Software Testing:

Process of checking for CORRECTNESS, COMPLETENESS, SECURITY & QUAILITY of developed software application. (CCSQ)

TC

**Objective**

To validate login functionality when valid data is provided.

**Pre-Requisite**

Login page should display & test data should ready

**Steps**

1. Enter valid user name
2. Enter valid password
3. Click on Login button

**Expected Result**

User should log in

Home page should display

**Automation Testing:**

Testing the application with the help of tools. (Automation testing Tools)

**Advantages:**

1. Time saving
2. Faster execution
3. Accuracy
4. Less manual work
5. Easy for the repetitions
6. Cost Saving
7. Improve the quality
8. Improve efficiency
9. Easy report management
10. Reduces the errors
11. Better allocation of resources
12. Less infra
13. Regression testing very easy

**When to automate:**

1. Build is stable
2. Repeated test cases
3. Huge amount of data
4. Compatibility Testing

**Tool Selection Criteria**

1. Tool support for the application
2. Availability of the tool
3. Market
4. Tester availability
5. Budget
6. Support availability for the tool
7. Reporting facility

**Process of automation**

1. Planning
2. Tool selection
3. Create the test script
4. Create the test data
5. Execute the test case (Script)
6. Report
7. Maintance

**Types of Automation Testing**

1. UNIT
2. GUI (Graphical User Interface)
3. API (Application Programming Interface)

**Selenium**

Bundle / Suite of automation testing tools.

**Components of Selenium**

1. Selenium IDE  
   Record and Playback
2. Selenium Grid  
   Parallel Execution on multiple browser and platforms
3. Selenium RC (Remote Control)  
   Create a script 🡪 Pass to RC Server 🡪 RC Server will take care of execution
4. Selenium WebDriver  
   In replacement of RC

**Selenium WebDriver**

Browser

(Chrome, FF, Edge, Safari)

Script

(Java, C#, Python)

1. Is automation testing tool for testing WEB (BROWSER) BASED APPLICATIONS
2. It is an API for communication with browser
3. It is an interface in Java.

**Pre-Requisite for Selenium WebDriver**

1. Minimum Windows 10
2. Java – Minimum 11 (java –version [on command prompt])
3. Editor (Java – IDE)
   1. Eclipse
   2. Intellij IDEA
4. Any one **updated** (latest) Browser
5. Jar File (API) (<https://www.selenium.dev/> 🡪 downloads 🡪 Download Latest Stable Version)

**Configuration:**

Create 2 folders (Better on D:)

1. **Selenium Jar File** (For storing all the jar files required for the Selenium)
2. **YourName\_SeleniumDemos** (For storing our Selenium Demos)

Open Eclipse 🡪 Select (Browse) a folder as Workspace 🡪 Launch

File 🡪 New 🡪 Java Project 🡪

Give Name 🡪 Select Java Version (Java SE 11) 🡪 Uncheck Create module-info.java file checkbox 🡪 Click on Finish

