Framework Document

Definitions of Framework: -

- Framework is a well-organized Structure of reusable components where one driver(.xml) file will take care of the execution without manual intervention.
- Framework is a collection of reusable components that makes automation development execution and modification easier and faster.
- Frame Work is a set of instructions followed by every organization that makes automation test engineer life easy.

Types of Framework Approaches: -

There are 2 types of frameworks we have

1. TDD (Test Driven Development)
2. BDD (Behavior Driven Development)

TDD	*BDD*
▪ *Cases are mandatory to perform the TDD framework.*	▪ *Test Scenarios are mandatory to perform the BDD framework.*
▪ ***@Test** is the driving factor for test script development.*	▪ ***@Given, @When,** and **@Then** are used for test script development.*
▪ *TDD focuses on the implementation of the system.*	▪ *BDD focuses on the behavior of the application for End Users.*
▪ *In the TDD, we use Java to write the test script.*	▪ *In the BDD, we use Gherkin (like English) to write the script.*
▪ *In the small applications, we use the TDD framework.*	▪ *In large applications, we use the BDD framework.*
▪ *If functionality changes then the impact on scripts is more.*	▪ *If functionality changes then the impact on script is less than TDD.*

Types of Frameworks: -

1) ***Data Driven Framework: -***
   - Reading the data from any external resources OR reading the data from the data provider is called a Data-Driven Framework.
   - Data is a driving factor in DDT.
   - Whenever the test data is huge compared to test scenarios, we prefer DDT.
   - For DDT, we use Properties File, JSON, Excel Sheets, PDF, Databases, TestNG(.xml), and cmd (Maven) as external resources.
2) ***Modular Driven Framework: -***
   - Maintaining the test scripts, test data, and suite XML files module-wise to make the debugging process easy is called Modular Driven Framework.
   - By using MDT, code modification, maintenance, and debugging gets easy.
   - When every application is huge and has a lot of modules, maintaining all the modules together will be difficult, hence we prefer Modular-Driven Framework.
3) ***Method Driven Framework: -***

Framework Document

- Developing reusable methods for all the repetitive actions/functionalities in the application and calling those methods in test scripts is called a Method-Driven Framework.
- By reusing the code, test scripts are optimized, hence we prefer Method-Driven Framework.
- Whenever the application contains more repeated functionalities like dropdowns, frames, switching windows, etc. we are using MDT.

4) Keyword Driven Framework: -

- Creating a keyword library and utilizing those keywords to develop the test scripts is called Keyword-Driven Framework.
- Keyword is a driving factor in KDT.
- If the non-coder wants to perform automation, preferring Keyword-Driven Framework.
- When non-technical people whose coding would not be good want to perform the automation, they can use a Keyword-Driven Framework where all the actions are converted into user-friendly methods so that they can call those methods and perform the actions.

5) Hybrid Framework: -

- Writing the test scripts by using a combination of any two or more frameworks then it's called Hybrid Framework.
- Some applications will have huge data and also a larger number of modules so we have combined two features of the framework to make the framework user friendly, we prefer Hybrid Framework.

Advantages of Frameworks: -

I. Test Script Development is faster and easier because of reusability
II. Modification and maintenance of data is easy because data is stored in external resources.
III. Framework provides automatic screenshots for failed scripts.
IV. Framework Supports Multiple executions like Batch Execution, Regional Regression Execution, Distributed Parallel Execution, and Cross Browser Execution.
V. Framework Generates Accurate Test Reports for each execution and Optimizes the Test scripts

