UI Automation – TestNG Framework

Version control

Version	Prepared by	Reviewed by	Date	Location
1.0	Neha Lalit	Sangeetha & Vidyashree	28/03/2025	Chennai
2.0	Neha Lalit	Sangeetha & Vidyashree	09/04/2025	Chennai
3.0	Neha Lalit	Sangeetha & Vidyashree	16/04/2025	Chennai
4.0	Neha Lalit	Sangeetha & Vidyashree	24/04/2025	Chennai

Version control History

Version	Reasons for change
1.0	Asked to make changes in Introduction and complete the pre-requisite
2.0	Asked to explain Framework Outline more in detail with examples
3.0	Asked to update the report generation and maintenance part
4.0	Asked to format the document properly and few changes in maintenance part

1. Introduction

1.1 Purpose

This document provides a guide to the UI automation framework using TestNG, outlining its structure, tools, and best practices. It ensures consistent, scalable, and efficient testing processes, helping teams improve software quality and streamline collaboration.

1.2 Overview

UI Automation refers to the use of tools and scripts to automatically test the graphical interface of an application. It ensures that the user interface components, such as buttons, menus, forms, and input fields, function as intended and provide a seamless user experience. Instead of manually testing the application, automation allows repetitive tasks to be executed efficiently, reducing time and effort while increasing accuracy.
UI Automation is essential for large-scale applications or systems requiring frequent regression testing, as it accelerates the testing process and enhances software quality.

Various types of UI Automation Testing are - Functional, Cross-Browser, Cross-Platform, Visual/UI Validation, End-to-End, Regression, Data-Driven.

Common Frameworks for UI Automation:

Several frameworks are available for UI automation, each designed to address specific testing needs. Here are a few commonly used ones:

1. **Selenium WebDriver:**
    - A browser automation tool that supports testing web applications across multiple browsers (e.g., Chrome, Firefox, Edge).
    - Offers multi-language support (e.g., Java, Python, C#, Ruby).
    - Frequently used in combination with frameworks like TestNG or JUnit for managing test cases.

2. **TestNG:**
    - A powerful testing framework designed for Java applications.
    - Features include annotations (@Test, @BeforeMethod), parallel execution, and advanced reporting capabilities.
    - Often paired with Selenium for streamlined UI automation.

3. **Appium:**
    - A cross-platform framework for automating mobile applications (Android and iOS).
    - Supports testing native, hybrid, and web-based mobile apps.
    - Written in the same language as the application, enhancing compatibility.

4. **Cypress:**
    - A JavaScript-based framework designed for modern web applications.
    - Known for its fast and reliable testing, interactive debugging, and real-time execution.
    - Ideal for end-to-end testing of single-page applications.

5. **Playwright:**
    - Developed by Microsoft, this framework supports testing across Chromium, WebKit, and Firefox browsers.
    - Offers reliable cross-browser compatibility and parallel execution features.
    - Ideal for complex scenarios requiring high-speed automation.

6. **Robot Framework:**
    - A generic test automation framework that uses simple tabular syntax.
    - Compatible with various libraries and tools, including Selenium.
    - Popular for its readability and ease of use, even for non-developers.

1.3 Benefits

UI (User Interface) automation offers several advantages that make it a powerful tool for testing, development, and overall efficiency. Here are some of its key benefits:

1. Improved Accuracy
- Eliminates human errors that can occur during manual testing or repetitive tasks.

2. Time Efficiency
- Executes repetitive tasks or test cases faster than humans, saving significant time during software development cycles.

3. Cost-Effective in the Long Run
- Though initial setup costs might be high, automation reduces manual effort, leading to long-term cost savings.

4. Enhanced Test Coverage
- Allows for more test cases to be executed across different scenarios, environments, and devices, which might not be feasible manually.

5. Reusability
- Automated scripts can be reused for similar testing scenarios, making the process efficient.

6. Consistency
- Automation ensures the same steps are followed every time, leading to reliable and consistent results.

7. Continuous Integration and Delivery (CI/CD)
- Facilitates seamless integration into CI/CD pipelines, enabling quick feedback and faster releases.

8. Better Resource Utilization
- Frees up human testers to focus on exploratory and high-value testing activities instead of mundane, repetitive tasks.

9. Scalability
- Handles a large number of test cases and scales easily as the application or system grows.

2. Framework followed in Mindsprint

2.1 Pre-requisite

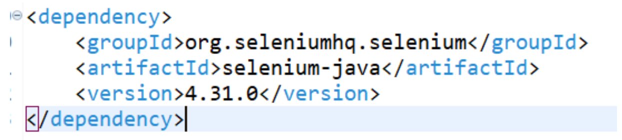
Tool	Version	Reference
Java Development Kit	11.0.16.1	Java Downloads | Oracle
Selenium WebDriver	4.30.0	Downloads | Selenium
Maven	3.9.9	Download Apache Maven – Maven
TestNG	7.11.0	Maven Repository: org.testng » testng » 7.11.0

Setup Guide

Java Setup
1. Download and install the latest version of the Java Development Kit (JDK) from the official Oracle website.
2. Set up the JAVA_HOME environment variable to point to the JDK installation directory.
3. Verify the installation by running java -version in the command prompt or terminal.

Selenium Setup
1. Download the Selenium WebDriver from the official Selenium website.
2. Add the .jar files and libraries to your project's classpath:
   - In Eclipse, right-click your project, navigate to Build Path > Add External Archives, and select the downloaded .jar files.