Right Click on Project 🡪 New 🡪 Package

Create a class in this package

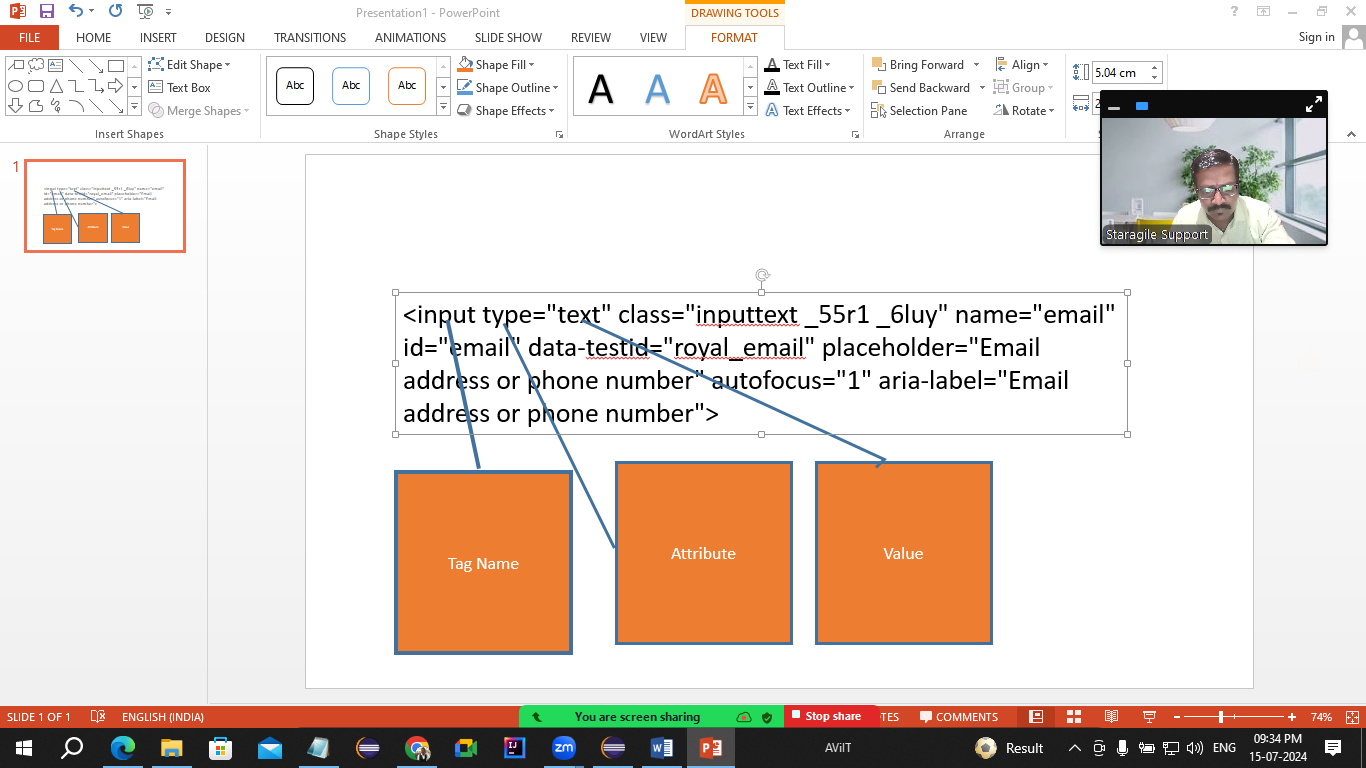
Right Click on Project 🡪 Build Path 🡪 Configure Build Path 🡪 Click on Libraries Tab 🡪 Click on ClassPath 🡪 Click on Add External JARs… 🡪 Select the Selenium Jar file which you have downloaded 🡪 Click on Open 🡪 Click on Apply and Close

**Selenium WebDriver**

1. Launch the browser 🡪 Create object of WebDriver interface
2. **get()** 🡪 Open the url
3. **driver.manage().window().maximize()** 🡪 Maximize the browser window
4. **close()** 🡪 Close the browser window which is opened by WebDriver object.
5. **getTitle()** 🡪 Return the title of the page. (String)
6. **getCurrentUrl()** 🡪 Return URL of the current page (String)
7. **getPageSource()** 🡪 Return rendered HTML (html code) of the page.
8. **findElement() 🡪** Read / Find a control (WebElement) on the page. Locates first occurrence. (WebElement)
9. **findElements() 🡪** Read / find multiple controls on the page. (List<WebElement>)

**Locator:** is the way of finding the control on the page.

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



**WebElement**

* Every control (text box, button, link, radio button, check box, dropdown list) is treated as WebElement in Selenium.
* **WebElement is an interface.**
* Which refers to the control on the page.

**Methods of WebElement**

1. **sendKeys()** 🡪 Will enter / type some text in the text box. It will append the text.
2. **click() 🡪** Will click on the control
3. **submit() 🡪** Will click on the button which is having type=”submit” attribute
4. **getText() 🡪** Will return the text on any control. (String)
5. **isSelected() 🡪** Verify that the control (Checkbox & Radio button) is selected or not. (Boolean)
6. **isEnabled() 🡪** Verify that the control is enabled or disabled. (boolean)
7. **isDisplayed() 🡪** Verify that the control is visible or not (boolean)
8. **getAttribute() 🡪** Returns the value of any attribute of the control. (String)
9. **getWindowHandles() 🡪** Return id’s of all the windows those are opened by WebDriver (Set<String>)
10. **quit() 🡪** Will close all the windows those are opened by WebDriver.

