Regression testing in selenium

[Regression Testing](https://www.browserstack.com/guide/regression-testing" \o "Regression Testing: A Detailed Guide" \t "_blank) is a kind of testing that is done to check the behavior of an application after a  new functionality has been introduced or bug fix has been implemented. It checks whether the new functionality is not affecting the existing application behavior.

### **Regression Testing with Selenium**

[Selenium](https://www.browserstack.com/selenium) is a web-based automation testing framework. It helps in automating functional and regression test cases that reduce the manual testing effort. Usually, regression suites include a huge number of test cases and it takes time and effort to execute them manually every time when a code change has been introduced. Hence almost every organization looks after automating regression test cases to reduce the time and effort. Choosing the right automation framework/ tool completely depends upon the application, technology used, testing requirements, and skill sets required for performing automation testing.

There are four components in Selenium – [Selenium Webdriver](https://www.browserstack.com/guide/selenium-webdriver-tutorial), [Selenium IDE](https://www.browserstack.com/guide/what-is-selenium-ide), [Selenium RC](https://www.browserstack.com/guide/selenium-rc-tutorial), and [Selenium Grid](https://www.browserstack.com/guide/selenium-grid-tutorial). Each of these is used for different testing purposes. Selenium Webdriver provides an interface that helps us develop automation scripts that interact with the browser and perform. Various browsers like Chrome, Edge, Firefox, IE, and Opera are supported by Selenium. Selenium also supports multiple programming languages like [Java](https://www.browserstack.com/guide/selenium-with-java-for-automated-test), [Python](https://www.browserstack.com/guide/python-selenium-to-run-web-automation-test), [Javascript](https://www.browserstack.com/guide/automation-using-selenium-javascript), Ruby, etc.

Let’s see some best practices that should be considered for regression testing.

* Defining Test Strategy: The test strategy defined may include the test cases to be considered for regression, estimates for test execution enhancements required to the existing test cases, and the new test cases if required.
* Maintaining/updating Regression suites: Testing teams have to regularly maintain the regression suites to check for any new failures, test script enhancements required, etc.
* Test Automation: Automating regression tests is a best practice to save the time and efforts required to execute regression tests manually every time during a release. There are multiple approaches for automating test cases like the one mentioned above using Selenium. Selenium can be used along with the Page object model (POM) design pattern, Data-driven, keyword-driven frameworks, etc.

### **How to Perform Regression Testing Using Selenium?**

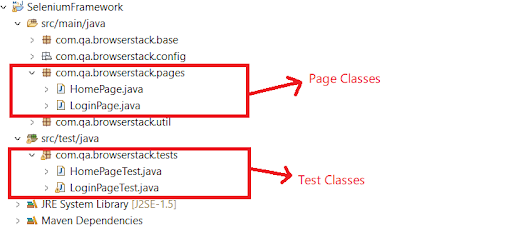
Automation completely depends on the framework that you choose to develop, and there is no such tool dedicated to performing only regression testing. The automation framework you select should be designed such that it supports regression testing effectively.

You can develop the regression suite for automation and keep adding new test scripts/test cases as and when required. Selenium Framework contains many reusable modules/functions that make it easy to maintain the existing code or add any new code.

You can integrate Selenium with TDD frameworks like TestNG, Junit Maven, etc. [TestNG annotations](https://www.browserstack.com/guide/testng-annotations-in-selenium) help in writing automation scripts effectively. You can also use the  [Page Object Model](https://www.browserstack.com/guide/page-object-model-in-selenium-python" \o "Page Object Model and Page Factory in Selenium Python" \t "_blank) design pattern while building an automation framework.

The page object model is a design pattern that makes it easy to maintain code, reduces complexity, and increases code reusability. In POM there is a separate class for each application web page. In these page classes, there are page objects and corresponding methods that implement these page objects while interacting with the browser.

Also, there are separate Test classes in which you can write your test cases using TestNG or Junit. You can also add assertions and verifications in your Test classes. The fact that verifications are separated from our page operations in page classes makes POM easy to understand and simplified.

