on your base knowledge on Selenium

**6. Can we automate full web application using Selenium?**

Ans-No

1- If app is based on silver light Microsoft we cannot automate

2- Application which has flash objects (ex: YouTube).

3- **Barcode Reader, Captcha etc.**

4- It is hard to automate application which has many dependencies on diff projects.

(If one app is down, then we cannot automate).

**a) SearchContext** is the base interface for WebDriver

**b) WebDriver and WebElement is an interface**.

Eg: FirefoxDriver is a class. It implements all the methods of WebDriver interface.

# **c)** [**Difference between == vs equals() in Java?**](https://stackoverflow.com/questions/7520432/what-is-the-difference-between-vs-equals-in-java)

* == -> is a reference comparison, i.e. both objects point to the same memory location
* .equals() -> evaluates to the comparison of values in the objects

**7. Explain the different exceptions in Selenium WebDriver.**

**TimeoutException**: Thrown when a command performing an operation does not complete in enough time.  
**NoSuchElementException**: FindBy method can’t find the element.

**ElementNotVisibleException**: Thrown when the element is present in DOM (Document Object Mod00el), but not visible on the web page.

**StaleElementException:** Thrown when the element is either deleted or no longer attached to the DOM.

**8. What is exception test in Selenium?**

An exception test is an exception that you expect will be thrown inside a test class. If you have written a test case in such way that it should throw an exception, then you can use the @Test annotation and specify which exception you will be expecting by mentioning it in the parameters.

Example: **@Test(expectedException = NoSuchElementException.class)**

Do note the syntax, where the exception is suffixed with .class

**9. Have you used excel sheet in your project**

Excel sheets is because it can be used as data source for tests.

* **Application URL for all environments: Testing like:** development environment or testing environment or QA environment or staging environment or production/ pre-production environment.
* **User name and password credentials of different environments**: You can store the access credentials of the different applications/ environments in the excel sheet (Development, System, UAT, Production)
* **Test cases to be executed**: You can list down the entire set of test cases in a column and in the next column, you can specify either Yes or No which indicates if you want that particular test case to be executed or ignored.

**10. Selenium Commands**

**Navigation Commands**

driver.navigate().to(“appUrl”);

driver.navigate().back();

driver.navigate().forward ();

driver.navigate().refresh();

**Manage Commands**

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

driver.manage().timeouts().implicitlyWait(50,TimeUnit.*SECONDS*)

driver.manage().deleteAllCookies();

driver.manage().deleteCookieNamed(“sessionKey”)

**11. Challenges in IE browser**

Zooming issue

Protected mode settings

Xpath changes in IE

IE is slow with 64-bit

**12. Link**

Get the links count in the page

**System.out.println (driver.findElement(By.tagName(“a”)).size());**

We have API called HTTPClient which comes with selenium library, it will allow you to send request and capture the response

**Find broken link on WebPage**

1. Find all links using TagName
2. Send request and get response, if response is 200 then link is pass or else broken

**Program:**

List<WebElement> links=driver.findElements(By.tagName("a"));

System.out.println("Total links are "+links.size());

for(int i=0;i<links.size();i++)

{

WebElement ele= links.get(i);

String url=ele.getAttribute("href");

verifyLinkActive(url);

}}

public static void verifyLinkActive(String linkUrl)

{

        try

        {

URL url = new URL(linkUrl);

HttpURLConnection httpURLConnect= (HttpURLConnection) url.openConnection ();

httpURLConnect.setConnectTimeout(3000);

httpURLConnect.connect();

if(httpURLConnect.getResponseCode()==200)

           {

               System.out.println(linkUrl+" - "+httpURLConnect.getResponseMessage());

            }

          if(httpURLConnect.getResponseCode()==HttpURLConnection.HTTP\_NOT\_FOUND)

           {

               System.out.println(linkUrl+" - "+httpURLConnect.getResponseMessage() + " - "+ HttpURLConnection.HTTP\_NOT\_FOUND);

            }

**13. How to handle HTTPs/Secure site**

 DesiredCapabilities cap=DesiredCapabilities.chrome();

// Set ACCEPT\_SSL\_CERTS variable to true

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

**// Set the driver path and open browser with cap**

System.setProperty("webdriver.chrome.driver","C:\\Grid\\chromedriver.exe");

WebDriver driver = new ChromeDriver (cap);

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

**b) Chrome – Incognito**

DesiredCapabilities cap = DesiredCapabilities.chrome();

ChromeOptions options = new ChromeOptions();

**options.addArguments("--incognito");**

**cap.setCapability (ChromeOptions.CAPABILITY, options);**

**// Set the driver path and open browser with cap**

**14. Check box and Radio buttons**

.isSelected() – Will say true or false based on the selection (Used for check-box)

.isEnabled() – True/False (Whether the object is enable/disable)

.size() – give the size of the elements

**Example 1**: Get the count of the radio buttons

int count =driver.findElements(By.xpath("//input[@name='group1']")).size();

for(int i=0;i<count;i++)

{ **//driver.findElements(By.xpath("//input[@name='group1']")).get(i).click();**

String text=driver.findElements(By.xpath("//input[@name='group1']")).get(i).getAttribute("value");

if(text.equals("Cheese"))

{

driver.findElements(By.xpath("//input[@name='group1']")).get(i).click();

} }

**15. How to handle multiple WebElements**

List<WebElement> allQuotes = driver.findElementsByLinkText("REQUEST A QUOTE");

int count = allQuotes.size();

System.out.println(count);

for (WebElement eachQuote : allQuotes)

{

System.out.println(eachQuote.getText());

}

WebElement secondEle = allQuotes.get(1).click();

**15. Direct way to upload file**

driver.findElement(By.xpath().sendkeys(“path of the file”)

**(Or) Using AutoIT**

*//Which calls the autoit exe file* Runtime.getRuntime().exec("G:/Tutorial/AutoItScripts/upload.exe");

**Save the above script with '.au3' which is AutoIt file extension**

WinWaitActive("File Upload")

Send("G:\Tutorial\AutoItScripts\TestScripts\Test.doc")

Send("{ENTER}")

**15. Configuration / Property File**

We can use property file in Selenium which will works as configuration file.