**Common Exceptions in WebDriver**

1. **SessionNotCreatedException** 🡪 If your selenium WebDriver and browser versions are not matching
2. **InvalidArgumentException** 🡪 The URL is not in the correct format. (URL should be absolute)
3. **NoSuchElementExceptio** 🡪 Selenium is unable to locate this control
   1. The value of locator is wrong.
   2. Locator value may be dynamic. (It is changing for every request)
   3. Synchronization issue.
   4. Control may be inside the **iframe**
4. **InvalidSelectorException** 🡪 The value of locator is not in the correct format.
5. **SessionTimeoutException 🡪** driver.get() waits for maximum 30 seconds to load the page. If page is not getting loaded in this 30 seconds then selenium will throw this exception.
6. **NoAlertPresentException 🡪** you are trying to switch some alert, but no alert is there.

**CssSelector**

(Cascading Style Sheet)

1. Using single attribute  
   tagName[attaibute = “value”  
   eg.   
   input[placeholder="Email address or phone number"]
2. Using multiple attributes  
   tagName[attribute1 = “value”][attribute2 = “value”]  
   input[placeholder="Password"][name="pass"]
3. Using special characters
   1. ^ 🡪 Starts With  
      tagName[attribute^=”value”]  
      button[id^="u\_0\_5"]
   2. $ 🡪 Ends With  
      tagName[attribute$=”value”]  
      input[data-testid$="email"]
   3. \* 🡪 Contains  
      tagName[attribute\*=”value”]

<a>Gmail</a>

**XPath**

Stands for XML Path

Pattankodoli Bus Stand 🡪 Take Right turn 🡪 Water Tank 🡪 Hupare Nagar Lane No 9 🡪 House No 1128

Types

* Absolute XPath  
  Starts with html
* Relative XPath  
  Starts with //

**Handling Drop Down List & List Box**

Select class is used to handle drop down list & list box

**Please note: if the control is having *select* tag, then only we can handle with help of Select class.**

Methods:

* getFirstSelectedOption() 🡪 Return the selected option from the drop down list. (WebElement)
* getOptions() 🡪 Returns list of all the options from drop down list. (List<WebElement>)
* selectByVisibleText() 🡪 It will select the option (Element) from the list using the text on the element. (Need to pass string value)
* selectByValue() 🡪 It will select the option (Element) from the list using the value of element. (Need to pass String value)
* selectByIndex() 🡪 It will select the option (Element) from the list using the index (zero based) of element. (Need to pass int value)
* getAllSelectedOptions() 🡪 Will return list of all selected elements from the LIST BOX. (List<WebElement>)
* isMultiple() 🡪 It will check whether we can select multiple options / whether it is list box or not. (boolean)
* deSelectByIndex()
* deSelectByValue()
* deSelectByVisibleText() 🡪 Will de select the element / option
* deSelectAll() 🡪 Will deselect all options

**Synchronization (Waits in Selenium)**

Process of adjusting speed of tool with speed of application.

Methods of Synchronization

1. Thread.sleep() 🡪 Pause the execution of script for specified number of time
   1. It is applicable to single statement only.
   2. It takes mandatory delay.
2. ImplicitWait (ImplicitlyWait)
   1. It is applicable throughout a script
   2. It doesn’t take mandatory delay.
3. ExplicitWait (WebDriverWait)
   1. It doesn’t take mandatory delay.
   2. It is applicable to single statement only.
4. FluentWait
   1. It doesn’t take mandatory delay.
   2. It is applicable to single statement only.
   3. We can manage the Exception as well  
        
      w 🡪 withTimeout  
      i 🡪 ignoring   
      p 🡪 pollingEvery  
      u 🡪 until
5. pageLoadTimeout 🡪 driver.get() waits for maximum 30 seconds to load the page. If page is not getting loaded in this 30 seconds then selenium will throw SessionTimeoutException exception. To avoid SessionTimeoutException and giving some additional time for loading the page, you can use pageLoadTimeOut().

**Handling Javascript Alert**

(If the message is not inspectable. [We are not able to inspect])

**Alert 🡪** is an interface to handle alerts in selenium.

Methods

1. **switchTo() 🡪** It will switch to the alert.
2. **getText() 🡪** Return the text on the alert.
3. **accept() 🡪** It will click on Ok button.
4. **dismiss() 🡪** It will click on Cancel button.
5. **sendKeys() 🡪** Will enter some text on Alert (Prompt Box)

**Mouse Actions:**

Actions class is used to handle the mouse action like

1. Hover
2. Right Click
3. Left Click
4. Double Click
5. Drag and Drop

TestNG: Testing framework

NG – Next Generation

Testing framework: Set of guidelines and rules for executing test cases

Inspired by JUnit & NUnit

Advantages:

