Module: Selenium WebDriver

Trainer: Ankush Vankore

No of Sessions: 19

Contains

* WebDriver
  + Introduction
  + Installation
  + WebDriver Methods
    - get()
    - getTitle()
    - getCurrentUrl()
    - getPageSource()
  + Locators
    - Name
    - Id
    - ClassName
    - CssSelector
    - Xpath
    - LinkText
    - PartialLinkText
    - TagName
    - RelativeLocator
  + Handling different Controls
    - Text Box
    - Command Button
    - Radio button
    - Check Box
    - Dropdown List
    - List Box
    - Hyperlinks
    - FileUpload
  + Synchronization
    - Thread.Sleep()
    - ImplicitWait
    - ExplicitWait
    - FluentWait
    - PageLoadTimeout
  + Handling Tables
  + Handling Multiple Windows
  + Handling Alerts
  + JavascriptExecutor
    - Clicking
    - Scrolling
  + Taking Screenshot
  + Robot Class
  + Action Class
    - Click
    - Right Click
    - Double Click
    - Drag and Drop
* TestNG
  + Configuration
  + Single Test
  + Multiple Tests
  + Setting priority
  + Annotations
    - BeforeTest
    - AfterTest
    - BeforeMethod
    - AfterMethod
    - DataProvider
    - Parameters
  + Assertions
  + Parallel Execution
  + Reports
  + Liner Framework
  + Modular Framework
  + Keyword Driven Framework
  + Page Object Model (POM)
  + Data Driven Testing
    - Reading data from Excel file
    - Writing data to Excel file
* Maven & Cucumber
  + Configuration
  + Configuration of pom.xml file
  + Adding dependencies
  + Creating Feature file
  + Creating Step Definition
  + Creating Runner Class
  + Keywords
  + Hooks
  + Tags
  + Reports
  + Data driven testing via cucumber

Revision of Java

* Basics
  + Variables
  + Data types
  + Conditions
    - If, if-else, switch case
  + Loops
    - for loop
    - for each / enhanced for loop
  + Writing and calling Methods
  + Arrays
  + Creating class and Objects
  + Static Members and static methods
  + Collections
    - **List**
    - Set
    - Map
  + Exception Handling
    - throws keyword

Software Testing

Process of checking **C**orrectness, **C**ompleteness, **S**ecurity & **Q**uality of developed software application.

Process

* Creating Test Scenarios
* Creating Test cases
* Creating Test Data
  + ECP
  + BVA
  + EG
* Execution
* Report the bug / defect

Manual Testing: Hand-Eye-Brain

* Entering some data (in text box)
* Selecting the values (from list box, dropdown list, check box, radio buttons)
* Taking action (Clicking on button)
* Navigation (Clicking on a link)
* Mark the test case as Pass or Fail

**Automation Testing:**

Performing above actions with the help of a machine. A machine in this context is nothing but Automation Testing Tool.

Every automation tool is a software.

Instructions needs to be provided in specific programming language.

Advantages:

* Faster
* Time Saving
* Avoids repetitions
* Less human efforts
* Ensure Quality
* Less human errors
* Accuracy
* Saves money
* Regression testing
* 24 / 7 test possible
* Reusability
* Test report
* Productivity

Need

* Faster execution
* Accuracy
* Less human errors
* Compatibility Testing
* Easy reporting

When to automate?

* Stable requirements
* Repeated test scenarios
* Large amount of data to be tested
* Performance testing
* More accuracy
* Compatibility Testing
* CICD

Types of automation Testing

* Unit testing
  + JUnit
  + NUnit
* API Testing
  + Postman
  + RESTAPI
* GUI (Functional Testing)
  + Selenium WebDriver
  + Tosca
  + QTP
  + Cucumber
  + Playwright
* Mobile Testing
  + Appium

Process of Automation

* Planning
* Tool selection
  + Technology
  + Market presence
  + Ease of the tool
  + Cost of tool
  + Support availability
  + Automated Report
* Generate the script
* Create the test data
* Execution
* Report
* Maintance

**Selenium**

Suite / Bundle of test automation tools to test web / browser based (Web Sites) application.

**Components of Selenium**

1. Selenium IDE (Record and Play)
2. Selenium Grid (Parallel Execution)
3. Selenium RC – Deprecated
4. Selenium WebDriver

**Selenium WebDriver**

* Tool to test Web Based application
* An interface in Java
* An API

Selenium WebDriver Configuration

1. Create 2 folders (Except on C: )
   1. YourName\_WebDriverDemos
   2. WebDriver Jar Files

**Pre-Requisite for Selenium WebDriver**

1. Minimum Windows 10
2. Minimum Java 11 (Check on command prompt via java –version)
3. Any 1 updated Browser
4. Editor for scripting
   * Eclipse
   * Idea Intellij
5. Selenium Jar File
   * Launch selenium.dev site
   * Click on Downloads
   * Download Latest stable version (Latest stable version [4.30.0](https://github.com/SeleniumHQ/selenium/releases/download/selenium-4.30.0/selenium-server-4.30.0.jar))
   * Cut paste this file to 2nd folder we have created.