Property file contains key and value pair

**How to load Property File:**

Properties prop = new Properties ();

FileInputStream input = new FileInputStream("config.properties");

prop.load(input); //load a properties file

System.out.println(prop.getProperty("database"));

driver.get(prop.getProperty("URL"));

driver.findElement(By.id("Email")).sendKeys(prop.getProperty("username"));

driver.findElement(By.id("SignIn")).click();

**b) Cross browser testing (Using Property file)**

public class base {

public WebDriver driver;

public WebDriver Initializebrowser() throws IOException

{

Properties prop=new Properties();

FileInputStream input = new FileInputStream("config.properties");

prop.load(input); //load a properties file

String browserName=prop.getProperty("browser");

System.out.println(browserName);

if(browserName.equals("chrome"))

{

System.setProperty("webdriver.chrome.driver","D:\\Selenium Prerequisite\\chromedriver.exe");

driver=new ChromeDriver();

}

else if(browserName.equals("firefox"))

{

System.setProperty("webdriver.gechodriver.driver","D:\\Selenium Prerequisite\\geckodriver.exe");

driver=new ChromeDriver();

}

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

return driver;

}

}

**WebDriver** is an interface which contain several abstract methods such as get(...), findElementBy(...) etc.

**What are different pop-ups that you have handle in your projects?**

* JavaScript Pop Ups --- Alert alert = driver.switchTo().alert();
* Browser Pop Ups --- Browser Profiles, Robot tool Kit, AutoIT
* Native OS Pop Ups--- Browser Profiles, Robot tool Kit, AutoIT

**How to work with dynamic web table?**

You can get the total number of tags within a tag by giving the xpath of the element by using this function - List ele = driver.findElements(By.xpath("Xpath of the table"));

Now you can use a for each loop to loop through each of the tags in the above list and then read each  value by using getText() method.

**Wrapper class:**

A Wrapper class is a class whose object wraps or contains a primitive data types.

**autoboxing** and **unboxing** feature converts primitive into object and object into primitive automatically

1. **Locators used in Selenium**

* className
* cssSelector
* id
* linkText
* partialLinkText
* name
* tagName
* xpath -- **How to handle dynamic element**: By using Xpath and CSS selector

**a) xpath (Complete/Absolute xpath, Partial xpath)**

[XPath](http://www.softwaretestinghelp.com/using-selenium-xpath-and-other-locators-selenium-tutorial-5/) is used to locate a web element based on its XML path.

**Absolute Xpath**: Begins with single frwd slash (/), which means you can select the element from the parent node.

**Eg**: html/body/div[1]/section/div[1]

**Relative Xpath:** Path starts from the middle of the HTML DOM structure. // which means it can search the element anywhere at the webpage. It doesn’t depend on parent nodes.

**Example:** //input[@name=’lang’ and @type=’radio’]

Partial xpath:

//\* -> all elements

//a -> all links

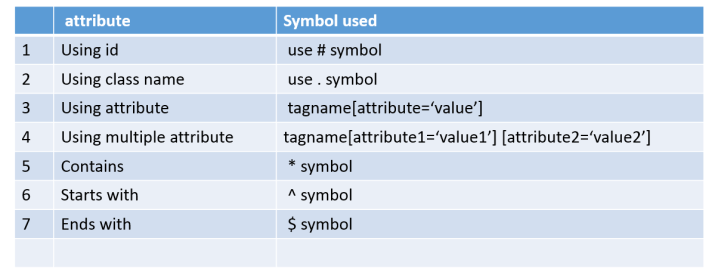
1. **How to traverse to sibling element using Xpath?**

.//\*[@id='tablist1-tab1']/following-sibling::li[2]

1. **How to traverse back to Parent element from Child element using Xpath?**

.//\*[@id='tablist1-tab1']/parent::ul

**b) Symbol used while writing CSS selector in Selenium Webdriver**



* tagname[attribute$='value']
* tagname[attribute^='value']
* tagname[attribute\*='value']
* tagname#id
* tagname.classname

//tagName [@attribute=’value’] --> **Xpath**

tagName[attribute=’value’] --> **cssSelector**

**Example for HTML code:** <input name=”username123”>

**a) CSS selector**

input[name\*=username”]

**b) Xpath**

//input[contains(@name, ‘username’)]

--- (“//input[starts-with(@name,’btn’)]”)

**c) Diff b/w Xpath and CssSelector**

In terms of performance, CSS perform well as compared to [XPATH](http://learn-automation.com/how-to-write-dynamic-xpath-in-selenium/)and CSS will not change based on browsers, that is it will behave same in all browsers but xpath will behave differently in IE browser.

1. **Why Screenshot required in automation?**

Screenshot helps us to understand the flow of application whether the application is behaving correctly or not.

You can track the execution of the test.

TakesScreenshot ts=(TakesScreenshot)driver;

File source=ts.getScreenshotAs(OutputType.***FILE***);

FileUtils.*copyFile*(source,**new** File("./Screenshot/one.jpg")); --Inside the project

System.***out***.println("Screenshot taken");

1. **What is Assert?**

Asserts are used to perform validations in the test scripts. Whether the test has passed or failed.

Assert is a class in TestNG which helps for validation.

**Hard Assert:** When using hard assertions in the test scripts, your test script will stop executing when the assertion fails, and the test will be failed in TestNG report.

**Soft Assert:** If you need to execute the remaining test case even after an assertion fails, and you also want to report assertion and test case failure in TestNG report, TestNG provides soft assertions for this. (i.e. a test script does not stop running even if an assertion fails, but the test itself is failed to indicate the right result)

Do not forget to call assertAll() method at last

**softAssert.assertAll() method**

*Whereas,* ***in case of "Verify", the test method continues execution even after the failure of an assertion statement****. Although the test method will still be marked as failed but the remaining statements of the test method will be executed normally. In TestNG, the "Verify" functionality is provided by means of "SoftAssert" class*

**a) Example program for Hard/Soft Assertion**

*@Test*

public void softAssert()

{

SoftAssert assertion= new SoftAssert();

System.out.println("Test 1 started");

assertion.*assertEquals(*12,13, “Count doesn’t match”);

Assertion.assertAll();

}

*@Test*

public void hardAssert()

{

System.out.println("Test 2 started ");

Assert.*assertEquals*(12,13);

System.out.println("hardAssert Method Was Executed")

}

**Output:**

[TestNG] Running: C:\Users\Administrator\AppData\Local\Temp\testng-eclipse--2097831536\testng-customsuite.xml

Test 2 Started

Test 1 started

Test 1 completes

FAILED: hardAssert

===============================================

Default suite

Total tests run: 2, Failures: 1, Skips: 0

**b) How to verify Error message in Selenium Webdriver?**

Using getText() method you can capture text and store in string. Using Assert verify the error msg

String actual\_error= driver. findElement (By.xpath (“. //\*[@id=’login get failed’]”)). getAttribute(“innerHTML”);

String expected\_error= “Please enter your email”;

**#Type 1**

Assert.*assertEquals*(actual\_error, expected\_error);

**#Type 2**

Assert.*assertTrue*(actual\_error. contains (“Please enter your email”);

S.O.P (“Test is completed”);

1. **Different types of waits in WebDriver?**

**IMPLICIT WAIT:**Implicit waits are used to provide a default waiting time (eg: 5 seconds) between each consecutive test step/command across the entire test script. Thus, subsequent test step would only execute when the 5 seconds have elapsed after executing the previous test step/command.

