**Part 3: Test Automation**

Test Automation Framework:

* **Selected Framework: Selenium WebDriver with TestNG.**
  + Explanation: Selenium is widely used for web automation, and TestNG provides a robust testing framework with parallel execution and test grouping capabilities.
* **Overview:**
  + Use Page Object Model (POM) for a clear separation of concerns.
  + Leverage TestNG annotations for test execution control.
  + Implement Maven for dependency management.

Automated Test Scripts:

1. **User Registration:**
   * Automate positive and negative registration scenarios.

CODE:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.testng.annotations.Test;

public class UserRegistrationTest {

WebDriver driver; // Assume the WebDriver instance is initialized elsewhere.

@Test(priority = 1)

public void testValidUserRegistration() {

// Navigate to the registration page

driver.get("your\_registration\_page\_url");

// Fill in valid registration details

fillRegistrationForm("validUsername", "validPassword", "validEmail@example.com");

// Submit the form

submitRegistrationForm();

// Validate successful registration

// Add assertions based on your application's behavior

}

@Test(priority = 2)

public void testInvalidUserRegistration() {

// Navigate to the registration page

driver.get("your\_registration\_page\_url");

// Fill in invalid registration details

fillRegistrationForm("invalidUsername", "invalidPassword", "invalidEmail");

// Submit the form

submitRegistrationForm();

// Validate that registration fails

// Add assertions based on your application's behavior

}

private void fillRegistrationForm(String username, String password, String email) {

// Locate registration form elements and fill in details

WebElement usernameField = driver.findElement(By.id("username"));

WebElement passwordField = driver.findElement(By.id("password"));

WebElement emailField = driver.findElement(By.id("email"));

usernameField.sendKeys(username);

passwordField.sendKeys(password);

emailField.sendKeys(email);

}

private void submitRegistrationForm() {

// Locate and click the registration submit button

WebElement submitButton = driver.findElement(By.id("registerButton"));

submitButton.click();

}

}

1. **Product Search:**
   * Automate positive and negative search scenarios.

CODE:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.testng.annotations.Test;

public class ProductSearchTest {

WebDriver driver; // Assume the WebDriver instance is initialized elsewhere.

@Test(priority = 1)

public void testValidProductSearch() {

// Navigate to the search page

driver.get("your\_search\_page\_url");

// Perform a valid product search

performSearch("validProduct");

// Validate search results

// Add assertions based on your application's behavior

}

@Test(priority = 2)

public void testInvalidProductSearch() {

// Navigate to the search page

driver.get("your\_search\_page\_url");

// Perform an invalid product search

performSearch("");

// Validate that no results are displayed

// Add assertions based on your application's behavior

}

private void performSearch(String searchQuery) {

// Locate the search input field and submit button

WebElement searchInput = driver.findElement(By.id("searchInput"));

WebElement searchButton = driver.findElement(By.id("searchButton"));

// Enter the search query and submit

searchInput.sendKeys(searchQuery);

searchButton.click();

}

}

1. **Adding Items to Cart:**
   * Automate adding items to the cart and verify the cart's content.

CODE:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.testng.annotations.Test;

public class AddToCartTest {

WebDriver driver; // Assume the WebDriver instance is initialized elsewhere.

@Test(priority = 1)

public void testAddToCart() {

// Navigate to the product page

driver.get("your\_product\_page\_url");

// Add a product to the cart

addToCart("productID123");

// Validate the product is added to the cart

// Add assertions based on your application's behavior

}

@Test(priority = 2)

public void testAddToCartWithInsufficientStock() {

// Navigate to the product page

driver.get("your\_product\_page\_url");

// Add a product with insufficient stock to the cart

addToCart("outOfStockProduct");

// Validate that the product cannot be added to the cart

// Add assertions based on your application's behavior

}

private void addToCart(String productId) {

// Locate the "Add to Cart" button and click it

WebElement addToCartButton = driver.findElement(By.id("addToCartButton\_" + productId));

addToCartButton.click();

}

}

Test Data Management:

* Utilize data-driven testing with external data files (CSV, Excel).
* For each test scenario, maintain separate data files with input values.

### Test Data Management Methods:

import org.testng.annotations.DataProvider;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class TestDataManagement {

// Data provider method to read test data from a CSV file

@DataProvider(name = "userRegistrationData")

public static Object[][] userRegistrationData() {

return readTestDataFromCSV("path/to/user\_registration\_data.csv");

}

// Data provider method to read test data from a CSV file

@DataProvider(name = "productSearchData")

public static Object[][] productSearchData() {

return readTestDataFromCSV("path/to/product\_search\_data.csv");

}

// Method to read test data from a CSV file

private static Object[][] readTestDataFromCSV(String filePath) {

Object[][] testData = null;

try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {

String line;

int row = 0;

while ((line = br.readLine()) != null) {

String[] data = line.split(",");

if (testData == null) {

testData = new Object[data.length][data.length];

}

testData[row++] = data;

}

} catch (IOException e) {

e.printStackTrace();

}

return testData;

}

// Method to generate random username for test data

public static String generateRandomUsername() {

// Implement your logic to generate a random username

return "testUser" + System.currentTimeMillis();

}

// Method to generate random email for test data

public static String generateRandomEmail() {

// Implement your logic to generate a random email

return "testEmail" + System.currentTimeMillis() + "@example.com";

}

}

### Using Test Data in Test Scripts:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.testng.annotations.Test;

import org.testng.annotations.DataProvider;

import java.util.concurrent.TimeUnit;

public class UserRegistrationTest extends BaseTest {

@Test(dataProvider = "userRegistrationData", dataProviderClass = TestDataManagement.class)

public void testUserRegistration(String username, String password, String email) {

// Navigate to the registration page

driver.get("your\_registration\_page\_url");

// Fill in registration form using test data

fillRegistrationForm(username, password, email);

// Submit the form

submitRegistrationForm();

// Validate successful registration

// Add assertions based on your application's behavior

}

private void fillRegistrationForm(String username, String password, String email) {

WebElement usernameField = driver.findElement(By.id("username"));

WebElement passwordField = driver.findElement(By.id("password"));

WebElement emailField = driver.findElement(By.id("email"));

usernameField.sendKeys(username);

passwordField.sendKeys(password);

emailField.sendKeys(email);

}

private void submitRegistrationForm() {

WebElement submitButton = driver.findElement(By.id("registerButton"));

submitButton.click();

}

// Additional test methods...

}