**Configuration of WebDriver with Eclipse**

* Launch Eclipse
* Select the 1st folder which you have created earlier as a workspace.
* Create a new Java Project
  + File 🡪 New 🡪 Java Project
  + While Creating make Create module-info.java file check box OFF.
  + Click on Finish
* Create a package inside this folder
* Create a class
* Right click on your project 🡪 Build Path 🡪 Configure Build Path 🡪 Libraries 🡪 Click on Class Path 🡪 Click on Add External Jars… 🡪 Open the 2nd folder created earlier and select the Jar file that we have downloaded. 🡪 Click on Apply and Close

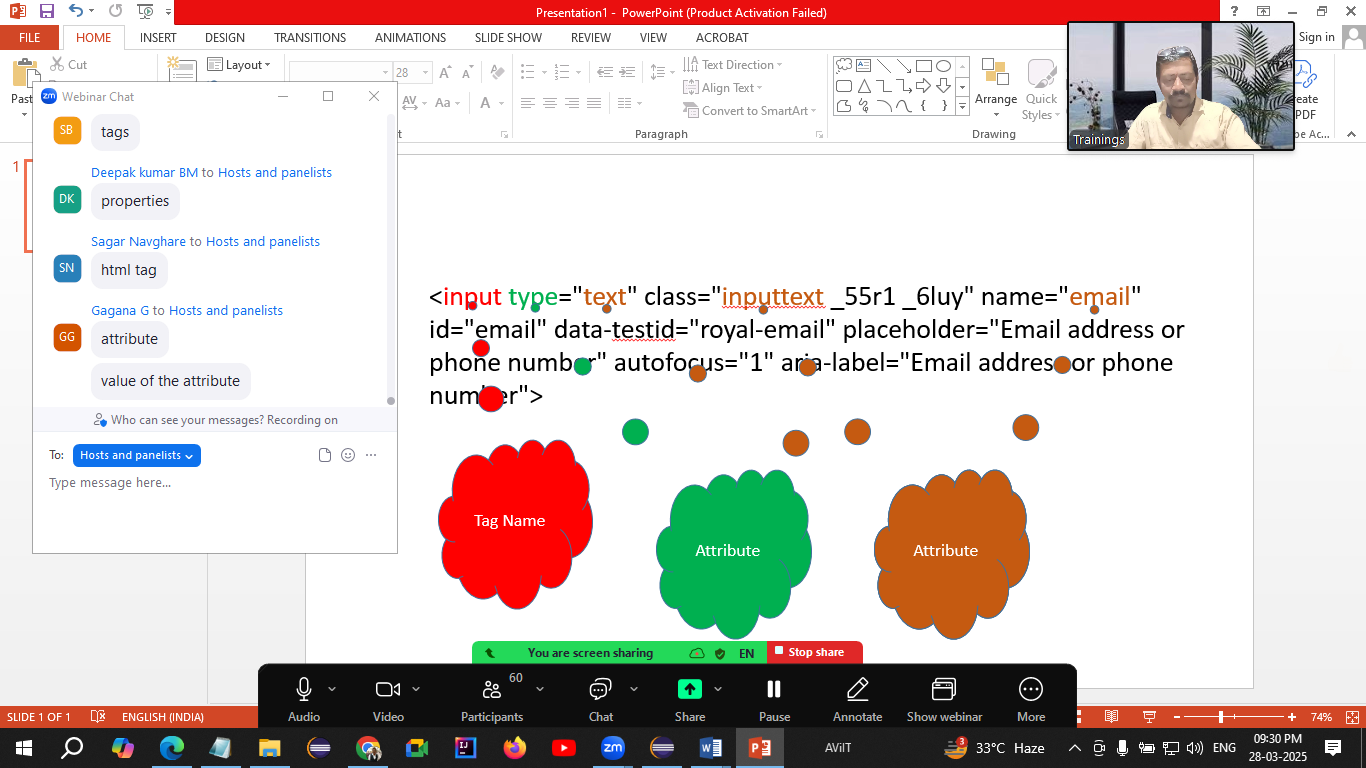
WebDriver Methods

1. Launching the browser 🡪 Create object of WebDriver interface.  
   This will launch the blank browser window
2. get() 🡪 Launch the specific website (URL)
3. close() 🡪 Close the current browser window which is launched by WebDriver object.
4. driver.manage().window().maximize() 🡪 Maximize the browser window.
5. getTitle() 🡪 Returns the title of the web page. (String)
6. getCurrentUrl() 🡪 Returns the URL of the web page. (String)
7. findElement() 🡪 Read a single control or WebElement from the page. Always locates the first occurrence. (WebElement)
8. findElements() 🡪 Returns multiple controls or WebElement from the page. (List<WebElement>)
9. getWindowHandles() 🡪 Returns ids of the windows those have been launched by WebDriver object. (Set<String>)
10. quit() 🡪 Will close all the windows those are opened by WebDriver object.

**Locators:**

This is the way to find out any control on the web page.

1. Name
2. Id
3. ClassName
4. CssSelector
5. XPath
6. LinkText
7. PartialLinkText
8. TagName
9. RelativeLocator



**WebElement**

* Every control (even text) is treated as WebElement in Selenium.
* Is an interface.