*(Will be applicable throughout your script).*

driver. manage().timeouts().implicitlyWait(50,TimeUnit.*SECONDS*);

**EXPLICIT WAIT:** Explicit waits are used to halt the execution till the time a particular condition is met or the maximum time has elapsed. Unlike Implicit waits, explicit waits are applied for a particular instance only.

*(Wait for specific condition)*

WebDriverWait wait=new WebDriverWait(driver,30)

WebElement element=wait. until(ExpectedConditions.*visibilityOfElementLocated(By.xpath(“//p[text()=’Web’])));*

**FLUENT WAIT:** Wait for specific condition and we can change polling settings as well.

Let’s say you have an element which sometime appears in just 1 second and some time it takes minutes to appear. In that case it is better to use fluent wait, as this will try to find element again and again until it find it or until the final timer runs out.

// 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(30, SECONDS)

.pollingEvery(5, SECONDS)

.ignoring(NoSuchElementException.class);

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

public WebElement apply(WebDriver driver) {

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

}});

**a) Write a code to wait for a particular element to be visible on a page.**

We can write a code such that we specify the XPath of the web element that needs to be visible on the page and then ask the WebDriver to wait for a specified time. Look at the sample piece of code below:

WebDriverWait wait=new WebDriverWait(driver, 20);

**Element=wait.until (ExpectedConditions.visibilityOfElementLocated(By.xpath(“<xpath>”)));**

**b) Write a code to wait for an alert to appear.**

We can write another piece of code asking the WebDriver to wait until an error appears like this

WebDriverWait wait=new WebDriverWait(driver, 20);

WebElement **Element = wait.until(ExpectedConditions.alertIsPresent());**

1. **How to work with dropdowns?**

**Static Dropdown** -> Select class is capable to handle dropdown if it is inside select tag

**Example:**

<select id="mySelect">

<option value="option1">Cars</option>

<option value="option2">Bikes</option>

</select>

1. Identify the ‘select’ html element (Because dropdowns must have the ‘select’ tag)
2. Select an option from that dropdown element.

WebElement transport= driver.findElement(By.id("mySelect"));

Select type1= new Select(transport);

type1.selectByVisibleText(“Bikes”); → Selecting an option by the text that is visible

type1.selectByIndex(“1”); → Selecting, by choosing the Index number of that option

type1.selectByValue(“option2”); → Selecting, by choosing the value of that option

**Get All option from dropdown (Month example)**

WebElement ele=driver.findElement(By.id("day"));

Select month=new Select(ele);

**List<WebElement> dropdown=month.getOptions();**

for(int i=0; i<dropdown.size(); i++)

{

System.out.println(dropdown.get(i).getText());

 }

If the dropdown doesn’t have any select class, [This is called BootStrap Dropdown](http://learn-automation.com/handle-bootstrap-dropdown-in-selenium-webdriver/) and you can handle them using findElements method.

**When to use SET and LIST**

Whenever we work with multiple windows then we use SET.

**Set <String> win = driver.getwindowhandles();**

Iterator <String> it = win.iterator();

String parentwin = it.next(); // traverse to the next window

String childwin = it.next();

driver.swtichTo().window(childwin);

System.out.println(driver.getTitle());

**getWindowHandles() -->** To get address of all open browser and Return type is Iterator<String> (window handle)

**getWindowHandle() -->** Used to get the address of the current browser and return type is String.

**In case of select or check all checkboxes we use List <WebElement> for example**

List <WebElement> checkbox = driver.findElements(By.xpath(“//input[@type=’checkbox’]”));

1. **Switch To commands (Frames, Alerts, Multiple windows)**

**a) How to switch to a new window (new tab) which opens up after you click on a link?**

In case you do not know the name of the window, then you can use the **driver.getWindowHandles()** command to get the name of all the windows that were initiated by the WebDriver.

Note: that it will not return the window names of browser windows which are not initiated by your WebDriver.

**Program:**

String handle= driver.getWindowHandle();

System.out.println(handle);

 // Click on the Button "New Message Window"

 driver.findElement(By.name("New Message Window")).click();

 // Store and Print the name of all the windows open

Set handles = driver.getWindowHandles();

 System.out.println(handles);

 // Pass a window handle to the other window

 for (String handle1 : driver.getWindowHandles()) {

 System.out.println(handle1);

 driver.switchTo().window(handle1);

 }

**b) FRAME**

In some application you might get some frames then you have to switch frame first and then you perform operations. Normally bootstrap pop-up has two frameworks; we need to switch the framework to select the elements.

driver.switchTo().frame(0) -- Using index

driver.switchTo().defaultContent() -- Move to the parent window

**c) Alerts**

driver.switchTo().alert().accept()

driver.switchTo().alert().dismiss()

driver.switchTo().alert().getText()

driver.switchTo().alert().sendKeys("Text")

**Example:**

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

String alertText = simpleAlert.getText();

System.out.println("Alert text is " + alertText);

simpleAlert.accept();

1. **What is Frame**

An IFrame (Inline Frame) is an HTML document embedded inside another HTML document on a website.

The IFrame HTML element is often used to insert content from another source, such as an advertisement, into a Web page.



**Different ways to switch frames**

1. Using name or id >> driver. switchTo().frame(“w3c”)
2. Using index >> driver. switchTo().frame(0)
3. Using WebElement

WebElement my\_frame=driver.findElement(By.xpath(“iframe[@title(‘selenium news’)]”));

**driver.switchTo().frame(my\_frame)**

**To switch to parent window**

driver.switchTo().defaultContent();

1. **Action Class (Ajax/Mouse Interaction)**

API allows us to perform operations from keyboard events and simple mouse events to complex events such as dragging-and-dropping, holding a key and then performing mouse operations by using the Actions class.

