Journal of Advances in Developmental Research (IJAIDR)

E-ISSN: 0976-4844 ● Website: www.ijaidr.com ● Email: editor@ijaidr.com

Automating Workday Advanced Compensation Testing with Selenium WebDriver and Cucumber Framework

Rasika Patil

Rasika.10p@gmail.com

Abstract

This white paper presents a comprehensive approach to automating the testing of Workday Advanced Compensation using the Selenium WebDriver and Cucumber framework. As enterprises increasingly rely on Workday for Human Capital Management (HCM) and compensation management, automating the testing of Workday's Advanced Compensation (AC) module becomes crucial to ensure accuracy, compliance, and functionality.This document outlines the rationale for automation, describes the technical framework, and provides a structured implementation plan. These open-source tools, when used in combination, provide an efficient, scalable, and maintainable approach to testing Workday's AC module.

1. Introduction

1.1 Problem Statement

Manual testing of Workday Advanced Compensation is time-consuming and can result in delayed releases and increased costs. Additionally, manual tests are often insufficient to cover the breadth of scenarios necessary for comprehensive validation, leading to potential business risks.

This white paper aims to provide a detail of the framework for implementing automated testing for Workday Advanced Compensation. By leveraging Selenium WebDriver for browser automation and Cucumber for behavior-driven development (BDD), teams can create robust, maintainable tests that align closely with business requirements.

1.2 What is Advanced Compensation?

Advanced Compensation is an event in Workday launched for a specific length of time for the purpose of recommending, reviewing, and awarding compensation changes, bonus payments and/or stock grants. Most organizations refer to these activities as their compensation cycle. Some functionality and common business practices within Workday are geared toward an annual compensation process; however, many companies execute compensation review cycles quarterly or monthly for recurring processes like bonus and stock.

A compensation administrator typically kicks off this process, during which managers and HR Partners enter and approve compensation recommendations for eligible employees. Upon completion, the award recommendations are associated with the employee records automatically and are subsequently picked up by payroll for processing. [(p. 1)]

1.3Automation Testing

In this era of continuous innovation, when platforms like Workday are incorporating new applications, test automation always ensures that your critical business processes work. Test automation not only eliminates the human element to improve accuracy but also enhances the predictability in testing timelines. With test automation, by validating business processes, you can achieve significant benefits when it comes to embracing upgrades. Automation also has the potential to bring down costs significantly. Some of the biggest perks related to automation testing is that enterprises can achieve maximum accuracy along with accelerated results and broader test coverage

2. Tools Overview

2.1. Selenium WebDriver

Selenium WebDriver is a widely used open-source tool for automating web applications. It interacts directly with the browser, simulating user actions such as clicks, text entry, and navigation. In the context of Workday AC testing, Selenium WebDriver can be used to test user interfaces and workflows that involve navigating through the compensation module, creating compensation plans, adjusting salaries, and generating reports.

Key benefits of Selenium WebDriver:

- Supports multiple browsers (Chrome, Firefox, IE, etc.).
- Can be integrated with various testing frameworks and tools.
- Allows for creating reusable and maintainable test scripts.

2.2. TestNG

TestNG is a testing framework inspired by JUnit but designed to overcome its limitations. It is a powerful tool for running automated tests, especially in the context of Java-based applications. TestNG provides features such as parallel test execution, grouping of tests, and detailed reporting, which are essential for executing large test suites efficiently.

Key benefits of TestNG:

- Supports parallel execution of test cases.
- Allows flexible configuration through annotations.
- Generate detailed test reports and logs.
- Easy integration with other tools like Selenium.

2.3. Cucumber Framework

Cucumber is a tool for behavior-driven development (BDD), allowing tests to be written in a human-readable format (Gherkin syntax). This makes it easy for non-technical stakeholders to understand the test cases and collaborate in the testing process. In Workday AC testing, Cucumber can be used to define the business rules, such as salary adjustments or compensation plan creations, in a way that testers, business analysts, and HR professionals can all contribute to the process.

Key benefits of Cucumber:

- Enables collaboration between business and technical teams.
- Uses Gherkin syntax, which is easy to read and write.
- Integrates seamlessly with Selenium WebDriver for functional testing.

Benefits of the Framework

- **Increased Test Coverage**: Automation enables the execution of a larger number of test cases across various scenarios.
- **Faster Feedback Loops**: Automated tests can run quickly and provide immediate feedback on the state of the application.
- **Reusability and Maintenance**: Test scripts can be reused across different test cycles and updated as necessary with minimal effort.