This can also be done by adding the Maven dependency in the pom.xml file of your project. This eliminates the need to manually download and add .jar files, as Maven automatically handles the required libraries for Selenium.



```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.31.0</version>
</dependency>
```

1. Setting up TestNG using Eclipse IDE

- Open Eclipse and navigate to Help > Eclipse Marketplace.
- Search for TestNG and click Install.
- Accept the license agreement and restart Eclipse after installation.
- Verify installation by going to Window > Show View > Other and checking if TestNG is listed.
- Create a new Java project in Eclipse.
- Right-click the project and go to Build Path > Configure Build Path, then add the TestNG library.

2. Setting up TestNG using Maven

- Open your project in a Maven-supported IDE (e.g., Eclipse or IntelliJ).
- Add the following dependency to the pom.xml file:

```
<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>7.9.0</version>
    <scope>test</scope>
</dependency>
```

- Save the file, and Maven will automatically download the required TestNG libraries.
- Create a TestNG class and run it by using the Maven test command.

3. Setting up TestNG using Gradle

- Open your project in a Gradle-supported IDE (e.g., IntelliJ or Eclipse).
- Add the following to the dependencies section in your build.gradle file:

```
dependencies {
    testCompile 'org.testng:testng:7.9.0'
}
```

- Save the file, and Gradle will download the TestNG dependencies.
- Use Gradle's test task to run your TestNG tests.

4. Manual Setup of TestNG

- Download the TestNG .jar file from TestNG's official website.
- Add the .jar file to your project's classpath:
  - In Eclipse, right-click your project, go to Build Path > Add External Archives, and select the downloaded .jar.
- Create a TestNG class with annotations such as @Test, and run the file as a TestNG test.

2.2 Framework Outline

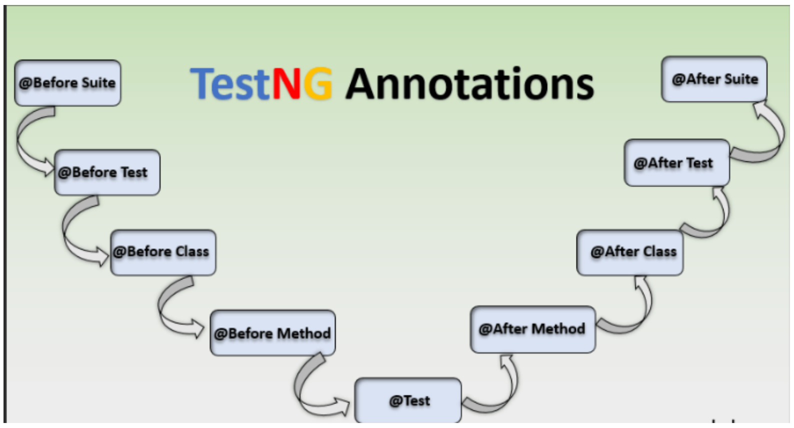
The TestNG-based UI automation framework followed in MindSprint is designed to provide a modular, scalable, and reusable approach to test development and execution.It is a Java-based testing framework designed to overcome the limitations of JUnit and other older frameworks.It leverages the Page Object Model (POM) for maintainable code, parameterization for data-driven testing, and integration with CI/CD pipelines for automated workflows. Below is the detailed outline:

KEY FEATURES OF TESTNG FRAMEWORK

1. Annotations

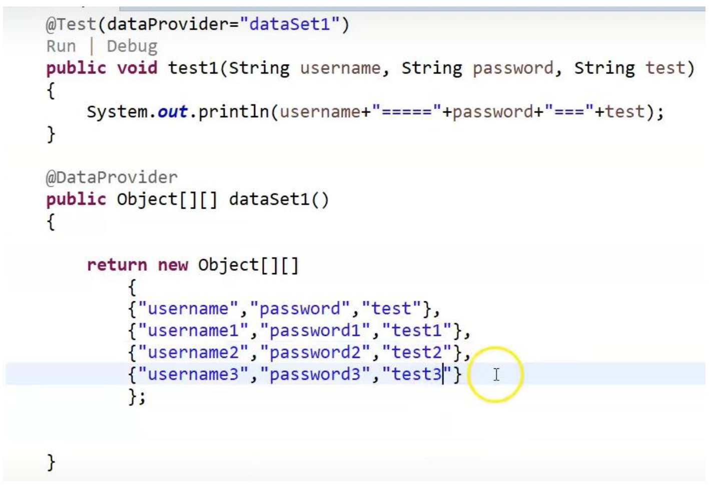
TestNG introduces a comprehensive annotation system that simplifies test structure and lifecycle management. These annotations define the execution order and control the flow of tests:

- @Test: Marks a method as a test case.
- @BeforeMethod and @AfterMethod: Executes setup and cleanup methods before and after each test.
- @BeforeClass and @AfterClass: Executes methods once before and after all tests in a class.
- @BeforeSuite and @AfterSuite: Executes methods at the beginning and end of the test suite.
- @DataProvider: Supplies dynamic data to test methods for data-driven testing. Annotations can also have attributes like enabled, priority, dependsOnMethods, and expectedExceptions to offer greater control over test execution.



2. Parameterization

TestNG allows parameterized testing by defining parameters in the testng.xml file or using @DataProvider. This makes tests flexible and reusable by allowing different input data sets for the same test logic. Parameterized testing is useful for testing various input-output scenarios and ensures broader test coverage.