Methods of WebElement

* sendKeys() 🡪 Used to enter some text in the text box.   
  If already there is some text box then next text will be appended to the earlier text.
* click() 🡪 Used to click on any control.
* getText() 🡪 Return the text on the control. (String)
* isSelected() 🡪 Checks that whether the control (check box, radio button) is selected or not. (boolean)
* isDisplayed() 🡪 Checks that whether the control is visible or not. (boolean)
* isEnabled() 🡪 Checks that whether the control is enabled or disabled. (boolean)
* getAttribute() 🡪 Return the value of any attribute of the control. (String)
* clear() 🡪 Clears the text from text box.

**Common Exceptions in WebDriver**

1. InvalidArgumentException 🡪 Your URL is not in the correct format. URL should be Absolute (start with http / https)
2. SessionNotCreatedException 🡪 When the version of browser and selenium is mismatched.
3. NoSuchElementException 🡪 Selenium is not able to locate this control, possible reasons are
   1. The value of locator is wrong.
   2. The value of locator is dynamic.
   3. Synchronization issue.
   4. The control may be inside iframe.
4. InvalidSelectorException 🡪 The value of locator / selector is not in correct format.
5. SessionTimeOutException 🡪 If the specified URL is not getting loaded in 30 seconds. (ie. The default wait added for driver.get()).
6. ElementClickInterceptedException 🡪 If the control is behind of some another control. Or it is hidden by some another control.

**CssSelector Locator**

1. Using single Attribute  
   TagName[attribute=”value”]  
   input[data-testid=”royal-email”]
2. Using multiple Attributes  
   TagName[attribute1=”value”][attribute2=”value”]
3. Using special characters
   1. ^ 🡪 Starts with
   2. $ 🡪 Ends With
   3. \* 🡪 Contains text
   4. . 🡪 Class Name
   5. # 🡪 Id

For creating CssSelector in browser

1. Open the Inspection Window (DOM)
2. Click anywhere in this window
3. Hit Ctrl+F from the keyboard
4. Copy the attribute along with value
5. And then put the tag name and [ and paste the value what you copied
6. After hitting enter, the control should highlighted with yellow color and the control should show 1 of 1 next to the search text box

**XPath**

XML Path – Guiding selenium to reach out to any control on the web page

Pattankodoli Bus Stand 🡪 Take a Right turn 🡪 Hupare Nagar 🡪 Water Tank 🡪 Enter in lane no 9 🡪 1128

1. Absolute XPath  
   Starts with html
2. Relative XPath  
   Starts with //
   1. Using parent tag  
      //tagName[@attribute=”value”]/tagName
   2. Using exact control tag

**Handling drop down list.**

**Point to be noted:** If the control is having <select> tag then only it is treated as drop down list in Selenium.

1. Display selected country
2. Display total no of countries
3. Display list of all the countries
4. Select United Kingdom from the list
5. Display selected country

**Select** is the class which is used to handle dropdown list or list box.

Select class accepts WebElement as a parameter while creating object.

Methods of Select class

1. getFirstSelectedOption() 🡪 Return the element which is selected. (WebElement)
2. getOptions() 🡪 Return list of all the elements from the drop down list / list box. (List<WebElement>)
3. selectByVisibleText() 🡪 Selects the option from the drop down list using the text displayed on the option. Pass the **inner text** as a parameter.
4. selectByValue() 🡪 Selects the option from the drop down list by using the value of attribute **value.**
5. selectByIndex() 🡪 Selects the option from the drop down list by using its **zero based integer index**.
6. selectByContainsVisibleText() 🡪 Selects the option from the list by using its partial visible text.
7. getAllSelectedOptions() 🡪 Returns the list of all selected options specifically from List Box. (List<WebElement>)
8. isMultiple() 🡪 Checks that whether the control is drop down list or list box. If returns true then it is list box. If returns false then it is drop down list. (boolean)
9. deselectByVisibleText() 🡪
10. deselectByValue() 🡪
11. deselectByIndex() 🡪
12. deselectAll() 🡪

**Synchronization (Waits in Selenium)**

This is the process of adjusting **speed of tool** with speed of application.

1. Thread.sleep()  
   Forcefully stops the execution of the script for specified time.
   1. Waits mandatory for specified time duration.
   2. It is applicable to single statement only.
2. ImplicitWait
   1. It doesn’t take mandatory delay
   2. It is applicable throughout the script
3. ExplicitWait (WebDriverWait)
   1. It is applicable to the single statement only
   2. It doesn’t take mandatory delay
   3. We can handle the conditions like
      1. Element to be visible
      2. Element to be clickable
      3. Alert visible
4. FluentWait
   1. It is applicable to the single statement only
   2. It doesn’t take mandatory delay
   3. We can handle the conditions like
      1. Element to be visible
      2. Element to be clickable
   4. Alert visible
   5. You can handle exception as well.  
      Parameters you need to add  
      w - withTimeout  
      i – ignoring   
      p –pollingEvery  
      u – until
5. PageLoadTimeout  
   You can add some more wait for loading the page to avoid SessionTimeoutException.

