Comeon-WebAutomation

Automation Solution Document

Charu Sachdeva

Purpose

This document provides the high-level design overview of automation testing assignment for snabbare and hajper white labels that Comeon operates. Both websites allow user to play slot games, live casino and do sport betting. User can play for fun without money or insert money and do real betting.

This document is also intended to help reviewer to run the testing framework for selected flows of both websites.

Design Considerations

The architecture and design of the solution is catering following key considerations:

- I have implemented the assignment with the structured framework
- I have structured the code from a maintainability and reusability perspective

Technology Stack

All the frameworks and technologies used are well established and have extensive industry wide acceptance. Automation suite for web applications is built upon below technology stack.

Area	**Technology Used**
IDE	Intellij Idea
Build Tool	Maven
Language	JAVA 8
Automation Tool	Selenium (WebDriver 3)
API testing	Rest Assured
Framework for Tests	Junit 5
Logger	Log4j2
Reporter	maven-surefire

Advantages of using Selenium

Selenium is an open-source automation testing tool which is used for automating tests carried out on different web-browsers. It has lot of features, listing few below here:

- Language and Framework Support
- Open Source Availability
- Multi-Browser Support
- Support Across Various Operating Systems
- Reusability and Integrations

Advantages of using Rest Assured

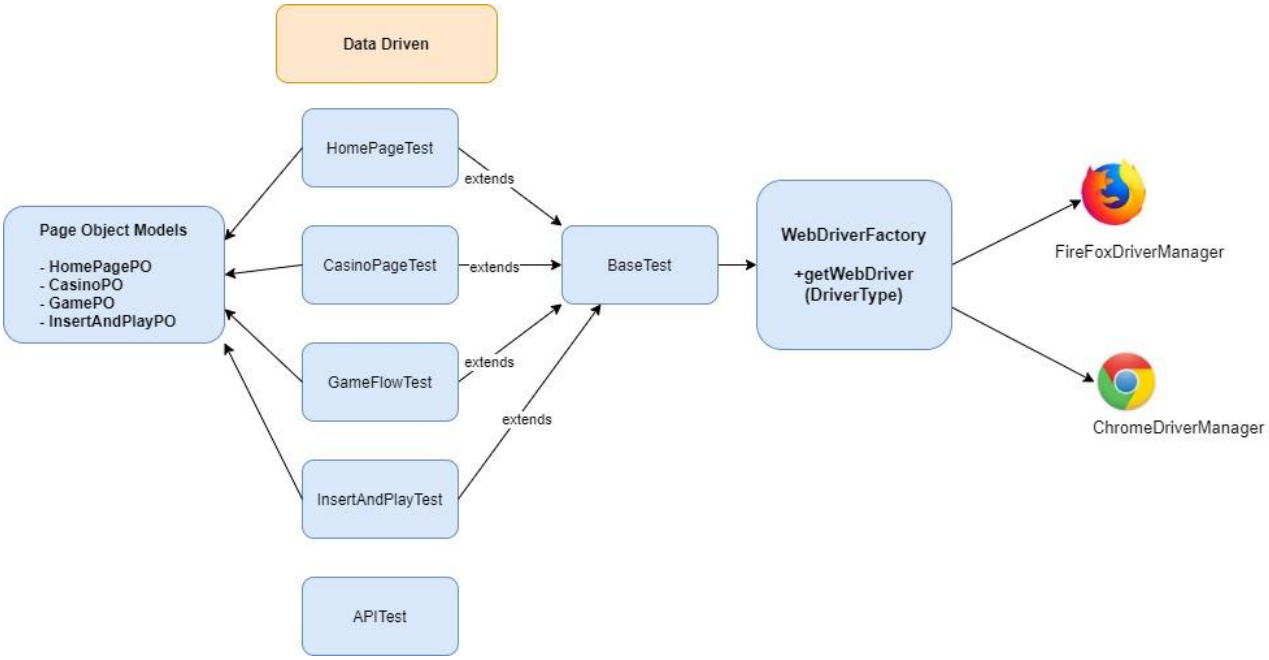
Rest Assured is an open source Java-based Domain-Specific Language (DSL) that allows you to write powerful, readable, and maintainable automated tests for your RESTful APIs.

With the help of this library:

- It removes the need for writing a lot of boilerplate code required to set up an HTTP connection, send a request and receive and parse a response
- It supports a Given/When/Then test notation, which instantly makes your tests human readable
- It helps in writing clean code and maintaining a clear separation of concerns within a test, thus leading to very readable test.

Solution Architecture

This section explains the system architecture of the application.



Page Object Models

Page Object model is an object design pattern in Selenium, where web pages are represented as classes, and the various elements on the page are defined as variables on the class. All possible user interactions can then be implemented as methods on the class.

Data Driven

Data Driven framework is focused on separating the test scripts logic and the test data from each other. Allows us to create test automation scripts by passing different sets of test data. This has been achieved by using Junit's Parametrized test. The test data set is kept in the csv files in resources folder. HomePageTest, CasinoPageTest, GameFlowTest and InsertAndPlayTest are the sample test classes.

WebDriverFactory

WebDriverFactory is used to create WebDriver for various browsers.

Driver Managers

I have used two different driver managers, FirefoxDriverManager and ChromeDriverManager, which return instance of WebDriver for respective browser type.

Code Description

Automation testing application has following packages.

Component	Description
logs	This folder contains generated log files
drivermanager	This folder will contain classes of WebDriverManager for various browsers
enums	This folder contains enum for various browsers (used while creating the WebDriver)

factory	This folder contains factory for creating WebDriver
pageobjects	This folder contains page object models having all web element locators
util	This folder contains utility classes for giving waits and loading properties from property file
tests	This folder contains all tests written in jUnit5
test.resources	This folder contains properties files for application and log4j2. Also, all test data files.
target.site	This folder contains final report after test run (surefire-report.html)

Challenges faced

Following challenges were faced by me while creating this assignment

- Automating the responsive websites for mobile devices using Selenium Webdriver
    - Only ChromeDriver has support for mobile emulation to run tests for mobile devices
    - Solution: Resize the viewport for desktop and mobile by setting Dimension in driver window size. Done with help of Parametrized test
- Asynchronous behavior of few elements
    - Solution: Used Implicit and Explicit waits
- Different class in DOM for desktop and mobile view (three dots link of the slot game)
- Faced ElementNotInteractableException for selecting Digipass radio option
    - Solution: Used outer WebElement instead of inner radio input field
- Loading of the game is very slow so had to use long explicit wait there
- Element (Love icon on the game) not clickable due to scroll in mobile view (faced only in mozilla)
    - Solution: Used JavascriptExecutor for clicking it

Testing the application

Prerequisite

You should have jdk8 and maven installed for running this automation application

You should have any of following browsers (support for other browsers can be added in future):

- Chrome
- Mozilla

All configurable properties have been defined in the application.properties file for ease of updating.

Running tests

The solution can be tested by running the “mvn test” command in the project main folder where pom.xml is present.

Running tests and Reporting

The solution can be tested by running the “mvn surefire-report:report” command in the project main folder where pom.xml is present.

The path to the unit test report is “/target/site/surefire-report.html”

Possible Improvements for future

There is scope of many improvements in this assignment, few things which could be improved:

- Cover all testing scenarios
- Add support for other browsers
- Cover all API’s and all flows in API testing
- Better mobile testing (example using Galen framework)
- Better reporting
- Continuous Integration and Continuous Deployment for the application