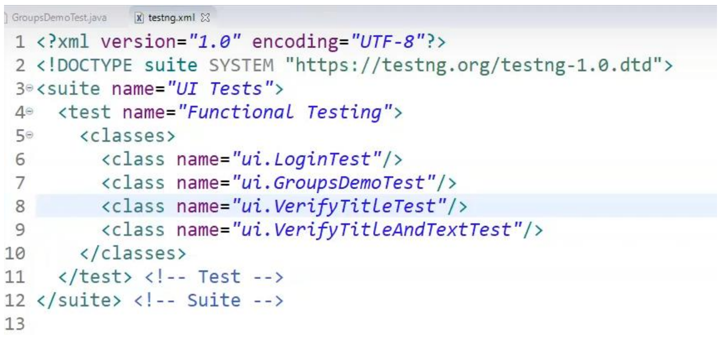


```java
@Test(dataProvider="dataSet1")
Run | Debug
public void test1(String username, String password, String test)
{
    System.out.println(username+"====="+password+"==="+test);
}

@DataProvider
public Object[][] dataSet1()
{

    return new Object[][]
        {
        {"username","password","test"},
        {"username1","password1","test1"},
        {"username2","password2","test2"},
        {"username3","password3","test3"}
        };


}
```

3. TestNG Suite

TestNG test suites are defined in the *testng.xml* file to organize and manage test cases efficiently. A suite can include multiple tests, classes, or groups. Key features of test suites include:

- Logical grouping of test cases for targeted execution.
- Centralized management of test classes and methods.
- Configuration of global parameters.
- Ability to define parallel execution to optimize time and resources.

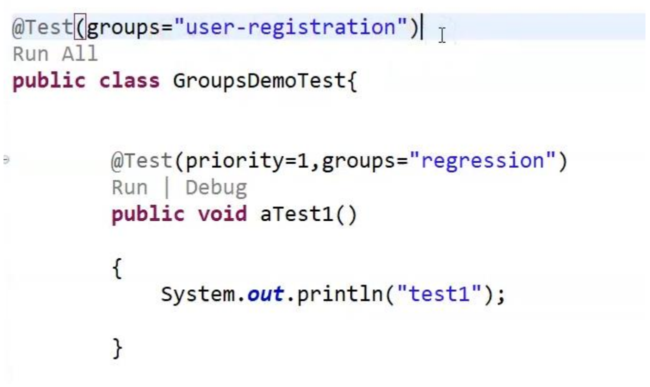


```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="UI Tests">
  <test name="Functional Testing">
    <classes>
      <class name="ui.LoginTest"/>
      <class name="ui.GroupsDemoTest"/>
      <class name="ui.VerifyTitleTest"/>
      <class name="ui.VerifyTitleAndTextTest"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

4. Testng Grouping

TestNG provides the ability to group test methods logically using the @Test(groups="groupName") annotation. This enables efficient execution of specific sets of tests, such as:

- Grouping tests by feature, functionality, or modules.
- Running selected groups based on project requirements.
- Managing dependencies between test groups using dependsOnGroups. Groups streamline test execution and improve organization in large-scale projects.



```
@Test(groups="user-registration")
Run All
public class GroupsDemoTest{

        @Test(priority=1,groups="regression")
        Run | Debug
        public void aTest1()

        {
            System.out.println("test1");

        }
```

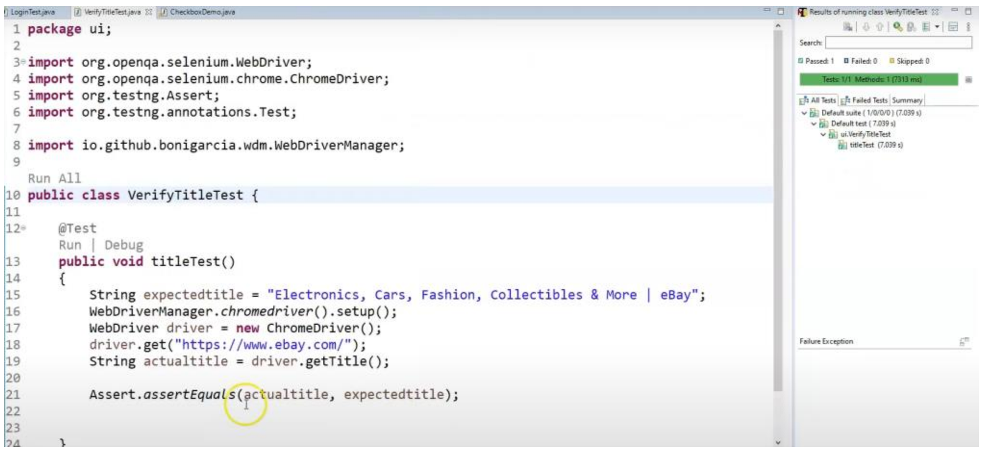
5. Assertions

Assertions validate test outcomes by comparing actual results to expected values. TestNG's assertion methods ensure test reliability:

- *assertEquals(actual, expected):* Validates equality.
- *assertTrue(condition):* Checks if a condition is true.
- *assertFalse(condition):* Ensures a condition is false.
- *assertNull(object):* Confirms that an object is null.
- *assertNotNull(object):* Ensures an object is not null.

Two types of asserts:

- **Soft Assert**
  A soft assert lets the test keep running even if something fails. All the issues are collected and reported together at the end. It's great when you want to check multiple things in one go.
- **Hard Assert**
  A hard assert stops the test as soon as something goes wrong. It's used for critical checks where the next steps depend on the earlier ones working properly.



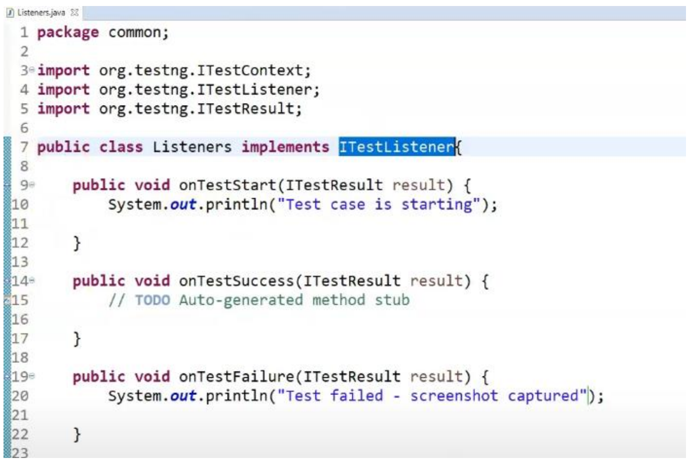
```
package ui;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.Test;

