**Features & Capabilities of the Framework**

1. **Java**– It uses Java programming language.
2. **TestNG**– It uses TestNG as a testing framework. You can learn more about TestNG from our
3. **Maven-based** – It will be maven-based. So all the dependencies will be in a POM file and the test suite can be triggered using maven commands.
4. **Hybrid Framework** – It will be a hybrid framework with a combination of the modular and data-driven framework.
5. **Page Object Model** – The framework will use the Page object Model design pattern in Selenium.
6. **Page Factory** – It will use page factory implementation of the page object model in Selenium.
7. **Screenshot on failure** – The framework will have the capability to capture screenshots in case of failed tests.
8. **Test data in Excel** – The framework will have a utility class that will read test data from an excel file.
9. **Logging**– Log4j is used for logging.

## Structure of the framework

### **TestBase.java inside Base Package**

This is the base file that performs set-up and tear-down operations like –  browser configurations, implicit and explicit waits handling, cookies deletion, etc. Each test class must extend this class.

### **Page Classes inside Pages Package**

The ‘pages’ package contains all the page classes. Each page class contains the web elements and actions that can be performed on those classes.

### **Test Classes inside Test Package**

The ‘test’ package contains all the test classes. Each test class extends the TestBase.java class and contains the test scripts.

### **Util Package**

In the util package, we can have all the utilities e.g. in this framework, I have an ExcelUtil file that reads data from an excel file and converts it into 2D array.

### **Resources folder**

Inside the resources folder, we can have different configuration files like log4j2.xml file.

### **Logs folder**

The logs folder contains all the log files generated while running the test scripts.

### **TestData folder**

The testData folder contains the test data for the test scripts e.g. we can have credentials and other sets of test data that are used by our test scripts.

### **Pom.xml file**

The pom.xml contains all the dependencies used in the project.

### **TestNG.xml**

TestNG.xml file contains the testNG configurations using which we can run a particular method, group, or test class and at the same time configure the tests to run in parallel.

