3. TEST AUTOMATION BASICS

Minoli Rashmitha

WIREAPPS | Intern QA Technical Assessment

Table of Contents

Approach to Creating a Test Automation Framework for an Online Shopping Site

1. Understanding the requirements.

Objective: Automate the testing of core functionalities such as user login, registration, product search, adding to cart, checkout, and order history.

Scope: Identify the most critical areas for testing, including user flows, business logic, and user experience.

Technology Stack: Choose the technology stack that works with the application environment,

(e.g., Selenium WebDriver, TestNG/JUnit, Maven/Gradle, etc.).

2. Defining the Structure of the Framework

Framework Type: Choose a hybrid framework that combines the benefits of data-driven, keyword-driven, and modular approaches.

Key Components:

1. Test Scripts: Scripts to automate individual test cases.
2. Test Data: External data source (e.g., Excel, CSV, or database) to feed into the test scripts for data-driven testing.
3. Reusable Methods: Functions that do the same thing over and over (like login, search, add to cart, etc.).
4. Page Object Model (POM): Create a separate class for each web page in the application to encapsulate the elements and actions for each page.
5. Configuration Files: Store global settings, such as URLs, browser configurations, and environment details.

3. Tool Selection

Automation Tool: Selenium WebDriver is a popular choice for browser automation because it works well with many computer languages. (e.g., Java and Python)

Test Runner: Use TestNG or JUnit for managing test execution and reporting.

Build Tool: Use Maven or Gradle for project management and dependency handling.

Version Control: Use Git for source code management and collaboration.

CI/CD Integration: Integrate with Jenkins or other CI tools to trigger automated tests on every code commit.

4. Test Case Identification and Prioritization

Critical User Journeys: Identify the most critical user flows, such as:

1. User Registration and Login
2. Product Search and Filtering
3. Add to Cart and Checkout Process
4. Order History and Tracking

Positive and Negative Scenarios: Ensure coverage of both happy paths and edge cases (e.g., invalid logins, empty cart scenarios).

Smoke and Regression Testing: Identify a set of smoke tests for quick validation of major functionalities and regression tests for validating the impact of new changes.

5. Designing the Test Cases

Modular Test Scripts: Write test cases that are modular, allowing reusability across different scenarios (e.g., login functionality can be reused across different test cases).

Data-Driven Testing: Use external data to run the same test with different input values. This ensures comprehensive coverage of all potential scenarios.

(e.g., testing the checkout process with various payment methods).

Assertion Strategy: Clearly define success criteria for each test case using assertions (e.g., validating that the order confirmation page is displayed after checkout).

6. Handling Dynamic Elements and Synchronization

Locators Strategy: Implement a robust locator strategy, using XPath, CSS Selectors, or ID/Name attributes that are resilient to UI changes.

Explicit Waits: Use explicit waits to handle asynchronous web elements, ensuring that the scripts wait until the required elements are visible or clickable.

7. Logging and Reporting

Logging Mechanism: Implement a logging framework (e.g., Log4j) to capture detailed logs of test execution, making it easier to debug issues.

Test Reporting: Integrate Allure Reports or Extent Reports to generate detailed and visual test execution reports, including pass/fail status, screenshots on failure, and execution time.

8. Test Data Management

Test Data Source: Store test data in an external file (e.g., Excel or CSV) or a database. Ensure that sensitive information (e.g., passwords) is encrypted or stored securely.

Test Data Reset: Implement mechanisms to reset test data after each run, ensuring that tests can be run repeatedly without manual intervention (e.g., reset the cart after each checkout test).

9. Continuous Integration and Continuous Testing

CI Integration: Set up Jenkins or another CI tool to automatically run tests on code pushes, nightly builds, or any other predefined schedule.

Test Execution Pipeline: Define a pipeline that includes running smoke tests first, followed by the full regression suite. Optionally, parallelize test execution to reduce execution time.

Notifications: Configure notifications (e.g., email, Slack) for the test results to keep the team informed of any failures or issues.

10. Maintenance and Scalability

Framework Maintenance: Ensure the framework is easy to maintain and extend as the application grows. This includes keeping the code modular, using proper version control, and documenting the test cases and framework structure.

Scalability: Design the framework to handle scaling, such as increasing the number of test cases or adding support for testing across multiple browsers and devices using tools like Selenium Grid.

11. Continuous Improvement

Regular Review: Continuously review and refactor the framework to adapt to new application changes and technologies.

Code Reviews: Encourage regular code reviews for test scripts to maintain quality and consistency.

Test Optimization: Periodically optimize test cases by removing redundant or outdated ones and focusing on areas that require more in-depth testing.

12. Documentation and Training

Provide documentation and training for team members to ensure effective use and maintenance of the framework.