import io.github.bonigarcia.wdm.WebDriverManager;

Run All
public class VerifyTitleTest {

    @Test
    Run | Debug
    public void titleTest()
    {
        String expectedtitle = "Electronics, Cars, Fashion, Collectibles & More | eBay";
        WebDriverManager.chromedriver().setup();
        WebDriver driver = new ChromeDriver();
        driver.get("https://www.ebay.com/");
        String actualtitle = driver.getTitle();

        Assert.assertEquals(actualtitle, expectedtitle);


    }
```

```
Passed: 1   Failed: 0   Skipped: 0
Tests: 1/1  Methods: 1 (7313 ms)
Default suite ( 1/0/0/0 ) (7.039 s)
  Default test ( 7.039 s)
    ui.VerifyTitleTest
      titleTest (7.039 s)
```

6. Listeners

Listeners are interfaces that allow customization of test execution behavior. They monitor and react to test events like success, failure, or skip. Few TestNG Listeners are ITestListener, ISuiteListener, IReporter, IAnnotationTransformer, IMethodInterceptor, and ITestNGListener. Common TestNG listeners include:

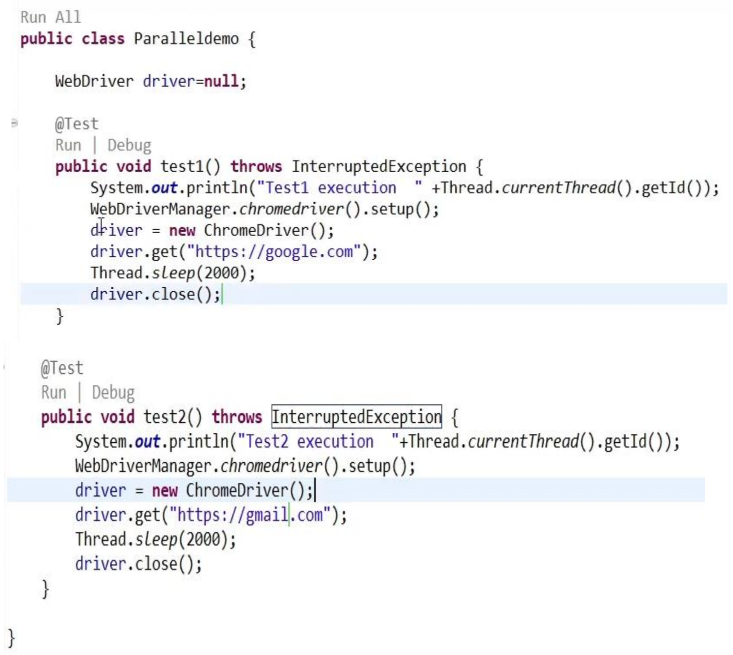
- *ITestListener*: Tracks test-level events such as success and failure.
- *ISuiteListener*: Monitors suite-level events like start and completion. Listeners are useful for adding custom logging, retrying failed tests, and generating enhanced reports.



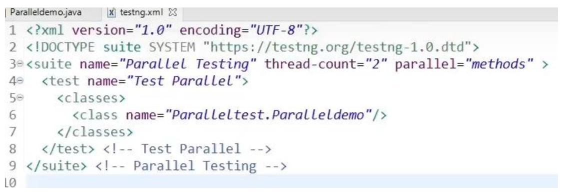
```java
package common;

import org.testng.ITestContext;
import org.testng.ITestListener;
import org.testng.ITestResult;

public class Listeners implements ITestListener{

    public void onTestStart(ITestResult result) {
        System.out.println("Test case is starting");

    }

    public void onTestSuccess(ITestResult result) {
        // TODO Auto-generated method stub

    }

    public void onTestFailure(ITestResult result) {
        System.out.println("Test failed - screenshot captured");

    }

```

7. Parallel Execution

TestNG supports parallel test execution at the method, class, or suite level. Parallel execution optimizes testing time by utilizing multiple threads. This is especially beneficial for large projects with extensive test cases. You can configure parallel execution directly in the testng.xml file.



