**FRAMEWORK EXPLANATION**

**Project overview:**

In my selenium automation project, I worked on automating the salesforce CRM website where user could login (with valid/invalid credentials), Navigate to Sales page 🡪 leads tab, and create a lead.

Application domain – CRM  
Type of Testing covered – Smoke, Sanity and Functional

**Programming language used:**

I have used Java because of it’s strong capability with Selenium webdriver and the rich support for tools like Maven, and TestNG.

Of course, we can also use C#/JavaScript language but I have went with Java because of my experience.

**Framework Type and Design**

I have used a Hybrid framework that combined the benefits of both Data-driven and Page Object model structures

Why POM?

I’ve chosen POM to separate test login from UI elements. Each page has its Java class file, making it easy to maintain the scripts as the application evolves.

This also helps us –

* Increased reusability.
* Code maintenance.
* Readability for future users.

**Folder Structure**

I have the following packages under “src/test/java”:

* Base package – This class deals and holds all the functions that are commonly used by all pages. This is class is responsible for loading the configuration from the property files like initialising the webdriver. This class drives the test scripts and helps to perform all the actions like on which browser the script should be executed.
* Libraries package – This class holds all the classes and methods that are commonly used across the entire framework. This class holds all the repetitive code like generating report, wrapper methods, faker class to fetch dynamic data. The main objective for creating this class is to achieve reusability.
* Pages package – This package holds the java class files for each of the page we expect to work in the application. Each class file holds the Web Element and the actions (methods) to be performed on the page.
* Test Scenarios / Cases – This package holds the java class file that has a purpose (Test case) and the purpose is achieved by calling methods present in the pages class files and verifying the results.
* Util package – holds the Java class files with Property file reader and Excel file reader, and writer. These are written in such a way that they can be used throughout the framework.
* Data folder – contains the Excel file (to drive the TDD) and property file to fetch the base URL value and browser to run the tests.
* Report folder – Extent report generated are stored in the reports folder.
* Pom.xml – contains the maven dependency.

**Execution tool:**

We have used TestNG as our execution tool as it helps us group our tests, assertions, set priorities to test, TestNG listeners to retry in case of failure, parallel execution, detailed logs, and reports.

**How did we achieve TDD:**

In our Selenium Automation Project, we used Apache POI to read and write test data from Excel files. With this, we have multiple sets of data using Data Provider with the same set of code.

**Execution and Reporting:**

We used TestNG and integrated ExtentReports to generate rich HTML reports, including test status, execution time, and screenshots on failure.

**Challenges faced:**

**Challenge 1: -** Some of the elements (For example, Lead status) will not be displayed on screen unless user scrolls to view the element and this sometimes gives us failure.

So, I have used the Actions class to move to the element next to lead status, and then I interact with the Lead status element.

**Challenge 2: -** Some of the element after click, takes time to load the sub window.

Example – On clicking the App launcher 🡪 View All or User image button, Application takes approximately 3 t o5 seconds to display the sub window.

So, I have used WebDriverWait and Thread.sleep methods to wait until the elements are in view to interact.

**Version control: -** I have leveraged Git and GitHub to maintain a centralised repository.