**Handling Table**

1. Display all the headers
2. Display total no of rows
3. Display any row randomly

**Handling Alert**

If you are not able to inspect the window, then it is an Alert. (JavaScript Alert).

**Alert** interface is used to handle any alert in Selenium.

Methods

1. driver.switchTo().alert() 🡪 Will switch on the alert / it will take you on the alert.
2. getText() 🡪 Return the text on alert. (String)
3. accept() 🡪 Click on Ok button on alert.
4. dismiss() 🡪 Click on Cancel button on alert.

**JavascriptExecutor Interface**

* Used to avoid ElementClickInterceptedException
* Used for scrolling a page
* You can click on the control even if the control is behind or hidden by some another control.

**Mouse Actions**

* Right Click
* Left Click
* Hover
* Double Click
* Drag and Drop

**Actions Class** 🡪 Is used to handle above mouse actions.

**TestNG** (Test Next Generation)

It is test automation framework.

Framework – Set of rules and guidelines to make your automation testing more easier.

**Advantages**

1. You can create multiple tests in one script
2. You can set priorities for your tests
3. Generates reports
   1. Normal Report
   2. HTML Report
4. Uses annotations (Annotations are the keywords those give additional information about the method or class)
   1. @Test
   2. @BeforeTest
   3. @AfterTest
   4. @BeforeMethod
   5. @AfterMethod
   6. @BeforeClass
   7. @AfterClass
   8. @DataProvider
   9. @Parameters
5. Execute / Skip single or multiple test cases
6. Implement frameworks
   1. Linear Framework
   2. Modular Framework
   3. Key driven Framework
   4. Page Object Model
   5. Data Driven Framework
   6. Hybrid Framework
7. Parameterization
8. Parallel Execution

**Annotations**

1. @Test 🡪 This is the method which is treated as Test Case.
2. @BeforeTest 🡪 This is the method which get executed **ONLY ONCE BEFORE EXECUTING 1ST TEST CASE.**This method is generally used to launch the browser, initialize the objects those needs to use in test.
3. @AfterTest 🡪 This is the method which get executed **ONLY ONCE AFTER EXECUTING LAST TEST CASE.**This method is generally used to close the browser and close the resources those you have used in the test.
4. @BeforeMethod 🡪 This is the method which will get executed **BEFORE EVERY TEST CASE**
5. @AfterMethod 🡪 This is the method which will get executed **AFTER EVERY TEST CASE**
6. @DataProvider 🡪 This is the method that pass the data to the test case. A single test will get executed for that many number of times.

BeforeTest

BeforeMethod

Test 1

AfterMethod

BeforeMethod

Test 2

AfterMethod

BeforeMethod

Test 3

AfterMethod

BeforeMethod

Test 4

AfterMethod

AfterTest

**Points to be noted about Before & After methods**

1. The can appear anywhere in the script. Sequence does not matter.
2. The need not to be in pair.

**Data Driven Testing**

Executing single test case with multiple data set. Get the data from any one of following

1. Two D Array (@DataProvider)
2. XML File
3. Excel File

**Assertion**

Assertion is the way to mark any test case as pass or fail.