**Actions builder = new Actions(driver);**

**a) Mouse hover**

 // Create object on Actions class

Actions action = new Actions(driver);

// find the element which we want to Select

  WebElement element = driver.findElement(By.linkText("Product Category"));

 // Use mouse hover for that element

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

// finally click on the element you want to select

        driver.findElement(By.linkText("iPads")).click();

**b) Drag and Drop in SE**

WebDriver driver=new FirefoxDriver();

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

driver.get("http://jqueryui.com/resources/demos/droppable/default.html");

Thread.sleep(5000);

Actions act=new Actions(driver);

WebElement drag=driver.findElement(By.xpath(".//\*[@id='draggable']"));

WebElement drop=driver.findElement(By.xpath(".//\*[@id='droppable']"));

// this will drag element to destination

act.dragAndDrop(drag, drop).build().perform();

}

**c) Advance activity in Selenium**

**Using action class we can do**

moveToElement(WebElement)-- Mouse Hover

contextClick()-- Right click on page

contextClick(WebElement)-- Right click on specific Element

doubleClick() – Double click on specific element

sendKeys(KEYS.TAB)--For keyboard events

keyDown(Keys.SHIFT) – To type in capital letters

clickAndHold(WebElement) --Click on element and hold until next operation

release () -- Release the current control

1. **What is JavaScript Executor**

**JavaScriptExecutor** is an interface which provides a mechanism to execute Javascript through the Selenium WebDriver. It provides “**executescript**” and “**executeAsyncScript**” methods, to run JavaScript in the context of the currently selected frame or window.

**Some usage**: You can work with disable WebElement

**Example:**

public static void main(String[] args) throws Exception {

WebDriver driver=new FirefoxDriver();

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

driver.get("http://jqueryui.com");

 // Wait for 5 second

 Thread.sleep(5000);

//This will scroll page 400 pixel vertical

((JavascriptExecutor)driver).executeScript("scroll(0,400)");

 }

}

**a) How to scroll down a page using JavaScript in Selenium?**

We can scroll down a page by using window.scrollBy() function.

Example:

|  |  |
| --- | --- |
|  | ((JavascriptExecutor) driver).executeScript("window.scrollBy(0,500)"); |

**b) How to scroll down to a particular element?**

To scroll down to a particular element on a web page, we can use the function scrollIntoView().

**SYNTAX:**

((JavascriptExecutor) driver).executeScript("arguments[0].scrollIntoView(true);", element);

Thread.sleep(500);

**Example program:**

It will scroll until element is not visible or in view. **Once it visible it will stop scrolling**

 JavascriptExecutor je = (JavascriptExecutor) driver;

 //Identify the WebElement which will appear after scrolling down

 WebElement element = driver.findElement(By.xpath(".//\*[@id='mCSB\_3\_container']/p[3]"));

 // Now execute query which actually will scroll until that element is not appeared on page.

 je.executeScript("arguments[0].scrollIntoView(true);",element);

 // Extract the text and verify

 System.out.println(element.getText());

1. **How to generate Log Files in SE**

**Step 1:** Add log4j jar file to your current project

public class Google {

public static void main(String[] args) {

// Create logger instance, so we need to pass Class name for which we want to create log file.

Logger logger=Logger.*getLogger*("Google");

// configure log4j properties file

PropertyConfigurator.*configure*("Log4j.properties");

WebDriver driver = new FirefoxDriver();

logger.info("Browser Opened");

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

logger.info("Implicit wait given");

driver.get("https://www.google.co.in/");

logger.info("Url opened");

driver.findElement(By.name("q")).sendKeys("Selenium");

logger.info("keyword type);

}

}

1. **Apache POI**

Apache POI (Poor Obfuscation Implementation) is an API written in Java to support read and write operations – modifying office files.

* **XSSFWorkbook:** Represents workbook in xlsx file.
* **HSSFWorkbook:** Represents workbook in xls file.
* **XSSFSheet:** Represents a sheet in XLSX file.
* **HSSFSheet:** Represents a sheet in XLS file.

JXL supports only .xls file

**Example:**

File src=**new** File("filepath/excelsheetname.xlsx");

FileInputStream fis=**new** FileInputStream(src);  **// load file**

XSSFWorkbook wb=**new** XSSFWorkbook(fis);

XSSFSheet sh1= wb.getSheetAt(0)

// get number of rows so that we can run loop based on this

numrow= sh1.getRows();

1. **Selenium Grid**

* Selenium grid is the concept which will allow you to run the test on multiple machines and on multiple browsers (multiple nodes)
* It will reduce the total execution time which is the main advantage of Test Automation.

**Hub**

* Hub is the central point that will receive all the test request and distribute them the right nodes.
* There should only be one hub in a grid.
* The machine containing the hub is where the tests will be triggered, but you will see the browser being automated on the node.

**Node** will be the actual machine, where the test will be executed.

**Commands in Hub Machine**

1. Setting the Hub:

java -jar selenium-server-standalone-3.4.0.jar -role hub

2. Validating Hub started

After starting the hub, we can view the status of the hub by opening any browser window and navigating to: <http://localhost:4444/grid/console>

**Commands in Node Machine** (Check if java is configured in system variables)

1. Download the selenium server jar in node machine as well

2. Accessing Hub from Node Machine

<http://iporhostnameofHub:4444/grid/console>

3. Login to another machine and register it as node for Hub

4. Registering Node with Hub

java -jar selenium-server-standalone-3.4.0.jar -role webdriver - hub http://192.168.43.177:4444/grid/register -port 5566

**Chrome Node Setup**java -Dwebdriver.chrome.driver=C:/eclipse/chromedriver/chromedriver.exe -jar selenium-server-standalone-2.48.2.jar -port **5556** -role node -hub http://localhost:4444/grid/register -browser "browserName=chrome, version=ANY, maxInstances=10, platform=WINDOWS"

**Program:**

public class SeleniumGridTest

{

@Test

public void runFirefox() throws MalformedURLException

{

DesiredCapabilities cap=DesiredCapabilities ();

cap.setBrowserName(“Chrome”);

cap.setPlatform(Platform.WINDOWS);

//For latest 3.8.1 jars

ChromeOptions options=new ChromeOptions();

options.merge(cap);

// Create driver with hub address and capability

WebDriver driver=new RemoteWebDriver(new URL(“<http://localhost:4444/wd/hub>”) , cap);

driver.get("http://learn-automation.com/");

}

}

CLOUD – Browser stack (With three lines of code username, pwd and URL)

**Parallel execution in GRID**

public class BaseClass {

    //ThreadLocal will keep local copy of driver

    public static ThreadLocal<RemoteWebDriver> dr = new ThreadLocal<RemoteWebDriver>();