3. Automation Strategy for Workday Advanced Compensation Testing

3.1 Test Automation Framework Architecture

- **Install Java Development Kit (JDK)**: Required for running Selenium and Cucumber.
- **Set Up Maven**: Use Maven for dependency management.
- **Install Selenium WebDriver**: Add the WebDriver dependency to the pom.xml file.
- **Install Cucumber**: Include Cucumber dependencies in the pom.xml.
- **Install TestNG**: Include TestNG dependencies in the pom.xml
- **Page Object Model (POM)** – A design pattern used to create object-oriented classes that serve as an interface to the web pages, helping in maintaining a clean and reusable test code.In Page Object Model, consider each web page of an application as a class file. Each class file will contain only corresponding web page elements. Using these elements, testers can perform operations on the website under test.
- **Page Factory** - It is a class provided by Selenium WebDriver to support Page Object Design patterns.AjaxElementLocatorFactory is a lazy load concept in Page Factory. This only identifies web elements when used in any operation or activity. The timeout of a web element can be assigned to the object class with the help of the AjaxElementLocatorFactory.[(p. 2)]
- **Integration with CI/CD** – The test automation suite can be integrated into a continuous integration/continuous deployment (CI/CD) pipeline to ensure automated testing in every build cycle. We used Jenkins to integrate our project. Since this automation suite contains sensitive compensation data, any CI/Cd tool needs to be used with utmost security scrutiny.

3.2 Define Project Structure

Defining a structured project landscape helps with easy navigation and maintenance of the code. Things to consider while defining project structure aresegregating codeinto modules like calculation modules and pages modules, design of the framework and test reporting. For example, my project defined the structure as below:

Src/main/java

- *com.<companyname>.<team>.hr.driver*
- *com.<companyname>.<team>.hr.helper*
- *com.<companyname>.<team>.hr.model*
  - *Calc.java*
  - *Compensation.java*
  - *JobFamily.java*
- *com.<companyname>.<team>.hr.pages*
  - *CommonPage.java*
  - *GridPage.java*
  - *homepage.java*
  - *LettersPage.java*
  - *LoginPage.java*
  - *OutsideGidpages.java*
- *com.<companyname>.<team>.hr.utils*
  - *Convert.java*
  - *DataIdentifier.java*
  - *Environment.java*
  - *Events.java*
  - *Excel.java*
  - *Global.java*
- *com.<companyname>.<team>.test.listener*

Src/test/java

- *com.<companyname>.<team>.hr.hook*
- *com.<companyname>.<team>.hr.stepdefinitions*
  - *GeneralWDStps.java*
  - *GridStps.java*
  - *HomeSteps.java*
  - *LoginSteps.java*
  - *UserDefinedCapabilities.java*
- *com.<companyname>.<team>.hr.test*
  - *TestRunner.java*

Src/test/resources

Features

Walkthrough of few packages:

Under "hr.model", we have defined classes to handle calculation of bonus, salary after merit increase and salary and grade/level changes after promotion. Likewise, compensation.java contains code related to grades and job family has code define related to job families.

```
package testRunner;

import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;
import cucumber.api.CucumberOptions;
import cucumber.api.testng.CucumberFeatureWrapper;
import cucumber.api.testng.TestNGCucumberRunner;

@CucumberOptions(
   features = "src/test/java/FeatureFile",
   glue = {"stepDefinitions"},
   tags = {"@SmokeTest"},
   format = {
              "pretty",
              "html:target/cucumber-reports/cucumber-pretty",
              "json:target/cucumber-reports/CucumberTestReport.json",
              "rerun:target/cucumber-reports/rerun.txt"
        },
   plugin = "json:target/cucumber-reports/CucumberTestReport.json")
```

```
public class TestRunner {

   private TestNGCucumberRunner testNGCucumberRunner;

   @BeforeClass()
   public void setUpClass() throws Exception {
      testNGCucumberRunner = new TestNGCucumberRunner(this.getClass());
   }

   @Test(dataProvider="features")
    public void feature(CucumberFeatureWrapper cucumberFeature)      {
      testNGCucumberRunner.runCucumber(cucumberFeature.getCucumberFeature());
   }

   @DataProvider
   public Object[][] features()
   {
      return testNGCucumberRunner.provideFeatures();
   }

   @AfterClass
   public void tearDownClass() throws Exception
   {
      testNGCucumberRunner.finish();

   }
}
```

Under “hr.test”, we have defined TestRunner.java which in our runner file, we have CucumberOptions annotation in which we have defined the paths of our feature and step definition file along with the tags name and HTML reporting Plugin.

To integrate Cucumber framework with Java TestNG framework, we have used a predefined class here called “TestNGCucumberRunner” which provides various methods such as runCucumber, provideFeatures, finish etc. “CucumberFeatureWrapper” used in a parameter of Test method is an interface for making TestNG report more descriptive.