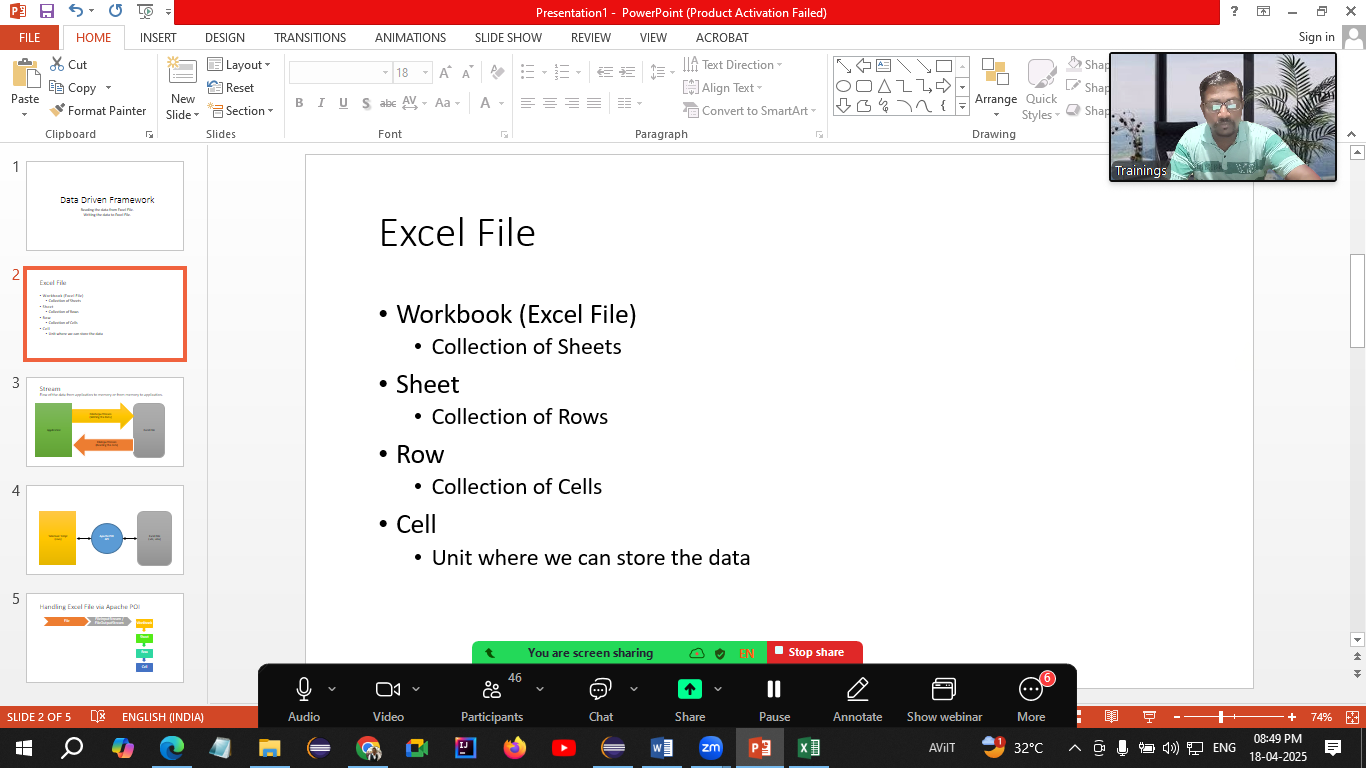
Modular Framework

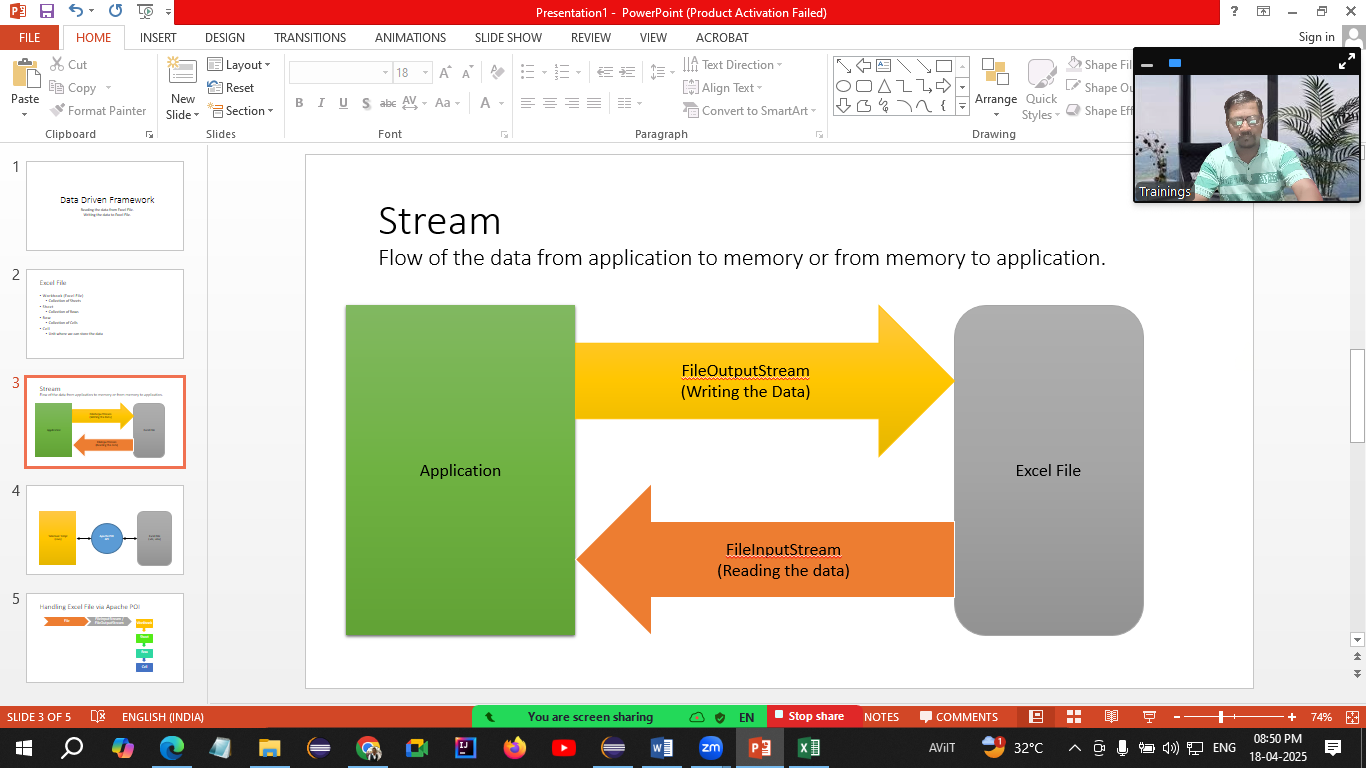
* Execute / Skip Single / multiple test case(s)
* Execute your script via XML file