```java
public class Paralleldemo {

    WebDriver driver=null;

    @Test
    public void test1() throws InterruptedException {
        System.out.println("Test1 execution  " +Thread.currentThread().getId());
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
        driver.get("https://google.com");
        Thread.sleep(2000);
        driver.close();
    }

    @Test
    public void test2() throws InterruptedException {
        System.out.println("Test2 execution  "+Thread.currentThread().getId());
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
        driver.get("https://gmail.com");
        Thread.sleep(2000);
        driver.close();
    }

}
```

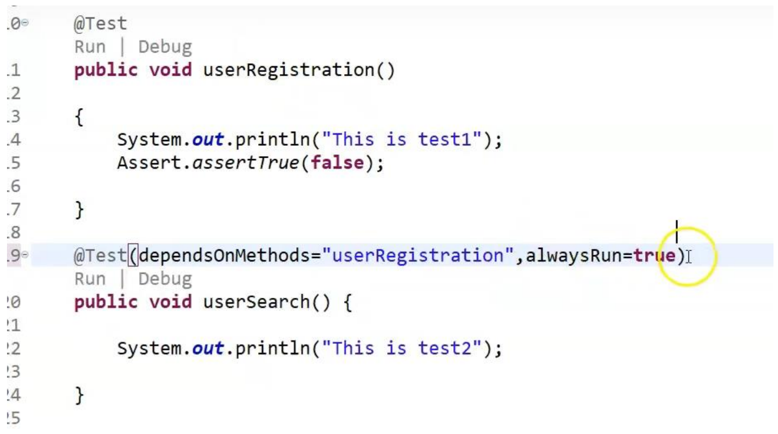


```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Parallel Testing" thread-count="2" parallel="methods" >
  <test name="Test Parallel">
    <classes>
      <class name="Paralleltest.Paralleldemo"/>
    </classes>
  </test> <!-- Test Parallel -->
</suite> <!-- Parallel Testing -->
```

In this example Google and Gmail will run parallelly.

8. Test dependency

TestNG allows specifying dependencies between test methods using attributes like *dependsOnMethods* and *dependsOnGroups*. This ensures logical flow and prevents dependent tests from running if prerequisite tests fail. Dependency management is crucial for maintaining the correct sequence in complex testing scenarios.



```
    @Test
    Run | Debug
    public void userRegistration()

    {
        System.out.println("This is test1");
        Assert.assertTrue(false);

    }

    @Test(dependsOnMethods="userRegistration",alwaysRun=true)
    Run | Debug
    public void userSearch() {

        System.out.println("This is test2");