3.3 Identify Test Cases and Requirements

Begin by identifying the critical business processes and compensation rules that need testing. For example:

- Salary adjustments based on performance for all grades and locations.
- Bonus calculations covering all plans eligible for Bonus
- Promotion requirements based on performance and potential real time scenarios
- Equity ranges and identifying eligible population for equity
- Creating test data set for complex scenarios involving parallel rules
- Reporting requirements for the project

3.4 Create Feature Files with Cucumber

Write the test cases in Gherkin syntax using Cucumber, which defines the expected behavior of the compensation process in a non-technical language. For example:

```
Feature: Bonus Grid Validation

  Background:
    Given Login with security admin
    Then I should see inbox

  @BonusValidations @AIPgrade15FTSalary
  Scenario Outline: IBPPreset
    When proxy as Manager <Proxy>
    Then I should see an email for Rewards cycle <Proxy> and <type>
    When Pick an employee <emp> with <type>
    When Given the IBP <guideline>
    When Give the award <awardtype> as <value>
    When eligible for Bonus
    Then validate individual target amount
    Then validate bonus pay amount

  Examples:
    | Proxy  | emp    | type | guideline          | awardtype | value |
    | '00123'| '00234'| 'AIP'| 'Preset: ↓ %'      | 'Bonus'   | '110' |
```

Feature file walkthrough

Here we use Gherkin's keywords to define our scenario in simple language. Gerkin syntx/keywords are:

- **Feature**: Describes the overall feature being tested — "Bonus Grid Validation."
- **Background**: Provides setup steps that are common for each scenario.
- **Scenario Outline**: Defines a template for the scenario, where <Proxy>, <emp>, <type>, <guideline>, <awardtype>, and <value> are placeholders.
- **Examples**: Provides the actual data that will be substituted into the scenario outline.

3.5 Implement Step Definitions and code

Create Java classes in the step definitions package to map Gherkin steps to code. Each method should perform actions using Selenium WebDriver.

```
@When ("^ I open the inbox email$")
public void i_open_the_inbox_email() throws Throwable {
}

@Then("^ I should see a email for Rewards cycle '(.+)' and '(.+)'$")
public void i should_see_a_email (String proxy, String type) throws Throwable{

        String cyclename = "";
        if type. contains ("test")){
                cyclename = configprop.getProperty("TestcycleName");
                System. out. println("Test cycle name:"+ cyclename);
        }else {
                cyclename = configprop.getProperty("cycleName");
        }

        proxy = DataIdentifier.ManagerId!=null?DataIdentifier.ManagerId:proxy;

        boolean chkInbox=homePage. checkInbox(cyclename, proxy) ;
        homePage. gridFullScreen();
        homePage. editDirects();
        assertTrue(chkInbox, "Inbox email verified");
}
```

Step Definition walkthrough

Step Definitions are the glue between the Gherkin feature files and the underlying code that performs the actions described in those scenarios. Each Step in a Gherkin scenario (e.g., Given, When, Then etc) corresponds to a Step Definition in a java. These step definitions contain the code that executes the actions described in the feature file.When Cucumber runs the feature file, it looks for matching step definitions and executes the associated code.

3.6 Execute Tests and Reporting

Run the TestNG suite, which will trigger the Cucumber feature files and execute the Selenium-based steps for defined scenarios and carry out tests.

4. Benefits of Test Automation in Workday Advanced Compensation

Benefits of the Framework

- **Increased Test Coverage**: Automating test cases ensures that all key business processes and edge cases are tested, providing more comprehensive coverage than manual testing.
- **Faster Feedback Loops**: Automated tests can run faster than manual tests, especially in repetitive scenarios. This enables faster feedback during development cycles.

- **Reusability and Maintenance**: Test scripts can be reused across different test cycles and updated as necessary with minimal effort.
- **Scalability:** Automated tests can be easily scaled to accommodate more test cases as the system grows, providing long-term value for large-scale implementations of Workday.

5. Challenges and Considerations

- **Dynamic UI Elements:** Workday's UI is complex and dynamic, making it challenging to create reliable locators. Strategies like XPath or CSS selectors need to be frequently updated.
- **Data Management**: Test data setup and teardown processes must be managed carefully to ensure consistency.
- **Access to Workday**: Automated testing of Workday requires access to test environments i.e Workday Sandbox or implementation tenants with appropriate data. It may also need integration with third-party systems.
- **Security**: Ensure that sensitive information is handled securely during test execution since it handles compensation data for the whole organization.

6. Conclusion

Automating the testing of Workday's Advanced Compensation module with Selenium WebDriver, TestNG, and Cucumber Framework offers significant benefits in terms of efficiency, scalability, and maintainability. By following the strategies outlined in this white paper, organizations can create a robust automation suit that ensures the accuracy of compensation-related business processes, reduces testing time, and accelerates the release cycle of Workday implementations.

Automation will continue to play a critical role in maintaining the reliability and functionality of complex systems like Workday, ensuring that compensation-related decisions are both accurate and compliant.

References:

[1] Workday Community and documentationhttps://www.workday.com(accessed Oct 2019)

[2] L Sharma/ "Page Object Design Pattern with Selenium PageFactory in Cucumber". https://toolsqa.com/. https://toolsqa.com/selenium-cucumber-framework/page-object-design-pattern-with-selenium-pagefactory-in-cucumber/.(accessed Oct 2019)