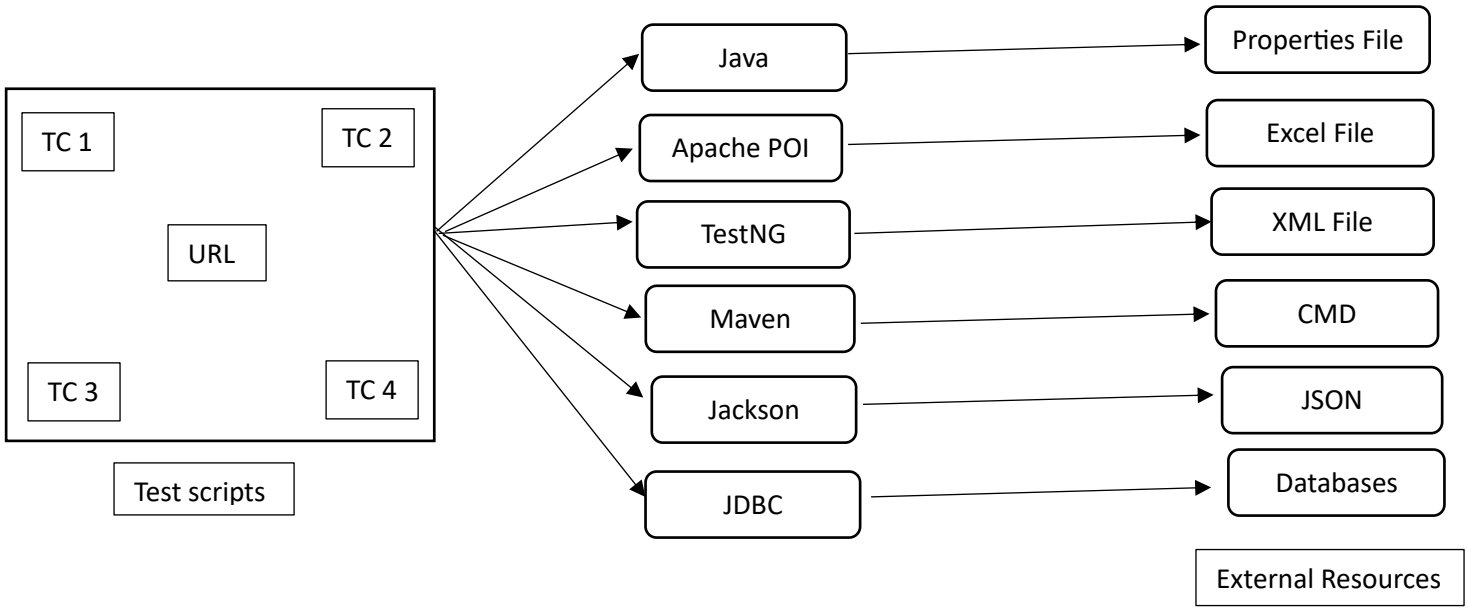
Disadvantages of Framework: -

I. Should be good at programming
II. Initial Framework Development cost and Time is High.

Data-Driven Framework: -

- A data-driven framework involves reading data from external resources OR reading data from the data provider.
- Data is a driving factor in DDT.
- Whenever the test data is huge compared to test scenarios, we prefer DDT.
- For DDT, we use Properties File, JSON, Excel Sheets, PDF, Databases, TestNG(.xml), and cmd (Maven) as external resources.

Framework Document



Properties File: -

Properties is a Java feature file where we can store data as a key-value pair. The key-value data type should always be a string.

Get the java representation Object of the Physical file using “FileInputSteam”. Create an Object of the “Properties” class & load all the keys. Read the data using getProperty (“Key”).

Format of Properties File-

File Name – PropertiesFile.properties

KEY	**Value**
URL	https://www.google.com
Browser	Chrome
Username	Admin
Password	Admin

Code-

```
FileInputStream fis = new FileInputStream(Path);

Properties p = new Properties();

p.load(fis);

String value = p.getProperty(key);
```

Framework Document

Excel File: -

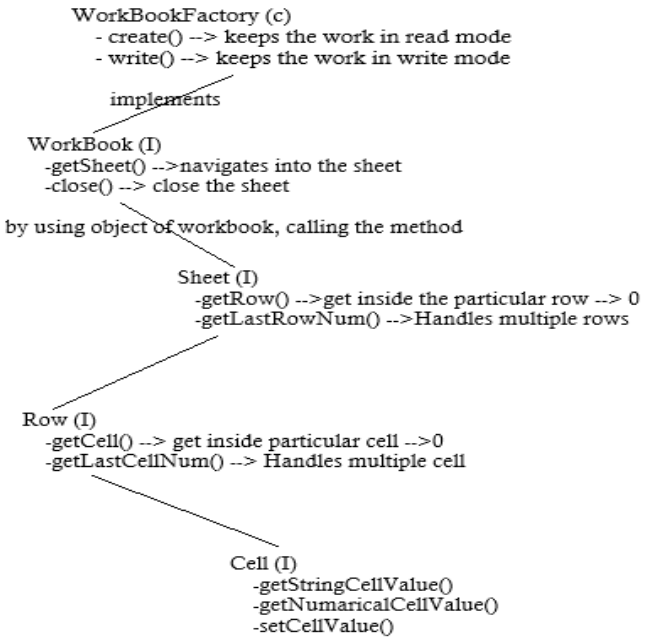
Apache POI is an open-source library for reading and writing data from Microsoft documents such as Excel, docx, ppt, etc.