    }
```

In this example, userSearch() method depends on userRegistration() method. So if userRegistration() method fails to execute then userSearch() method get skipped. In this scenario to execute userSearch() method we write ‘alwaysRun=true’.

9. Report Generation

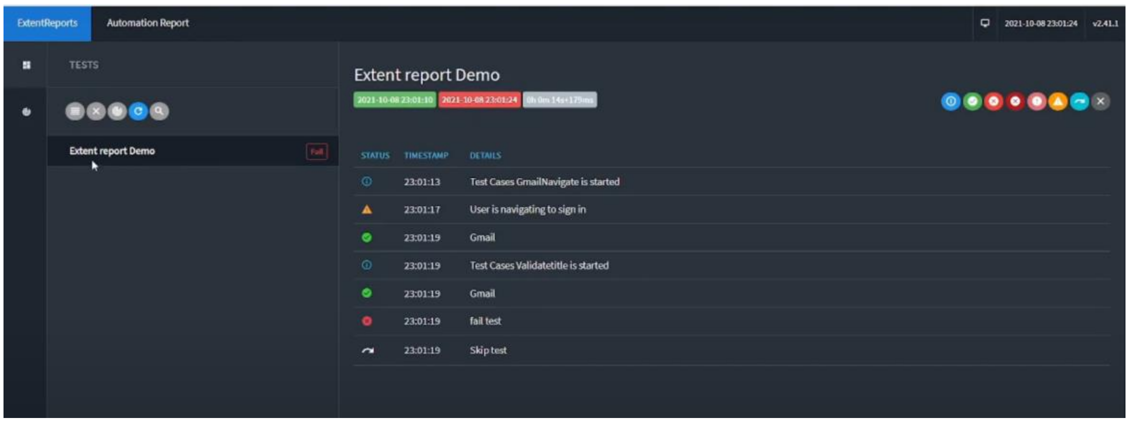
TestNG provides detailed HTML and XML reports after executing test suites. These reports include information on passed, failed, and skipped tests, along with execution time and error stack traces. Advanced reporting tools, such as ExtentReports, ReportNG etc. can be integrated to create visually appealing and comprehensive reports for stakeholders.

Analysis of Test Results: Pass, Fail, Skip

1. **Pass**:
   - A test is marked as "passed" when all assertions within the test method are successful.
   - No exceptions are thrown during the execution of the test.
2. **Fail**:
   - A test is marked as "failed" if any assertion fails.
   - If an exception is thrown during the test execution, it is also marked as failed.
   - The report includes detailed stack traces and error messages for debugging.
3. **Skip**:
   - A test is marked as "skipped" if it is dependent on another test that fails or is skipped.
   - Tests annotated with @Test(enabled = false) are also skipped.
4. **Warning:**
   - A warning is not directly related to the pass/fail status but indicates a potential issue that could cause problems in future test runs.
5. **Info:**
   - Info logs are used to provide additional context or information during test execution.

- They include details such as the starting and ending of test methods, timestamps, or environment details.
- Info logs are valuable for understanding the flow and behavior of the test suite without indicating errors or warnings.

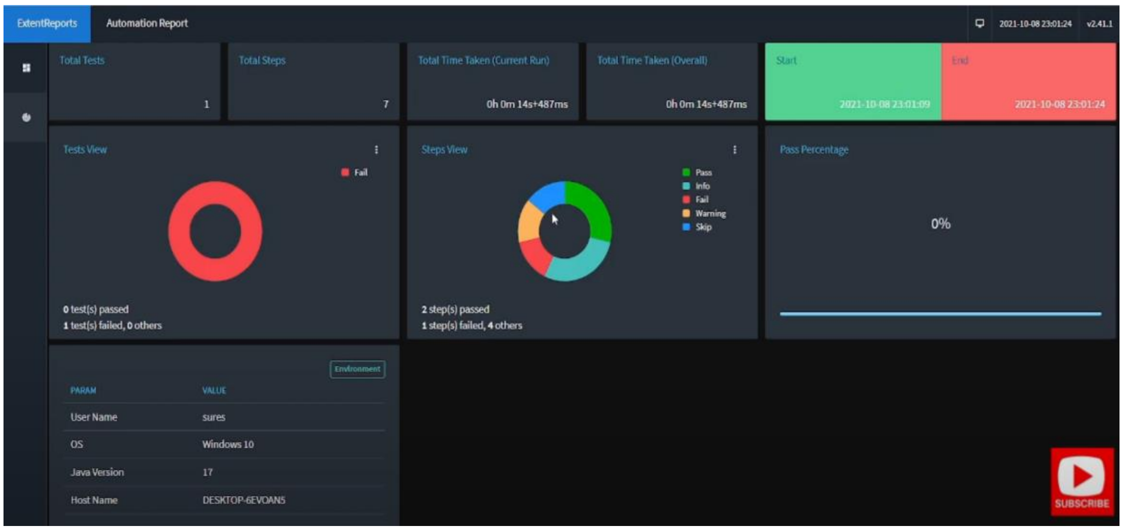
ExtentReports: ExtentReports is a robust reporting library that allows for the generation of dynamic and interactive reports. It offers features such as dashboards, pie charts, and logging integration, which make it easier for stakeholders to analyze test results comprehensively. ExtentReports is highly flexible and can seamlessly integrate with TestNG and Selenium. These reports not only highlight the status of each test but also present detailed error logs and execution details in an engaging format. Sample output generated by ExtendReports is as follows:



Extent report Demo

2021-10-08 23:01:10 | 2021-10-08 23:01:24 | 0h 0m 14s+179ms