    @BeforeTest

    //Parameter will get browser from testng.xml on which browser test to run

    @Parameters("myBrowser")

    public void beforeClass(String myBrowser) throws MalformedURLException{

    RemoteWebDriver driver = null;

        if(myBrowser.equals("chrome")){

            DesiredCapabilities capability = new DesiredCapabilities().chrome();

            capability.setBrowserName("chrome");

            capability.setPlatform(Platform.WINDOWS);

            driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), capability);

        }

        else if(myBrowser.equals("firefox")){

            DesiredCapabilities capability = new DesiredCapabilities().firefox();

            capability.setBrowserName("firefox");

            capability.setPlatform(Platform.WINDOWS);

            driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), capability);

        }

**TestNG.XML**

<suite name="Parallel test suite" parallel="classes" thread-count="2">

    <test name="Regression 1">

        <parameter name="myBrowser" value="firefox"/>

        <classes>

            <class name="myPackage.TestParallel" />

            <class name="myPackage.TestParallel" />

        </classes>

    </test>

    <test name="Regression 2">

        <parameter name="myBrowser" value="chrome"/>

        <classes>

            <class name="myPackage.TestParallel" />

            <class name="myPackage.TestParallel" />

        </classes> </test> </suite>

1. **TestNG**

**What is TestNG and Why we have to use in Selenium**

TestNG is a testing framework which is designed to cover all categories of tests:  unit, functional, end-to-end, integration, etc. **(We can control the test case execution using TestNG xml file)**

Using TestNG we can create TCs, group TCs, prioritize TCs, execute TCs and generate reports

1. Unit Test Framework
2. Annotation
3. Support for [data-driven testing](http://learn-automation.com/data-driven-framework-in-selenium-webdriver/) (with @DataProvider)
4. Support for parameters.
5. Generate automatic [reports](http://learn-automation.com/generate-xslt-report-in-selenium/)
6. We can run failed test case only using **testing-failed.xml**, so no need to run the full test suite in case of failure
7. Supported by a variety of tools and plug-ins (Eclipse, IDEA, [Maven](http://learn-automation.com/maven-integration-with-selenium/), etc…).

**b) Failed test cases in TestNG**

TestNG will create separate testing-failed.xml file if test cases are failing. We can run the xml file to trigger the failed test cases again.

If you run the **testing-failed.xml**, and if it get pass. “Failed Suite” folder will be created (Failed test case report only).

**Annotations:**

@Test (description =”Open Browser”)

@Test (priority = 2, description=”Open Yahoo homePage”)

@Test (groups = {“Smoke”})

@BeforeTest

@Test(enabled=false)

@Test(dependsOnMethods= {“Methodname”})

**d) To run Specific test cases**

<test name="Regression1">

**<groups>**

**<run>**

**<exclude name="brokenTests" />**

**<include name="checkinTests" />**

**</run>**

</groups>

<classes>

<class name="test.IndividualMethodsTest">

**<methods>**

**<include name="testMethod" />**

**</methods>**

</class>

</classes>

</test>

**e) Cross Browser Testing (Using TestNG)**

@BeforeTest

@Parameters("browser")

public void setup(String browser) throws Exception{

// Passing Browser parameter from TestNG xml

if(browser.equalsIgnoreCase("chrome")){

System.setProperty("webdriver.chrome.driver",".\\chromedriver.exe");

driver = new ChromeDriver();

}

else if(browser.equalsIgnoreCase("Edge")){

System.setProperty("webdriver.edge.driver",".\\MicrosoftWebDriver.exe");

driver = new EdgeDriver();

}

else{

throw new Exception("Browser is not correct");

}

**f) Parameterization**

@Parameters({ "browser" })

@Test

public void testCaseOne(String browser)

{

System.out.println("browser passed as : " + browser);

}

XML file

<parameter name="browser" value="Firefox"/>

@Parameters(“myName”)

<parameter name=”myName” value=”john”/>

**g) Parallel execution:**

We are defining two attributes 'parallel' and 'thread-count' at suite level. The assigning of the thread is take care by the processor. So we can't say which thread is going to execute which method.

Just set the ‘**parallel**‘ attribute to ‘**tests**‘ in the below used xml and give a run again. This time you will notice that your both browsers will open almost simultaneously and your test will run in parallel.

<suite name="Suite" parallel="tests">

<test name="FirefoxTest">

<parameter name="browser" value="firefox" />

<classes>

<class name="automationFramework.MultiBrowser" />

</classes>

</test>

<test name="IETest">

<parameter name="browser" value="ie" />

<classes>

<class name="automationFramework.MultiBrowser" />

</classes>

</test>

</suite>

1. **TestNG Listeners**

**Listener** is defined as interface that modifes the default TestNG's behavior.

It allows customizing TestNG reports or logs. As the name says it listen to certain events in the Selenium script and behave accordingly. **(Listen to your execution status; it will trigger the specific methods you want).**

**TestListenerAdaptor - Class**

**ITestListener** - **Interface** (It has certain methods which we have to implement). which will keep track of test status and you can customize this as for reporting purpose.

**Different ways to implement**

Class level and Suite level

**IRetryAnalyzer** – You can execute failed test cases automatically multiple times.

================================= +++++++ + +++++++ ================================

**Testing.xml**

<suite name="Loan Department" parallel="none">

<listeners>

<listener class-name="utility.Listener"/>

</listeners>

<test name="Batch-Listeners">

1. **POM – Page Object Model**
2. As the name says we will be working on pages (pages will be pure Java classes).
3. You can store all locators and respective methods in the separate class and Call them from the test in which you have to use.
4. The benefit from this will be if any changes in Page then you do not have to modify the test, simply modify the respective page and that all.

**Page Object Model** is a design pattern to create **Object Repository** for web UI elements.

* Since the Object Repository is independent of Test Cases, multiple tests can use the same Object Repository
* Reusability of code

**Advantages of POM**

Easy to maintain, readable format, reusable scripts.

**2 ways to implement**

Using normal approach

Using PageFactory and @findBy (Imp qns: Difference b/w Pagefactory and normal approach)

1. **What is Page Factory?**

**Page factory:** In order to support PageObject pattern, WebDriver’s support library contains a factory class.

We have separate method to initialize web element of Page object is initElement.

Cache feature that will store all frequently used WebElement in cache

@CacheLookup

@FindBy(id=”user\_login”)

WebElement username;

@FindBy(how=How.***XPATH,***using=”.//\*[@id=’wp-submit’]”)

WebElement submit\_button;

Page Factory gives an optimized way to implement Page Object Model. When we say it is optimized, it refers to the fact that the memory utilization is very good and also the implementation is done in an object oriented manner.