Most of the time, the company prefers to keep the test script data in Excel because the data will be well-organized, making modification and maintenance easier.

Repositories that need to add the POM.xml file –

```
<!-- https://mvnrepository.com/artifact/org.apache.poi/poi -->
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>5.3.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml -->
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml</artifactId>
    <version>5.3.0</version>
</dependency>
```

Class Diagram of Apache POI –



Framework Document

Code –

```
FileInputStream fis = new FileInputStream(path);

Workbook book = WorkbookFactory.create(fis);

Sheet sheet = book.getSheet("Sheet1");

Row row = sheet.getRow(2);

Cell-cell = row.getCell(1);

String ExcelData = cell.getStringCellValue();

System.out.println(ExcelData);
```

JDBC:

It's a J2EE API that allows Java code to connect to any database. JDBC helps Java programs store/retrieve data from the physical database.

In real-time, every application requires the database to store/retrieve the user information in the database.

There are types of databases –

1. SQL Database -> Data will be stored as a table. E.g., MySQL, Oracle 11g
2. NoSQL Database -> Data will be stored in the form of JSON format. E.g., MongoDB

There are types of Queries –

1. Select Query -> Any query starting from select is called select query.
2. Non-select Query -> Any query starting from other than select is called a non-select query.

Components: -

1. *Common Utility*
2. *Object Repository (POM)*
3. *Test Data*
4. *Resources*
5. *Test Scripts*
6. *XML Files*
7. *HTML Report*
8. *Screenshots*
9. *Maven*
10. *GITHUB*
11. *Jenkins*

1. Common Utility –
   It's one of the automation framework components that is common to all applications. It is a collection of generic classes containing reusable methods/libraries.

Framework Document

The method that can be used for any application is called Generic/Common Utility. There are multiple classes we can add to the Generic Utility package like,

Package Name – GenericUtility
Classes – Java_Utility, File_Utility, Excel_Utility, WebDriver_Utility

- **Java_Utility –**
  It is one class in the generic component, which contains Java-specific methods that can be used across the test script/application.

  Code:

```
Class Java_Utility {
    Public int getRandomNum() {
        Random ran = new Random();
        Int ranNum = ran.nextInt(1000);
        return ranNum;
    }
}
```

- **File_Utility –**
  As per the rule of automation, data should not be hardcoded within the test scripts, so to get the data from an external file like File utility. File utility is used to get the data from the .properties file.

  Code:

```
Class File_Utility {
    Public string getKeyandValuePair(String key) {
        FileInputStream fis = new
FileInputStream(iPath.Property_filePath);
        Properties pro = new Properties();
        pro.load(fis);
        String value = pro.getProperty(key);
        return value;
    }
}
```

- **Excel_Utility –**
  As per the rule of automation, data should not be hardcoded within the test scripts to get the data from an external file like Excel utility.
  Excel Utility class is developed using Apache POI libraries, which are used to read the data from Excel.

  Code:

```
Class File_Utility {
    Public string getworkbookData(String sheetname, int rowValue, int
cellValue) {
```

```
                FileInputStream fis = new
FileInputStream(iPath.Property_filePath);
                Workbook book = WorkbookFactory.create(fis);
                Sheet sheet = book.getSheet(sheetname);
                Row row = sheet.getRow(rowValue);
                Cell cell = row.getCall(cellValue);
                String Data = cell.getStringCellValue();
                Return Data;
            }
        }
```

- **WebDriver_Utility –**
  WebDriver utility is a generic class, that contains WebDriver-specific reusable actions like,
  - WaitForPageToLoad()
  - WaitForElement()
  - Select()
  - acceptAlert()
  - cancleAlert(), etc.

  code:

```
public class WebDriver_Utility {

    public void maximizetheWindow(WebDriver driver) {

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

    }

    Public void waitfortheElement(WebDriver driver) {

     Driver.manage().timeouts().implicitlyWait(Duration.odSeconds(5));

     }

}
```

2. object Repository (POM) –
   It's a collection of elements, locators, and business libraries in one place, developed using the POM design pattern.