STATUS	TIMESTAMP	DETAILS
ⓘ	23:01:13	Test Cases GmailNavigate is started
⚠	23:01:17	User is navigating to sign in
✓	23:01:19	Gmail
ⓘ	23:01:19	Test Cases Validatetitle is started
✓	23:01:19	Gmail
●	23:01:19	fail test
↷	23:01:19	Skip test



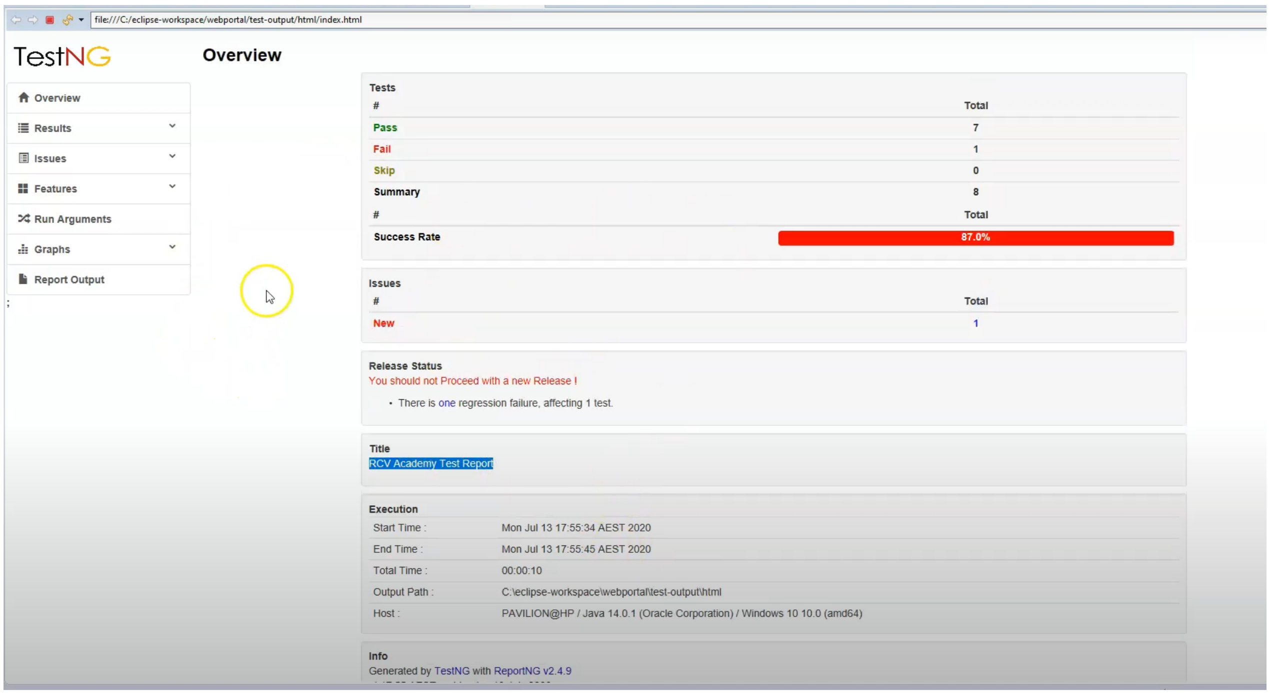
Total Tests	Total Steps	Total Time Taken (Current Run)	Total Time Taken (Overall)	Start	End
1	7	0h 0m 14s+487ms	0h 0m 14s+487ms	2021-10-08 23:01:09	2021-10-08 23:01:24

0 test(s) passed
1 test(s) failed, 0 others

2 step(s) passed
1 step(s) failed, 4 others

PARAM	VALUE
User Name	sures
OS	Windows 10
Java Version	17
Host Name	DESKTOP-6EVOAN5

ReportNG: ReportNG is an enhanced reporting plugin for TestNG that simplifies the readability of reports. It produces visually appealing and straightforward HTML reports with better formatting compared to the default TestNG reports. Its customization options make it ideal for presenting test results effectively in industry settings.
Report Generated using ReportNG is as follows:



Overview

Tests

#	Total
Pass	7
Fail	1
Skip	0
Summary	8
#	Total
Success Rate	87.0%

Issues

#	Total
New	1

Release Status
You should not Proceed with a new Release !

- There is one regression failure, affecting 1 test.

Title
RCV Academy Test Report

Execution

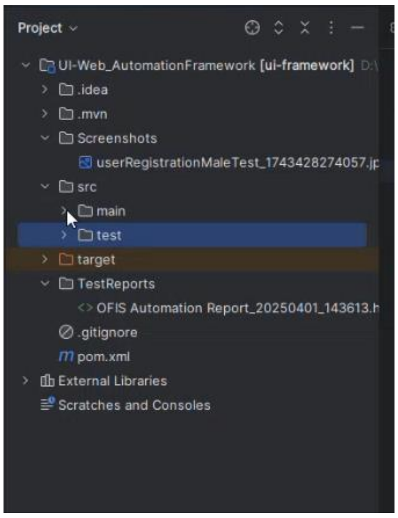
Start Time :	Mon Jul 13 17:55:34 AEST 2020
End Time :	Mon Jul 13 17:55:45 AEST 2020
Total Time :	00:00:10
Output Path :	C:\eclipse-workspace\webportal\test-output\html
Host :	PAVILION@HP / Java 14.0.1 (Oracle Corporation) / Windows 10 10.0 (amd64)

Info
Generated by TestNG with ReportNG v2.4.9

2.3 UI Automation Framework- Mindsprint

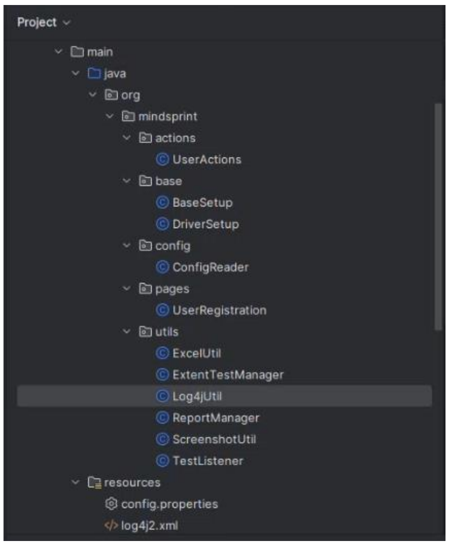
This framework is designed to simplify and standardize UI automation processes, ensuring efficient and reliable testing practices. It provides a well-organized structure that makes it easier to manage code, resources, and test evidence while supporting industry-standard tools and techniques.

Below is the folder structure and its components:



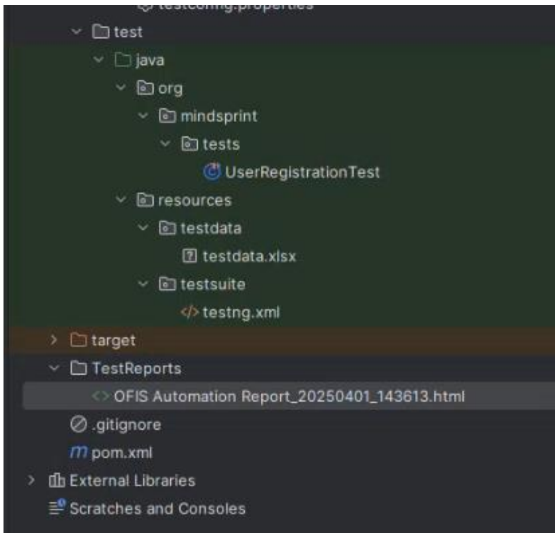
```
Project
  UI-Web_AutomationFramework [ui-framework] D:\
    .idea
    .mvn
    Screenshots
      userRegistrationMaleTest_1743428274057.jp
    src
      main
      test
    target
    TestReports
      OFIS Automation Report_20250401_143613.h
    .gitignore
    pom.xml
  External Libraries
  Scratches and Consoles
```

src/main folder