**LoginPage obj=PageFactory.initElements(driver, LoginPage.class);**

we use annotations like: **@FindBy** to find WebElement, and **initElements** method to initialize web elements from the Page Factory class.

1. **Reports**

Extent Report –

* 1. Create dependency in MAVEN pom.xml
  2. Create ExtentReport.java class
  3. In TestNG – add listener for extend report

<listener class-name=”resources.ExtentReport”/>

For Maven – target folder – Surefire suite – index.html (or) ExtentReportTestNG.html –Properties and copy the location in browser and check the report.

public class Listener implements ITestListener {

1. **Online repository to store data (Github)**

You can update all test with framework on Server (github.com) and execute the Jenkins job, which will take the data from Github directly and it will execute the test.

**\*\*\* Add Eclipse project to GitHub. Commit, Push and Pull \*\*\***

1. Create GitHub account and Signin
2. Start Project and Create Repository
3. In Eclipse, Go to Perspective – Github – Add Git as a clone URI and give HTTPS
4. Create a Project in eclipse – Right click on project – Team – Share project – Select the repository (Which u have clone) -- Finish
5. Right click – Team – commit. And then stage all from unstagged. “Commit and Push” the project.

**Via SSH**

Github – New Repository – Repository Name – Create Repository.

To get the code from eclipse – We can use https/SSH.

In Eclipse – Window preference – SSH – Key Management – Generate RSA Key (copy it) – Save Private Key

In Github – Go to home – settings – SSH and GPG keys – New SSH Key and paste the key.

1. **Jenkins (**Maven, Github and Jenkins Integration Steps**)**

**1. In Real Time**: Just give the URL of Jenkins and open the Jenkins. (**Locally**: D:\>Java –jar Jenkins.war)

**2. Configuring Global settings:**

In Jenkins -- Manage Jenkins -- Global tool configuration – JDK Installation & Maven Installation – Click Save

(JDK name and JAVA\_HOME path)

3. **New Item** – give Job Name – Maven project – check “Github” and give URL– In Source code management, again give Github URL.

**In Build** --

* Root POM (give path of the project) or POM.xml
* Goal and options (give some goals like clean install package etc.)

**In Project**, Select build now (It will run) and

**Email Notification:**

1. SMTP server and SMTP authentication
2. In Post build action – Email notification.

**Schedule the jobs:**

Configure – Build Triggers – Build periodically

43 15 \* \* \* (3pm 43 seconds)

H 22 \* \* \* (10 pm)

1. **Maven**

Maven is a build management tool for Java Frameworks which will allow you to create build (Build- is a set of task which you want to perform).

It automatically downloads the jars from the server, we can avoid manual downgrade and upgrade.

Why Maven?

* **Central repository to get dependencies (jars)**
* **Maintaining common structure across the organization (To maintain consistency)**
* **Flexibility in Integrating with CI tools like Jenkins (Continuous integration)**
* **Plugins for Test framework execution (TestNG)**

In-build commands – clean, install, test and so on.

Running Tests with Surefireplugin

Understanding POM.xml file dependencies..

Maven terminologies

1. **Artifact**: An artifact is a file, usually a JAR, that gets deployed to a Maven repository.
2. **GroupId:** groupId will identify your project uniquely across all projects,
3. archetype:generate; Generates a new project from an archetype

(When you add dependencies, maven will download the jars from the central servers and place it in the M2 repository)

1. Creating Maven Project with Terminologies

mvn archetype:generate -DgroupId=mavenjava -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

artifactId – project name groupId – package name

3. Integrate Maven with Eclipse -- mvn eclipse:eclipse

4. Then import the existing maven project in exclipse

5. Maven **Sure-fire plugin** (Used to execute all the test-cases in your maven src/test/java folder)

6. Clean, build, install, test, generate-source

7. Testng xml files configuration in POM file

================================= +++++++ + +++++++ ================================

**POM.xml (With TestNG)**

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.21.0</version>

<configuration>

<suiteXmlFiles>

<suiteXmlFile>testng.xml</suiteXmlFile>

</suiteXmlFiles>

</configuration>

</plugin>

Cucumber-core, cucumber-junit, cucumber-extentsreport

Maven-compiler-plugin, cucumber-java, cucumber—jvm

1. **What are different types of frameworks?**

A framework defines a set of rules or best practices which we can follow in a systematic way to achieve desired standards. **(How better you maintain your test-cases or data)**

Set of rules and guidelines which help us to write test cases in more efficient way.

1. All test cases in proper structure
2. Less code writing and no duplication (Code reuse)
3. Easy to maintain the script
4. Test data and config file should be separate from code
5. Proper log and good reporting with screenshot

**1. Data Driven Framework:**  
When the entire test data is generated from some external files like Excel, CSV, XML or some database table,

**2. Keyword Driven Framework:**  
When only the instructions and operations are written in a different file like an Excel worksheet.

**3. Hybrid Framework:**  
Combination of both the Data Driven framework and Keyword Driven framework is called Hybrid framework.

**a) Data Driven Framework**

* A framework which is driven by data
* It can be used when one Testcase has to execute with different set of data.

Example: Let’s say we have 20 credentials and we need to test our application with all credentials so will be writing only 1 test script and test will be passed with Excel sheet.

Create one class that will focus on DD part, (Test Application with multiple data) e repository to store data

**@Test(dataProvider="testdata")**

public void TestFireFox(String uname,String password1)

{

driver.findElement (By.*id*(“login")).click();

driver.findElement (By.*id*("email")).sendKeys(uname);

driver.findElement (By.*id*("pass")).sendKeys(password1);

}

**@DataProvider (name="testdata")**

public Object[][] TestDataFeed(){

try {

wb=Workbook.getWorkbook(new File("location of excel sheet/fbdata.xls"));

sh1= wb.getSheet(0);

numrow=  sh1.getRows(); // get number of rows so that we can run loop based on this

}

catch (Exception e){

e.printStackTrace();

}

Object [][] facebookdata=new Object[numrow][sh1.getColumns()];

for(int i=0;i<numrow;i++){ // This will run a loop and each iteration it will fetch new row

// Fetch first row username

facebookdata[i][0]=sh1.getCell(0,i).getContents();

facebookdata[i][1]=sh1.getCell(1,i).getContents(); // Fetch first row password

}

return facebookdata;

}

**b) Hybrid Framework**

**Project --**

**Src/main/java**

Pages –> Home page, Login Page

Resources –> Properties file (Config.properties; log4j.properties);