1. Combine multiple tests together.
2. Uses annotations.
3. Allows setting the priorities for test cases.
4. Get automated report
   1. Normal report
   2. HTML Report
5. Skip / Execute single or multiple tests.
6. Parallel execution.
7. Allows executing tests via group.
8. Implement different types of testing framework like
   1. Linear framework
   2. Modular framework
   3. Keyword driven framework
   4. Page object model
   5. Data driven framework

Annotations

1. @Test 🡪 Actually your test case
2. @BeforeTest 🡪 This is the method which get executed ONLY ONCE BEFORE EXECUTING 1ST TEST CASE
3. @AfterTest 🡪 This is the method which get executed ONLY ONCE AFTER EXECUTING LAST TEST CASE
4. @BeforeMethod 🡪 This is the method which get executed BEFORE EXECUTING EVERY TEST CASE.
5. @AfterMethod 🡪 This is the method that get executed AFTER EXECUTING EVERY TEST CASE
6. @DataProvider 🡪 This is the method which sends the data to Test case
7. @Parameters 🡪 This will read the parameters from XML file

**Points to be noted**

1. **Then need not to be in pair**
2. **They need not to be in sequence.**

BeforeTest

BeforeMethod

Test1

AfterMethod

BeforeMethod

Test2

AfterMethod

BeforeMethod

Test3

AfterMethod

BeforeMethod

Test4

AfterMethod

AfterTest

**Data Driven Testing:**

Executing single test case with multiple data set.

1. Arrays
2. XML File
3. Excel File

**Modular Framework:**

* Skip / Execute single or multiple tests
* Execute tests via XML File

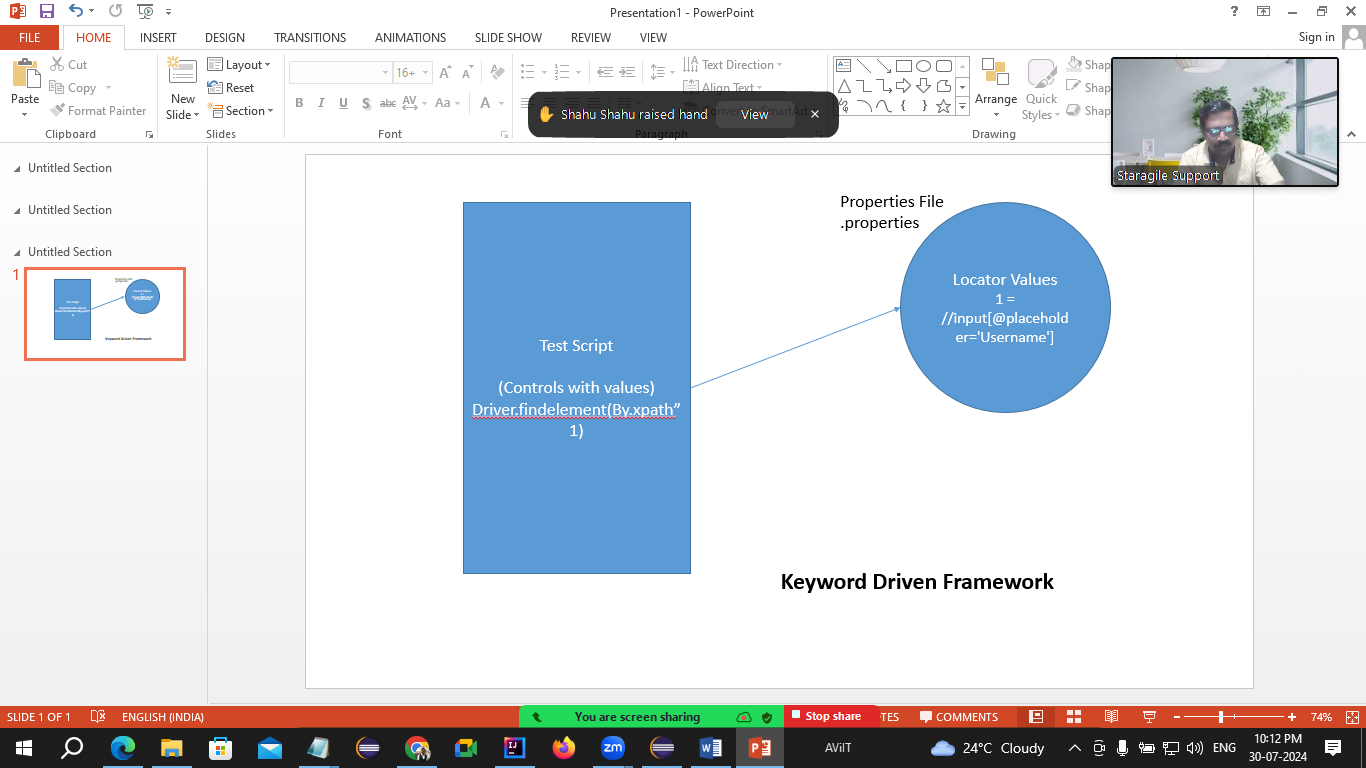
**Points to be noted while creating XML file**