```
Project
  main
    java
      org
        mindsprint
          actions
            UserActions
          base
            BaseSetup
            DriverSetup
          config
            ConfigReader
          pages
            UserRegistration
          utils
            ExcelUtil
            ExtentTestManager
            Log4jUtil
            ReportManager
            ScreenshotUtil
            TestListener
    resources
      config.properties
      log4j2.xml
```

src/test folder



```
  test
    java
      org
        mindsprint
          tests
            UserRegistrationTest
      resources
        testdata
          testdata.xlsx
        testsuite
          testng.xml
  target
  TestReports
    OFIS Automation Report_20250401_143613.html
  .gitignore
  pom.xml
External Libraries
Scratches and Consoles
```

1. **pom.xml**

The pom.xml file in Maven projects is essential for:

- Managing dependencies like TestNG, Selenium, and ExtentReports.
- Configuring build and plugin settings for tasks like compiling, testing, and generating reports.
- Automating version control and project metadata.
- Supporting profiles to tailor builds for different environments.

It simplifies project management and streamlines workflows for building, testing, and deploying applications.

2. **src/main/java**

Contains the main application code, which is neatly organized for easy management and reuse:

- Actions: Classes for defining user actions and business logic.
- Base: Classes for foundational setup (e.g., driver setup and base configurations).
- Configuration: Classes to manage property files and centralized settings.
- Pages: Implements Page Object Model (POM) with classes representing UI pages and workflows.

- Utilities: Helper classes for reusable functionalities like logging, reporting, and data handling.

3. **src/main/resources**
   Contains resources used by the application:
   - Configuration Files: Centralized property files for application-specific settings (e.g., config.properties).
   - Log Configuration: Files for logging framework setup (e.g., log4j2.xml).

4. **src/test/java**
   Contains test scripts and classes:
   - Test Classes: Test methods implemented using TestNG annotations (@Test) to define test cases.

5. **src/test/resources**
   Contains resources specific to testing:
   - Test Data Files: External files (e.g., testdata.xlsx) for data-driven tests.
   - Test Suite Configuration: Central TestNG configuration file (e.g., testng.xml) for organizing test suites and defining execution priorities.

6. **Target**
   Automatically generated folder during build and test execution:
   - Contains compiled classes and test execution results.

7. **TestReports**
   Designated folder for storing test execution logs and reports:
   - Includes HTML/XML reports generated by TestNG or advanced tools like ExtentReports.

8. **Screenshot:**
   This folder is used for storing screenshots captured during test execution:
   - Screenshots can be taken based on requirements, particularly for failed tests or specific test validation points.
   - Maintaining test evidence through screenshots helps in debugging issues and provides visual proof for stakeholders.

2.4 Code review and maintenance

Code Review

1. **Functionality**: Ensure the code achieves its intended purpose and meets all requirements.

2. **Readability**: Code should be clean, easy to read, and follow established naming conventions and formatting guidelines.

3. **Test Coverage**: Verify that there are sufficient test cases covering positive, negative, and edge scenarios.

4. **Security**: Identify and address potential vulnerabilities (e.g., handling of sensitive data, injection attacks).

5. **Performance**: Check for efficiency and ensure the code does not introduce unnecessary bottlenecks.

6. **Coding Standards**: Ensure compliance with industry or project-specific coding guidelines.
   Example:
   - Class and method names should follow the camelCase naming convention (e.g., calculateTotalAmount()), while constants should be written in uppercase with underscores (e.g., MAX_RETRY_COUNT).
   - Use clear and meaningful names for variables and methods to improve readability.
   - Include meaningful comments in the code to explain complex logic, methods, or important decisions. This enhances understanding and helps future developers easily navigate the codebase.

7. **Version Control**:
   - Validate proper use of Git, including atomic commits with clear messages.
   - Check pull requests for concise, well-documented changes and proper branch usage (e.g., feature or bugfix branches).

8. **Feedback**: Offer constructive feedback for improvements and ensure developers address all comments before merging.

Maintenance

1. **Code Update**: Regularly updating the code to add new features or fix bugs.

2. **Tool Update**: Keeping development tools up-to-date to ensure compatibility and leverage new features.
   Example: The development team updates their Java Development Kit (JDK) from version 11 to version 17 to take advantage of new language features and performance improvements.

3. **Maven Clean/Build**: Cleaning and building the project using Maven to ensure it compiles correctly.

4. **Code Clean Up**: Periodically cleaning up the code to remove unnecessary elements and improve readability.
   Example: A developer refactors the codebase to remove unused imports, redundant code, and outdated comments. They also improve variable names and add documentation to make the code more readable and maintainable.

5. **Continuous Integration (CI)**: A process where code changes are automatically tested and integrated into the main code base. This ensures that new changes do not break the existing code.

6. **Periodic Execution, Monitoring, and Reporting**: Regularly running tests, monitoring the system for issues, and generating reports to track the system's health and performance.

3. Conclusion

This documentation provides a clear overview of UI automation concepts and builds upon them by focusing on the TestNG framework. It outlines how TestNG's advanced features like annotations, parameterization, listeners, and parallel execution make testing efficient, reliable, and scalable. Combined with a well-structured folder setup and reusable components, the framework simplifies the testing process while maintaining adaptability. Regular code reviews and maintenance ensure it stays updated, supporting long-term success in handling complex UI automation tasks.