Let’s see the below framework structure using POM:

In the above structure, there are two Page classes – HomePage and LoginPage. Similarly, there are two corresponding test classes – HomePageTest and LoginPageTest.

LoginPage class

package com.qa.browserstack.pages;

import java.time.Duration;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import com.qa.browserstack.base.BasePage;

public class LoginPage extends BasePage {

WebDriver driver;

By emailID = By.id("user\_email\_login");

By password =  By.id("user\_password");

By SignIn = By.cssSelector("li.sign-in-link>a");

By Login = By.id("user\_submit");

By checkBox = By.id("tnc\_checkbox");

public LoginPage(WebDriver driver)

{

this.driver = driver;

}

public String getLoginPageTitle()

{

return driver.getTitle();

              }

public void doLogin(String username,String pwd) {

driver.findElement(SignIn).click();

driver.findElement(emailID).sendKeys(username);

driver.findElement(password).sendKeys(pwd);

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(20));

driver.findElement(Login).click();

}

public Boolean signInLinkIsDisplayed()

{

boolean signIn;

signIn = driver.findElement(SignIn).isDisplayed();

return signIn;

}

}

In this Page class, page objects like emailID, password, signIn are designed first and then there are corresponding methods like getLoginPageTitle,doLogin,signInLinkIsDisplayed that implement these page objects to interact with the browser.

LoginPageTest class

package com.qa.browserstack.tests;

import com.qa.browserstack.base.BasePage;

import com.qa.browserstack.pages.LoginPage;

import static org.testng.Assert.assertEquals;

import java.net.MalformedURLException;

import java.net.URL;

import java.util.Properties;

import org.openqa.selenium.By;

import org.openqa.selenium.Platform;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.remote.DesiredCapabilities;

import org.openqa.selenium.remote.RemoteWebDriver;

import org.testng.Assert;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.AfterTest;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

import com.qa.browserstack.util.Constants;

public class LoginPageTest {

BasePage basePage;

Properties prop;

WebDriver driver;

LoginPage loginPg;

@BeforeTest

public void setUp() throws Exception

{

basePage = new BasePage();

prop = basePage.init\_properties();

driver =  basePage.init\_driver(prop);

loginPg =  new LoginPage(driver);

}

@Test(priority = 3)

public void LoginTest() throws Exception

{

loginPg.doLogin(prop.getProperty("username"), prop.getProperty("password"));

}

@Test(priority = 2)

public void LoginPageTitleTest()

{

String title = loginPg.getLoginPageTitle();

System.out.println(title);

Assert.assertEquals(title, Constants.LOGIN\_PAGE\_TITLE);

}

@Test(priority = 1)

public void SignupLinkTest()

{

Assert.assertTrue(loginPg.signInLinkIsDisplayed());

}

@AfterTest

public void tearDown()

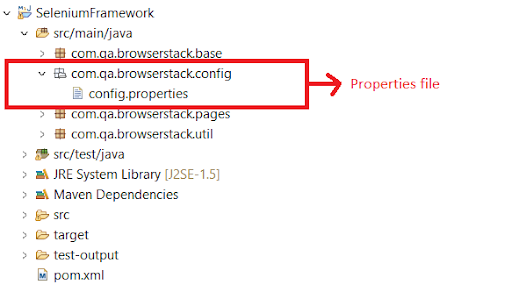
{

driver.quit();

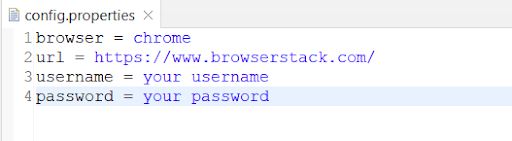
}

}

The above test cases are written using TestNG. Through these test cases, you can call the page class methods like doLogin,getLoginPageTitle, etc. You can also maintain the data in the properties file as shown below.



**config.properties**



Properties file plays a crucial role within the automation framework and helps to implement regression testing effectively. The properties file consists of key and value pairs which we require while executing our main automation test scripts. This way, you just have to update the value of any key if required, and no major code change is required.