* All the tags are case sensitive
* Tags needs to be in the proper sequence
* All the tags are pre-defined.

**Keyword Driven Framework**

Storing the values (especially locators) in the file and read those values in the test case

1. .properties
2. .xlsx (Excel file)

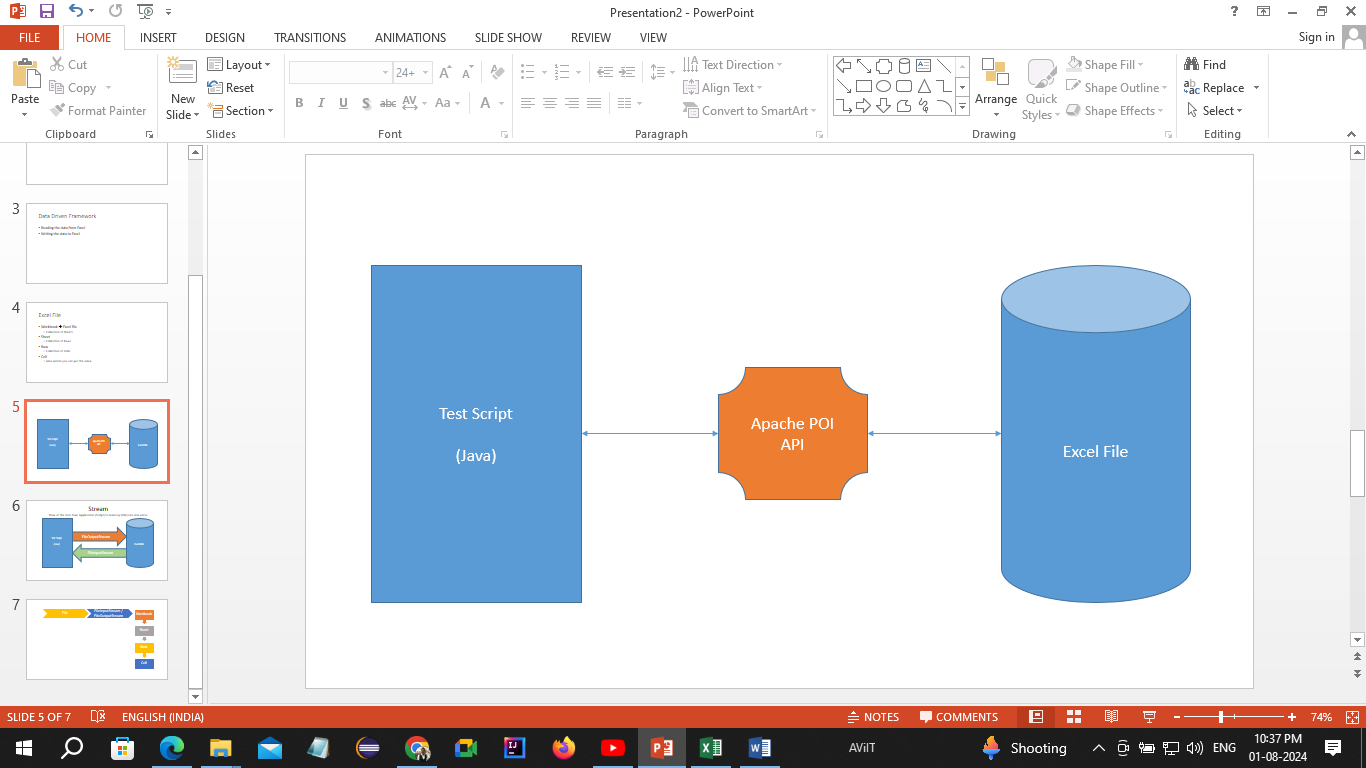


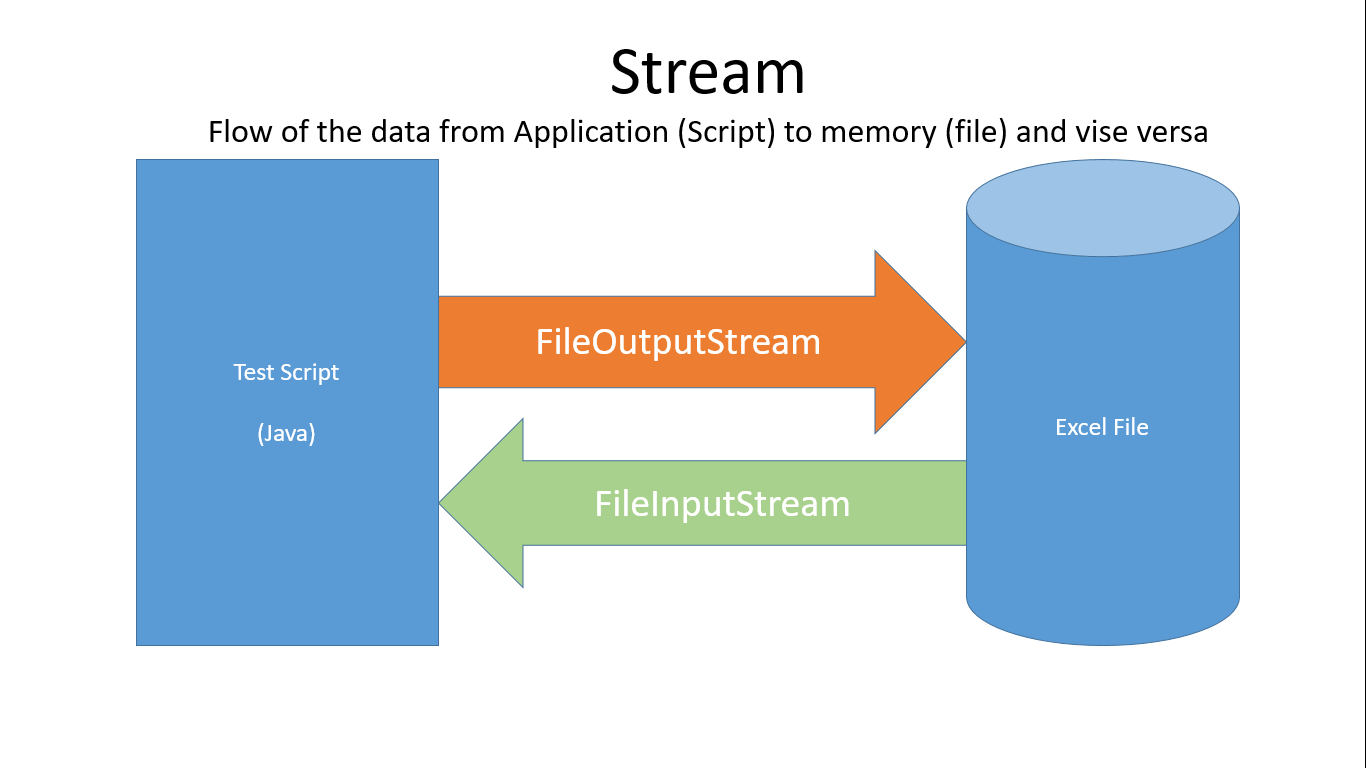
Frameworks

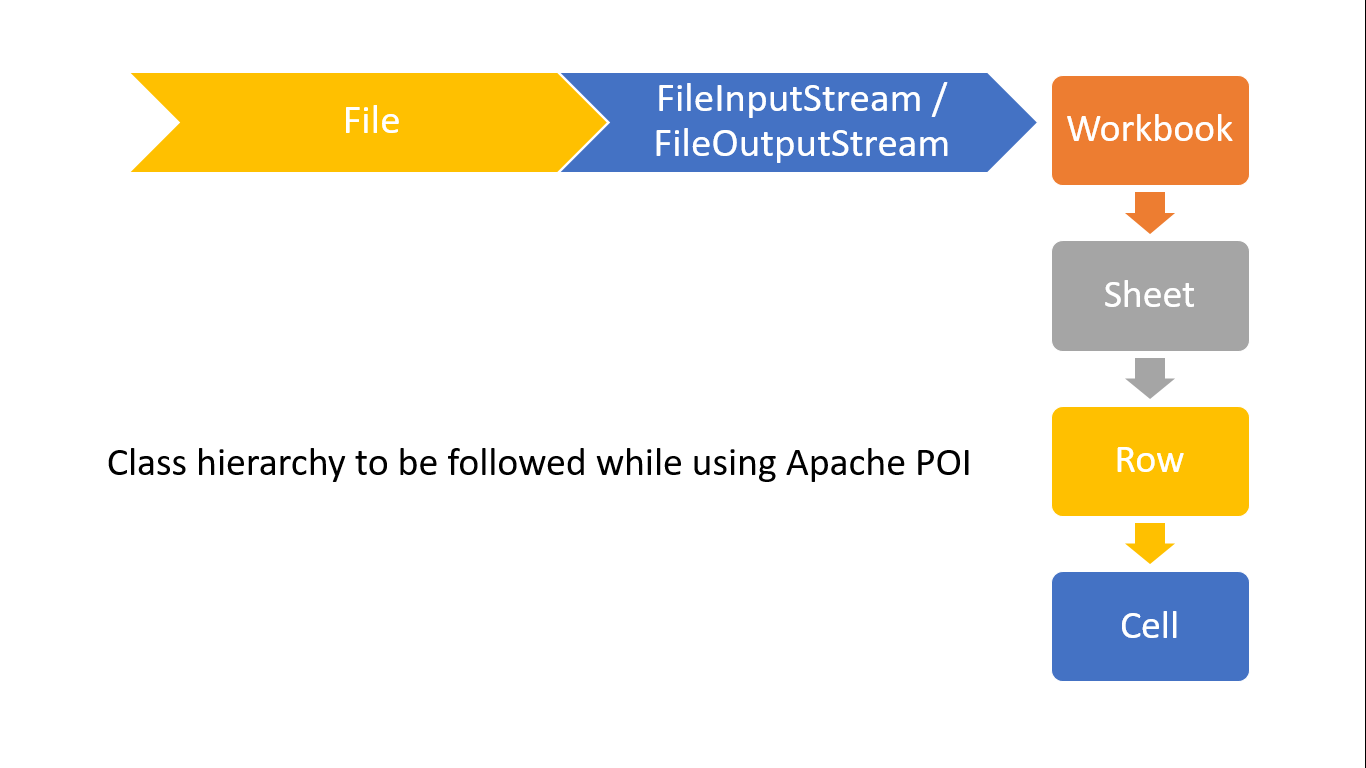
1. Linear Framework: Executing the test cases in some sequence. (priority)
2. Modular Framework: Executing / Skipping the test cases via XML file
3. Keyword Driven Framework: Storing locators of control and some data in .properties file and using it in script
4. Page Object Model: Creating 2 classes. 1 for WebElements, WebDriver code, logic and 2nd for test execution.
5. Data Driven Framework: Reading / writing data from Excel file.

Excel File

* Workbook 🡪 Excel file
  + Collection of Sheets
* Sheet
  + Collection of Rows
* Row
  + Collection of Cells
* Cell
  + Area where you can put the value







MAVEN

* Opensource build management tool
* Product of Apache
* Uses pom.xml file (Project Object Model)
  + Configuration of your maven project
* Uses dependencies.