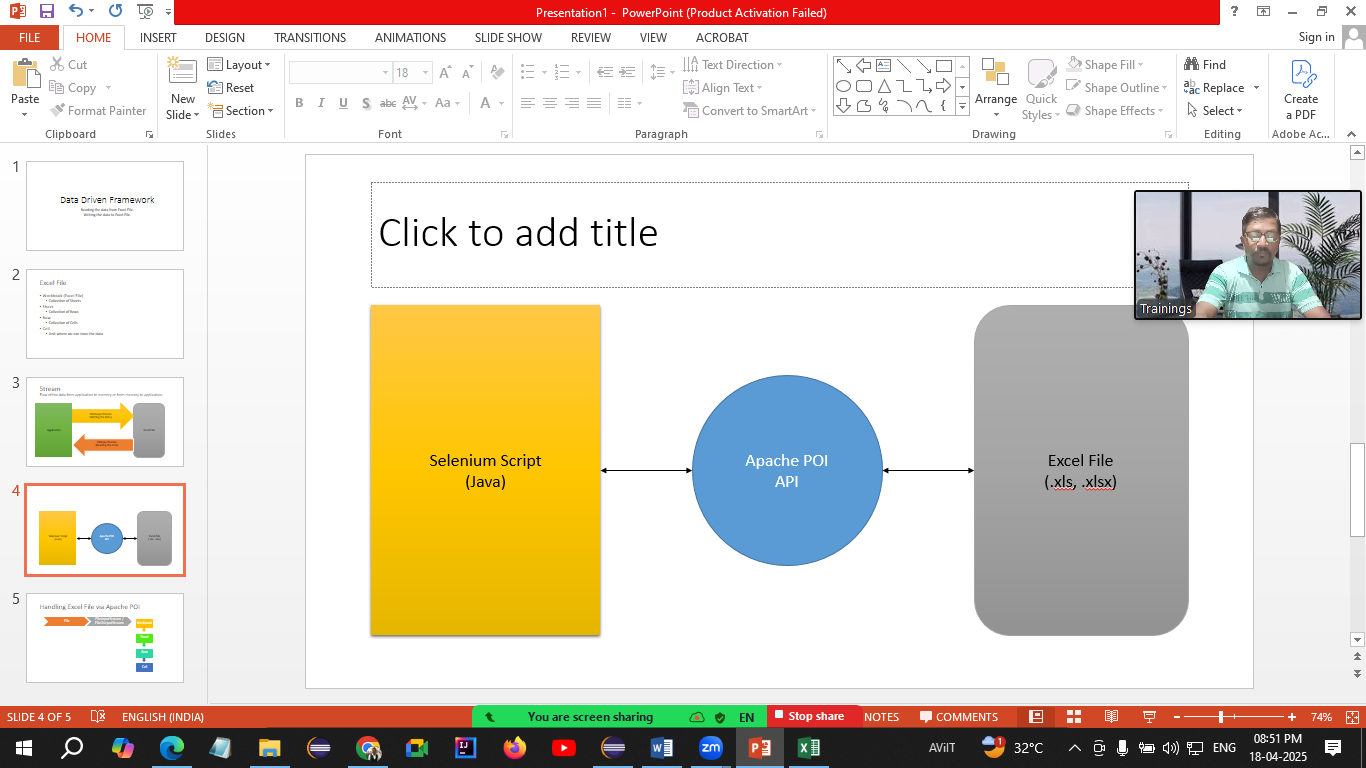
While creating XML file please note that

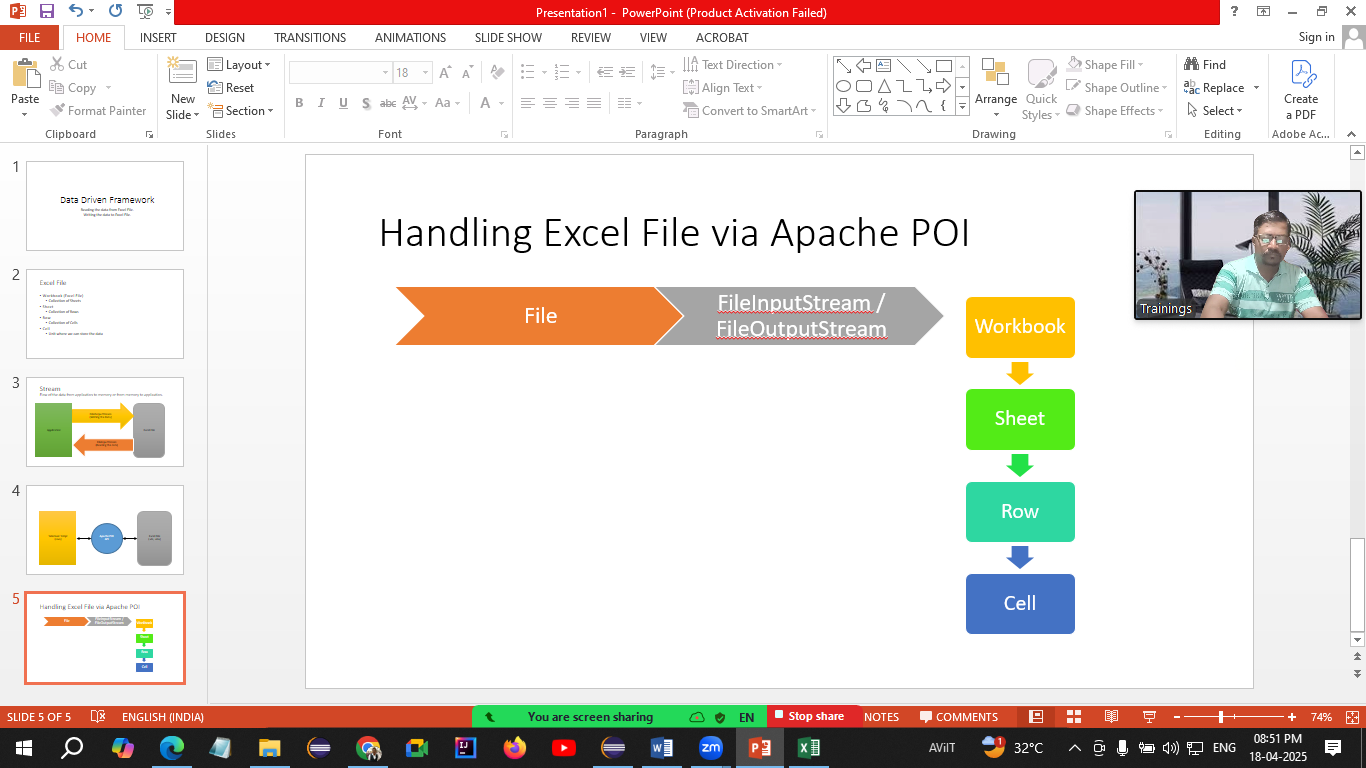
1. All the tags are case sensitive
2. You cannot alter the sequence of any tag