Drivers –> Browser Driver files

Utilities –> TestUtil.java (waits, actions, capturing screenshots, accessing excels, sending emails)

Listeners.java, Extent Report.java

Test data –> Controller.xlsx (First Name, Last Name, Email, Contact number, Address)

Screenshots -->

Results –> Reports (Test Reports)

Maven Dependency –> Jar Files

**Src/test/java**

TestCases – > HomeTest, LoginTest.java;

TestBase.java (loading the configurations from properties files, Initializing the WebDriver, Implicit Waits, Extent Reports and also to create the object of FileInputStream)

**How to explain Test Automation Framework to the interviewer**

We need to specify in and out of our Test Automation Framework such as **programming language**  used, **Type of framework** used, Test Base Class (Initializing WebDriver, Implicit Waits), How we separate Element locators and tests (Page Objects, Page Factory), Utility functions file, Property files, TestNG annotations, How we parameterize tests using Excel files, How we capture error screenshots, Generating reports(Extent Reports), Emailing reports, Version Control System used and Continues Integration Tool used.

**Language:**In our Selenium Project we use Java language.

**Type of Framework**:In our project, we are using [Data-driven Framework](https://www.softwaretestingmaterial.com/data-driven-framework-selenium-webdriver/) by using [Page Object Model design pattern](https://www.softwaretestingmaterial.com/page-object-model/) with Page Factory.

**POM:**We have maintained a class for every web page. Each web page has a separate class and that class holds the functionality and members of that web page. Separate classes for every individual test.

**Packages:**We have separate packages for *Pages* and *Tests*. All the *web page* related classes come under Pages package and all the *tests* related classes come under Tests package.

As per below standardised maven project, all the tests are kept in the ‘**src/test/java***‘*and remaining files (such as config.properties, element locators (POM classes), utility files, test data, etc.,) kept under ‘**src/main/java***‘*.

**Test Base Class:**Test Base class (TestBase.java) deals with all the common functions used by all the pages. This class is responsible for loading the configurations from **properties files, Initializing the WebDriver, Implicit Waits,** Extent Reports and also to create the object of FileInputStream which is responsible for *pointing towards the file from which the data should be read.*

**Utility Class (AKA Functions Class):**Utility class (TestUtil.java) stores and handles the functions (The code which is repetitive in nature such as **waits, actions, capturing screenshots, accessing excels, sending email** etc.,) which can be commonly used across the entire framework. The reason behind creating utility class is to achieve reusability. This class extends the TestBase class to inherit the properties of TestBase in TestUtil.

**Properties file:**This file (*config.properties*) stores the information that remains static throughout the framework such as browser specific information, application URL, screenshots path etc.

All the details which change as per the environment and authorization such as URL, Login Credentials are kept in the *config.properties* file. Keeping these details in a separate file makes easy to maintain

**Screenshots:**  Screenshots will be captured and stored in a separate folder and also the screenshots of failed test cases will be added in the extent reports**.**

**Test Data**: (*controller.xlsx*). By using *this* we pass test data and handle data driven testing. We use [Apache POI](https://www.softwaretestingmaterial.com/handling-excel-files-using-apache-poi/) to handle excel sheets. (First Name, Last Name, Email, Contact number, Address)

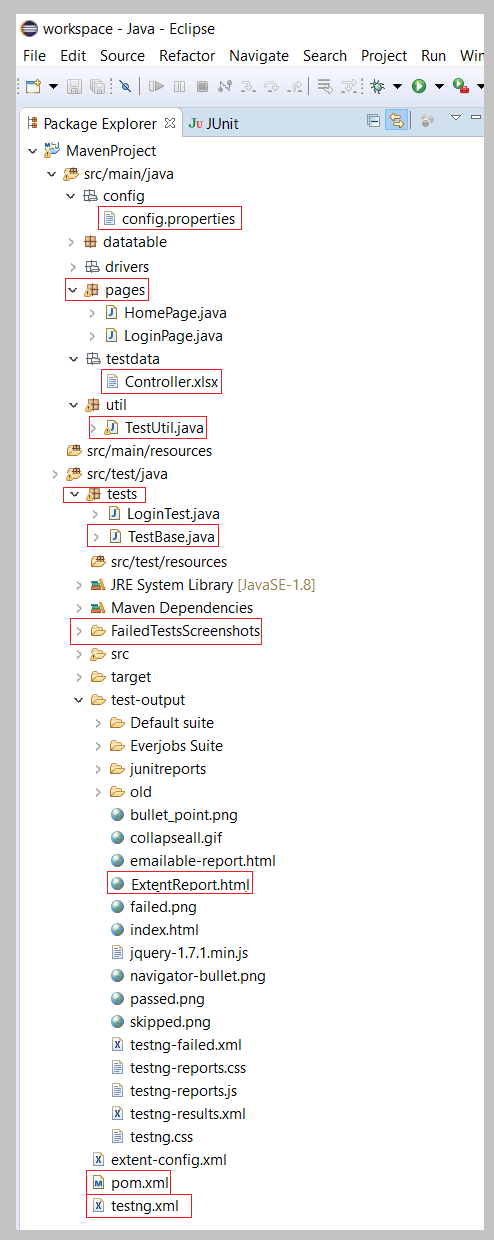
**TestNG**:Using TestNG for Assertions, Grouping and Parallel execution.

**Maven**:Using Maven for build, execution and dependency purpose. Integrating the TestNG dependency in POM.xml file and running this POM.xml file using Jenkins.

**Version Control Tool:** We use Git as a repository to store our test scripts.

**Jenkins:** By using Jenkins CI (Continuous Integration) Tool, we execute test cases on daily basis and also for nightly execution based on the schedule. Test Result will be sent to the peers using Jenkins.

**Extent Reports:**  It generates beautiful HTML reports. We use the extent reports for maintaining logs and also to include the screenshots of failed test cases in the Extent Report.

[](https://i1.wp.com/www.softwaretestingmaterial.com/wp-content/uploads/2017/07/framework-structure.png?ssl=1)

**c) BDD Framework**

>> Behaviour driven development is a process of developing software based on TDD (Test Driven Development) which focuses on behavioural specification of software units. **(Gherkin’s keyword)**

>> Test-cases should be defined in a Business level by following a Simple language.