   Instead of writing test scripts, we can get elements from the Object Repository because in the Agile process, due to frequent requirement changes, modifications, and maintenance of elements are tedious.

   POM is a Java design pattern preferred by Google to develop Object Repositories. It's

Framework Document

A well-organized structured design pattern, where we can maintain all the web elements on a page. wise, due to POM design pattern maintains and modifications are easy and faster.

Code:

```
Public class LoginPage_Vtiger {
    Public LoginPage_Vtiger(WebDriver driver) {
        PageFactory.initElements(driver, this);
    }
    @FindBy(name = “user_name”)
     Private WebElement UserTextField;
     Public WebElement getUserTextField() {
         return UserTextField;
     }
     @FindBy(name= “user_password”)
      Private WebElement PasswordTextField;
      Public WebElement getPasswordTextField() {
          return PasswordTextField;
      }
      @FindBy(id= “submitButton”)
       Private WebElement LoginButton;
       Public WebElement getLoginButton() {
           return LoginButton;
       }
       Public void LoginIntoApp(String username, String password) {
           UserTextField.sendKeys(username);
           PasswordTextField.sendKeys(password);
           LoginButton.click();
       }
```

TestNG:

Automation Testers will make use of TestNG for developing the test scripts in more Optimized way.

Annotations in TestNG –

- @Test -> Works as main method, which is identified by JVM to start the execution.
- @BeforeSuite -> It is used to establishing a database connection.
- @AfterSuite -> It is used to close database connections.
- @BeforeTest -> It is mostly used for parallel executions as it creates multiple threads.
- @AfterTest -> It is mostly used for parallel executions as it creates multiple threads
- @BeforeClass -> It is used for launching the browser.
- @AfterClass -> It is used for closing the browser.

Framework Document

- @BeforeMethod -> It is used to login into the application.
- @AfterMethod -> It is used to logout from the application.

Batch Execution –

Executing all the existing test scripts sequentially/one after another. Batch execution is also called FULL REGRESSION testing. All the test scripts are loaded inside one suite XML file. In one suite xml file / testNg.xml file, we can invoke any number of classes but all the classes should be present inside the project. All the class names should be provided as qualified class name

packageName.Classname -

```
<class name="vtiger.ContactsTests.CreateContactWithOrgTest"/>
```

Code:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
 <test thread-count="5" name="Test">
        <classes>
                <class name="VtigerCRM.createCampaigns"/>
        </classes>
 </test> <!-- Test -->
</suite> <!-- Suite -->
```

Group Execution –

Executing the similar kind of test script under a group. [ex: smokeSuite, RegressionSuite]
All type test script belong either to smoke suite or to Regression suite. To achieve group execution, Every @Test should be included in the group

```
@Test(groups = "RegressionSuite")
@Test(groups = {"SmokeSuite","RegressionSuite"})
```

Code –

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
 <groups>
        <run>
                <include name="smokeTest"></include>
        </run>
 </groups>
```

Framework Document

```
 <test thread-count="5" name="Test">
        <classes>
            <class name="VtigerCRM.createCampaigns"></class>
            <class name="VtigerCRM.createProduct"></class>
            <class name="VtigerCRM.CreateCampaignsWithProduct"></class>
        </classes>  </test> <!-- Test -->
</suite> <!-- Suite -->


public class TestNG_BaseClass {
        public WebDriver driver;
        public WebDriverUtility WDU = new WebDriverUtility();
        public static WebDriver sDriver;

        @BeforeSuite(groups = {"regressionTest","sanityTest"})
        public void beforeSuiteAnn() {
                System.out.println("DataBase Connection");
        }
.
.
.
@AfterSuite (groups = {"regressionTest","sanityTest"})
        public void afterSuitAnn() {
                System.out.println("Close DataBase");
        }
}
```

Parallel executions

Parallel execution means multiple threads start the execution simultaneously.
We can achieve parallel execution in the form of,

- distributed Parallel execution.
- Cross Browser Parallel Execution.
- Cross Platform Parallel Execution

- Distributed Parallel execution:

  We prefer distributed paralle execution when we have to reduce the total execution time taken by a suite. We distribute the total number of existing test scripts amoung multiple threads in Suite xml file with <test> and start the execution. Browser will not be changed here. Only the test scripts are distributed.