**Data Driven Framework**









**Configuration of Apache POI to Project**

1. Download apache poi <https://archive.apache.org/dist/poi/release/bin/>
2. Copy paste the .zip file to 2nd folder which we have created in 2nd lecture
3. Observe the folder structure
   1. 8 jar files in root folder
   2. 6 jar files in lib folder
   3. 7 jar files in ooxml-lib folder
   4. Total 21 files
4. In Eclipse
   1. Right click on project
   2. Click on Build Path
   3. Click Configure Build path
   4. Click on Libraries
   5. Click on ClassPath
   6. Click on Add External Jars…
   7. Add all 21 files one by one
   8. Click on Apply and close

**Maven**

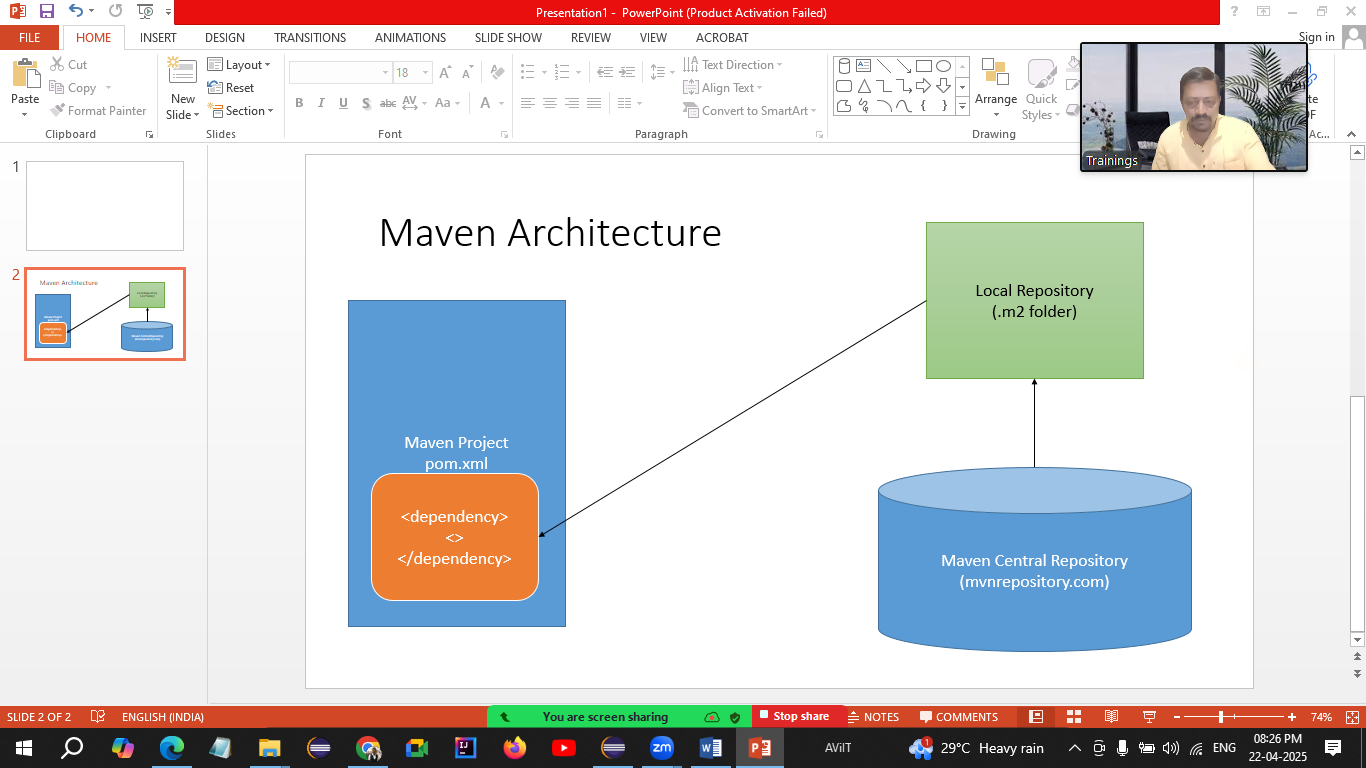
* Build Management Tool used by both developer and tester.
* Apache Product
* Open source
* Easy configuration through pom.xml file. (pom 🡪 Project Object Model)
* Uses dependencies