**Code**

|  |
| --- |
| import java.io.File; |
|  | import java.io.FileInputStream; |
|  | import java.io.IOException; |
|  | import java.io.InputStream; |
|  | import java.lang.reflect.Method; |
|  | import java.util.Arrays; |
|  | import java.util.Properties; |
|  | import java.util.concurrent.TimeUnit; |
|  |  |
|  | import org.apache.commons.io.FileUtils; |
|  | import org.openqa.selenium.OutputType; |
|  | import org.openqa.selenium.TakesScreenshot; |
|  | import org.openqa.selenium.WebDriver; |
|  | import org.openqa.selenium.chrome.ChromeDriver; |
|  | import org.openqa.selenium.chrome.ChromeOptions; |
|  | import org.openqa.selenium.firefox.FirefoxDriver; |
|  | import org.openqa.selenium.ie.InternetExplorerDriver; |
|  | import org.openqa.selenium.support.ui.WebDriverWait; |
|  | import org.testng.ITestResult; |
|  | import org.testng.annotations.AfterMethod; |
|  | import org.testng.annotations.AfterSuite; |
|  | import org.testng.annotations.BeforeMethod; |
|  | import org.testng.annotations.BeforeSuite; |
|  |  |
|  | import io.github.bonigarcia.wdm.WebDriverManager; |
|  |  |
|  |  |
|  | /\*\* |
|  | \* The TestBase class is the base class to fetch environment specific configuration parameters from |
|  | \* Jenkins/Maven. Based on the parameters, it performs the browser setup and tear-down functions. |
|  | \* |
|  | \* @author Kuldeep Rana |
|  | \*/ |
|  |  |
|  |  |
|  | public class TestBase { |
|  |  |
|  | protected static WebDriver driver; |
|  | public static Properties envConfig; |
|  | WebDriverWait wait; |
|  |  |
|  |  |
|  | //Environment value fetched from POM with 'careersIn' and 'production' being the valid values |
|  | public static final String ENV = System.getProperty("env", "Production"); |
|  |  |
|  | //BROWSER value fetched from POM with Chrome being the default value |
|  | private static final String BROWSER = System.getProperty("browser", "Chrome"); |
|  |  |
|  |  |
|  | //Automation suite setup method to configure and instantiate a particular browser |
|  | @BeforeSuite |
|  | public void suiteSetup() throws Exception { |
|  |  |
|  | //Browser configuration - can add more browsers and remote driver here |
|  | if (BROWSER.equals("Firefox")) { |
|  | WebDriverManager.firefoxdriver().setup(); //can also use set property method for browser executables |
|  | driver = new FirefoxDriver(); |
|  | } |
|  | else if (BROWSER.equals("Chrome")) { |
|  | WebDriverManager.chromedriver().setup(); |
|  | ChromeOptions options = new ChromeOptions(); |
|  | options.addArguments("--disable-notifications"); |
|  | driver = new ChromeDriver(options); |
|  | } |
|  | else if (BROWSER.equals("IE")) { |
|  | WebDriverManager.iedriver().setup(); |
|  | driver = new InternetExplorerDriver(); |
|  | } |
|  | else { |
|  | throw new RuntimeException("Browser type unsupported"); |
|  | } |
|  |  |
|  | //Setting implicit wait |
|  | driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS); |
|  |  |
|  | driver.manage().window().maximize(); |
|  |  |
|  | //Setting WebDriverWait with max timeout value of 20 seconds |
|  | wait = new WebDriverWait(driver, 20); |
|  |  |
|  | //Environment specific properties file loading |
|  | InputStream configFile = new FileInputStream(System.getProperty("user.dir") + |
|  | "\\src\\test\\java\\com\\artoftesting\\config\\" + ENV + ".properties"); |
|  | envConfig = new Properties(); |
|  | envConfig.load(configFile); |
|  |  |
|  | } |
|  |  |
|  |  |
|  | @BeforeMethod() |
|  | public void loadBaseUrl(Method method) { |
|  | driver.get(envConfig.getProperty("baseUrl")); |
|  |  |
|  | } |
|  |  |
|  |  |
|  | @AfterMethod |
|  | public void screenshotAndDeleteCookies(ITestResult testResult) throws IOException { |
|  | //Taking screenshot in case of failure |
|  | if(testResult.getStatus() == ITestResult.FAILURE){ |
|  | File scrFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE); |
|  | FileUtils.copyFile(scrFile, new File("errorScreenshots\\" + testResult.getName() + "-" |
|  | + Arrays.toString(testResult.getParameters()) + ".jpg")); |
|  | } |
|  |  |
|  | //Deleting cookies |
|  | driver.manage().deleteAllCookies(); |
|  | } |
|  |  |
|  |  |
|  | @AfterSuite |
|  | public void suiteTearDown() { |
|  | driver.quit(); |
|  | } |
|  |  |
|  | } |

**Login class**

|  |
| --- |
| import org.apache.logging.log4j.LogManager; |
|  | import org.apache.logging.log4j.Logger; |
|  | import org.openqa.selenium.WebDriver; |
|  | import org.openqa.selenium.WebElement; |
|  | import org.openqa.selenium.support.FindBy; |
|  | import org.openqa.selenium.support.PageFactory; |
|  |  |
|  |  |
|  | public class SauceDemoLoginPage { |
|  |  |
|  | WebDriver driver; |
|  |  |
|  | private static final Logger log = LogManager.getLogger(SauceDemoLoginPage.class); |
|  |  |
|  | @FindBy(id="user-name") |
|  | WebElement username; |
|  |  |
|  | @FindBy(id="password") |
|  | WebElement password; |
|  |  |
|  | @FindBy(id="login-button") |
|  | WebElement loginButton; |
|  |  |
|  | public SauceDemoLoginPage(WebDriver driver) { |
|  | this.driver = driver; |
|  | PageFactory.initElements(driver, this); |
|  | } |
|  |  |
|  |  |
|  | public void setUsername(String uname) { |
|  | username.sendKeys(uname); |
|  | } |
|  |  |
|  | public void setPassword(String pwd) { |
|  | password.sendKeys(pwd); |
|  | } |
|  |  |
|  | public void clickLoginButton() { |
|  | loginButton.click(); |
|  | } |
|  |  |
|  | public SauceDemoInventoryPage login(String username, String password) { |
|  | log.info("Logging with username - " + username + " and password - " + password); |
|  | setUsername(username); |
|  | setPassword(password); |
|  | clickLoginButton(); |
|  | return new SauceDemoInventoryPage(driver); |
|  | } |
|  |  |
|  | public String getTitle() { |
|  | return driver.getTitle(); |
|  | } |
|  |  |
|  | } |