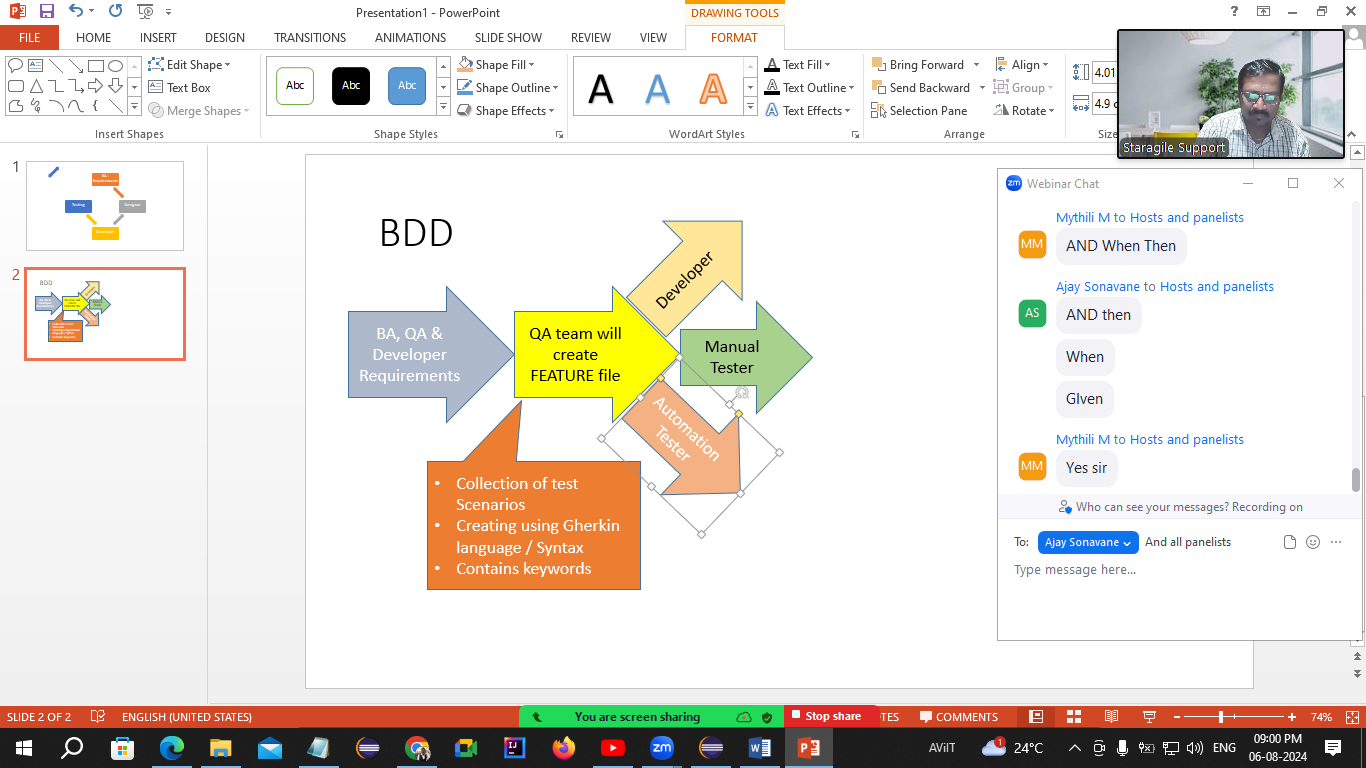
Creating Maven project

File – New Project – Maven – Maven Project – select 3rd check box (Add project to working set) – add filter **maven – archetype – quickstart** – give groupId & artifactId – finish

Update the java version

**Dependency:**

* Code of 4 – 5 lines
* Replacement for jar files. (No need to add any jar file)
* Will automatically download the required jar files and those jar files will be configured to the project automatically



Cucumber: Open source framework to implement BDD

Components

1. Feature file
   1. Collection of Scenarios
   2. Created using gherkin language / Syntax
   3. Contains keyword
2. Step Definition / Glue Code (This will get generated after executing .feature file)
   1. Is a normal class
   2. This is the automation script
   3. Uses JUnit annotations
3. Runner Class
   1. Normal class
   2. This class is used to execute your script
   3. Using JUnit annotations

Keyword in feature file

1. Feature: 🡪 Requirement
2. Scenario: 🡪 Test scenario to be tested
3. Given 🡪 Pre-Requisite
4. When 🡪 Steps to be performed
5. And / But 🡪 To combine multiple steps
6. Then 🡪 Expected Result
7. Background: 🡪 Common steps (especially for given)
8. Examples: 🡪 Specifically used in Data Driven Framework
9. Scenario Outline: 🡪 If you are passing the data through Examples then you have to use Scenario Outline instead of Scenario

Feature file

Validate Google title

1. Open Google
2. Read title
3. Validate that title is Google

Feature file

Feature: Test Google Title

Scenario: Validate title of Google

Given Open Google

When Read the title of page

Then Title should be Google

Validate google search

1. Open Google
2. Enter valid text in search bar
3. Hit Enter
4. Valid search result should display

Feature file

Feature: Validating Google Search

Scenario: To validate google search functionality

Given Open google in browser

When Enter valid text in search box

And Hit enter

Then Valid search result should display

**Tags in Cucumber**

Are like annotations in .feature file

Tags are specified in feature file only

You can give any name to tag

@TagName

Eg.

@SmokeTest

@GmailLink

**Hooks in Cucumber**

* Are two special methods (Annotations) namely @Before & @After
* @Before get executed only once before all scenarios
* @After get executed only once after all scenarios
* Hooks are never the part of .feature file
* You can add a separate class for hook (Even a new package can be created)