Creating Maven Project (1st way – With Archetype)

* File 🡪 New 🡪 Maven Project
* Select 3rd Checkbox (Add project to Working set) 🡪 Next
* Add maven-archetype-quickstart as a Filter 🡪 Select any latest version 🡪 Next
* Give GroupId and ArtifactId (Both should be same)
* Click on Finish

Creating Maven Project (2nd way – Without Archetype)

* File 🡪 New 🡪 Maven Project
* Select 1st Checkbox (Create a Simple Project) 🡪 Next
* Give GroupId and ArtifactId (Both should be same)
* Click on Finish



**Cucumber**

BDD Framework (Behavior Driven Development)

* BA, QA & Developer will gather the requirements
* Then requirements will be shared with QA team
* QA will create **Feature File (1st Component of BDD)**
* This feature file will be shared with
  + Developer – To implement the functionality
  + Manual Tester – To develop manual test cases
  + Automation Tester – To develop automation script
* When automation tester will execute feature file you will get **Step Definition / Glue Code (2nd Component of BDD)**
* Then **Runner class (3rd Component of BDD)** will be created.

Creating Cucumber Project (1st Way)

1. File 🡪 New 🡪 Maven Project
2. Select 3rd Checkbox (Add project to Working set) 🡪 Next
3. Add filter as **io.cucumber** (Archetype)
4. Select the latest version and click on Next
5. Give GroupId and ArctifactId (Both should be same) 🡪 Next

Creating Cucumber Project (2st Way)

1. File 🡪 New 🡪 Maven Project
2. Select 1st Checkbox (Create Simple Project) 🡪 Next
3. Give GroupId and ArctifactId (Both should be same) 🡪 Next

Configuration in Cucumber project

1. Open <https://github.com/cucumber/cucumber-java-skeleton/commit/d7249b50c570816eba27ce94557e1de7e9b0f97>
2. Then Click on Expand All button
3. Copy the code from Line no 11 <properties> till line no 41 </dependencies>
4. Open pom.xml file from your cucumber project.
5. Delete the code from <properties> till </dependencies>
6. Paste the code which you have copied.
7. Delete following lines
   1. <cucumber.version>6.8.2</cucumber.version>
   2. <maven.compiler.version>3.8.1</maven.compiler.version>
   3. <maven.surefire.version>2.22.2</maven.surefire.version>
8. Change the java version to 11 in following line <java.version>1.8</java.version>

**Components of Cucumber**

1. **Feature file**
   1. Collection of Test scenarios
   2. Having extension **.feature**
   3. Created using **Gherkin language** (Same as English)
   4. Contains some keywords
      1. Feature: 🡪 Requirements
      2. Scenario: 🡪 Test Objective
      3. Given 🡪 Pre-Requisite
      4. When 🡪 Steps
      5. And 🡪 Combine multiple steps
      6. Then 🡪 Expected Result
      7. Background: 🡪 For multiple Given statements
      8. Examples: 🡪 Used in data driven testing
      9. Scenario Outline: 🡪 Used in data driven testing
2. **Step Definition / Glue Code**
   1. Normal Class
   2. Contains Selenium Script
   3. Contains Cucumber annotation
3. **Runner Class**
   1. Normal java class
   2. Used for execution of Step Definition

Feature File

**Requirement – Validate Google Title**

1. Launch Google.com
2. Read the title
3. Title should be Google

Feature: Title of google

Scenario: To validate title of Google

Given Open Google

When Read the title

Then Title should be Google

**Requirement – Validate login functionality on Facebook**

1. Launch Facebook login page
2. Enter valid user name
3. Enter valid password
4. Click on Login button
5. Home page should display

Feature: Login functionality on Facebook with valid data

Scenario: To validate login functionality on Facebook with valid data

Given Launch Facebook login page

When I enter valid user name

When I enter valid password

And I Click on Login button

Then Home page should display

**Tags in Cucumber**

* Tags are used to execute / skip single or multiple scenarios
* Tags are created inside .feature file
* Tags are created using @ character
* Execution Criteria needs to be specified in Runner class

**Hooks in Cucumber**

* Hooks are the methods those get executed before and after every test scenario.
* Hooks are never the part of feature file
* There are two hooks
  + @Before
  + @After