  Different threads - Different test scripts - Same browser lauched in all Threads. One thread can have any number of test classes and all the test class inside the <test> and </test> will be executed Sequentially

Code –

```
<?xml version="1.0" encoding="UTF-8"?>
        <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
```

Framework Document

```
<suite name="Suite">
<test thread-count="5" name="TestRunner1">
<classes>
        <class name="VtigerCRM.createContact"></class>
        <class name="VtigerCRM.createOrganization"></class>
</classes>
</test> <!-- Test -->

 <test name="TestRunner2">
<classes>
        <class name="VtigerCRM.createCampaigns"></class>
        <class name="VtigerCRM.createProduct"></class>
</classes>
</test>

<test name="TestRunner3">
<classes>
<class name="VtigerCRM.CreateContactwithOrganization"></class>
</classes>
</test>
</suite> <!-- Suite -->
```

- Cross Browser Execution/Compability Testing:

  Cross browser execution means executing the same set of test scripts in Multiple browsers to ensure they are compatible. In cross browser execution, same set of test scripts are executed over different browser in different threads.

  Different Threads - Different Browsers - same set of Test scripts.

  Since during run time we have to choose the browser for execution, we have provide browser name from suite xml file instead of property file.

  <Parameter> is used to set the name and value which vll pass this data @Parameters annotation in base class.

Code –

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite parallel="tests" name="Suite">

 <test name="TestRunner2">
        <parameter name="BROWSER" value="edge"></parameter>
        <parameter name="URL" value="http://localhost:8888/"></parameter>
        <parameter name="USERNAME" value="admin"></parameter>
        <parameter name="PASSWORD" value="shruti"></parameter>
        <classes>
                <class name="VtigerCRM.createOrganization"></class>
        </classes>
 </test>
```

Framework Document

</suite> <!-- Suite -->

Exceptions:

1. NoSuchWindowException -

One of the most frequent exceptions in Selenium WebDriver, NoSuchWindowException occurs if the current list of windows is not updated. The previous window does not exist, and you can't switch to it.

To handle this exception, use WebDriver methods called "driver.getWindowHandles()."

2. NoSuchFrameException

Similar to NoSuchWindowException, the NoSuchFrameException occurs when switching between multiple frames is not possible.

The solution used for handling NoSuchWindowException must ideally work for this exception too.

3. NoSuchElementException

Happens when the locators are unable to find or access elements on the web page or application. Ideally, the exception occurs due to the use of incorrect element locators in the findElement(By, by) method.

To handle this exception, use the wait command. Use Try/Catch to ensure that the program flow is interrupted if the wait command doesn't help.

4. NoAlertPresentException

Happens when the user is trying to you switch to an alert which is not present. In simple terms, it means that the test is too quick and is trying to find an alert that has not yet been opened by the browser.

To handle or simply avoid this exception, use explicit or fluent wait in all events where an alert is expected.

5. InvalidSelectorException

This exception occurs due to an incorrect selector. More often than not, the selector syntax is wrong.

To avoid this exception, first, check the locator syntax. If it is incorrect, make sure the locator is syntactically correct.

6. TimeoutException

This exception is thrown if the command did not execute or complete within wait time. As already mentioned, waits are used to avoid NoSuchElementException. However, TimeoutException will be thrown after the page components fail to load within wait time.

Avoiding this exception is simple. All one needs to do is to calculate the average page load time and adjust the wait time accordingly.

7. ElementNotVisibleException

This exception occurs when the WebDriver tries to find an element that is hidden or invisible. To handle this exception, it is essential that the exact reason is identified, which can be due to nested web elements or overlapping web elements.

In the first case, you have to locate and correct the specific element. In the second case, use explicit wait.

8. ElementNotSelectableException

This exception belongs to the same class as InvalidElementStateException. In such exceptions, the element is present on the web page, but the element cannot be selected.

To handle this exception, the wait command must be used.

9. NoSuchSessionException

As the name suggests, the exception is thrown if a method is called after the browser is closed. Other reasons for this exception include a browser crash.

To avoid this handle, ensure that your browser is updated and stable.

10. StaleElementReferenceException

This exception occurs when the web element is no longer part of the web page. The element may have been part of the source code, but it is no longer part of the window. There can be multiple reasons for this exception. It can occur either from a switch to a different page, the element is no longer part of DOM, or due to a frame/window switch

To handle this exception, you can either use Dynamic Xpath for handling DOM operations or try to use the Page Factory Model, or try to access the element in loops before throwing the exception.