**Project --**

**Src/main/java**

PageObjects –> Home page, Login Page

Drivers –> Browser Driver files

TestData –> Controller.xlsx (First Name, Last Name, Email, Contact number, Address)

Utilities –> TestUtil.java (waits, actions, capturing screenshots, accessing excels, sending emails)

TestBase.java (Initializing the WebDriver, Implicit wait and also to create the object of FileInputStream)

Listeners.java

ObjectRepository 🡪 ObjRep.xlsx

Results 🡪

**Src/test/java**

Runner– > TestRunner.java

Step Definition 🡪 StepDef.java

The 2 files required to execute a Cucumber test scenario are

1. Features (Feature File) --
2. Step Definition

**Feature File** (what Test cases are sufficient to certify Business Scenario working)

Given (what you need to have to perform action) -Pre requisites

When (performs action) - Action

Then (the desired outcome for the user) - Validation

**Example 1:**

***Feature: Sign up***

***Scenario: Successful sign up***  
***New users should get a confirmation email and be greeted***  
***personally by the site once signed in.***

***Given I have chosen to sign up***  
***When I sign up with valid details***  
***Then I should receive a confirmation email***  
***And I should see a personalized greeting message***

**Scenario outlines** are used when the same test has to be performed with different data set.

**Eg- 2 (Data driven: Multiple parameters)**

Feature: Test Facebook

Scenario Outline: Test login with valid credentials

**Given** Open the Mozilla Firefox and Go to login page

**When** I enter the valid “<username>” and “<password>”

**Then** The page should be successfully logged in

**Examples:**

|username |password |

[|zakirhus37@gmail.com](mailto:|zakirhus37@gmail.com)|pwd2 |

|zakirhus37@gmail.com|pwd3 |

|zakirhus37@gmail.com|pwd4 |

**Feature File – Have the Test Scenario**

Feature – R.click and run as cucumber feature  (step definition code will be created by cucumber framework on the CONSOLE as snippets)

**Step Definition:**

@When("^User enters \"(.\*)\" and \"(.\*)\"$")

public void user\_enters\_UserName\_and\_Password(String username, String password) throws Throwable {

             driver.findElement(By.id("log")).sendKeys(username);

             driver.findElement(By.id("pwd")).sendKeys(password);

             //driver.findElement(By.id("login")).click();

         }

Cucumber annotation is used to map with feature file.

Step definition is nothing but the steps you want to perform under the cucumber method

**TestRunner.java**

@RunWith(Cucumber.class)

@Cucumber.Options(features=(“src//test//java//features”), glue={"StepDefinition"},

                 plugin = {"pretty"},

                 monochrome = true

                 )

)

{"StepDefinition"} -- Package

features="Features – Folder

@RunWith annotation tells **JUnit that** tests should run using Cucumber class present in ‘Cucumber.api.junit‘ package.

 @CucumberOptions (are like property file) annotation tells Cucumber, where to look for feature files, what reporting system to use and some other things also.

**Cucumber finds the Step Definition file with the help of Glue code**

When you execute cucumber runner class, cucumber will start reading feature file steps. For example, when you execute @smokeTest, cucumber will read **Feature** step and **Given** a statement of **scenario**. As soon as cucumber finds Given the statement, same **Given** statement will be searched for your java files. If the same step is found in java file then cucumber executes the function specified for the same step otherwise cucumber will skip the step.

**Report in BDD**

### ***Monochrome Mode Reporting:***

*To filter this output in readable one which is****Monochrome***.

@CucumberOptions(monochrome = true );

### ***Usage Report***

*If we are more concerned about the time taken by each***Step Definition***, then we should use the***usage plugin***.*

@CucumberOptions( plugin = { “usage” })

***HTML, XML, JSON & TXT.****Cucumber frameworks generate very good and detailed reports, which can be shared with all stake holders.*

plugin = { "pretty", "json:target/cucumber-reports/Cucumber.json",

"junit:target/cucumber-reports/Cucumber.xml",

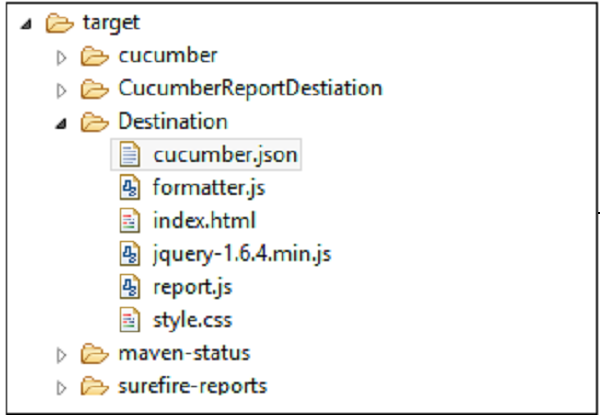
"html:target/cucumber-reports"}

Right Click on **TestRunner** class and Click **Run As**  > **JUnit**

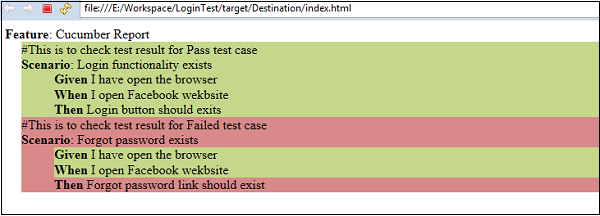
**@Cucumber.Options(format = {"pretty", "html:target/Destination"} )**

//Specifying pretty as a format option ensure that HTML report will be generated.

//When we specify html:target/Destination - It will generate the HTML report



The report will be there named as “Index.html”. Open Index.html with web browser.



It exactly highlights the color of failed scenario. Moreover, you will see highlight for failed step in that scenario. This makes the debugging very easy.

1. **Agile Testing:**

It is an iterative software development methodology where requirements keep changing as per the customer needs.  **Testing is done in parallel to the development of an iterative model**. Test team receives frequent code changes from the development team for testing an application.

**Principle:**

Highest priority is to satisfy the customer through early and continuous delivery of business valuable software

**What approach do you follow when requirements change continuously?**

The team should work closely with the Product Owner to understand the scope of requirement change and to negotiate to keep the requirement changes to a minimum or to adopt those changes in next sprint. Based on the requirement changes Test Team could update the Test Plan and Test Cases to achieve the deadlines. The team should understand the risk in the requirement change and prepare a contingency plan. It is a best practice not to go for the automation process until requirements are finalized.

The testing team works on **small features in Agile** whereas the test team works on a complete application after development in the traditional models.

**In Scrum, the project is divided into Sprints.** Each Sprint has a specified timeline (2 weeks to 1 month). This timeline will be agreed by a Scrum Team during the Sprint Planning Meeting. Here, User Stories are split into different modules.