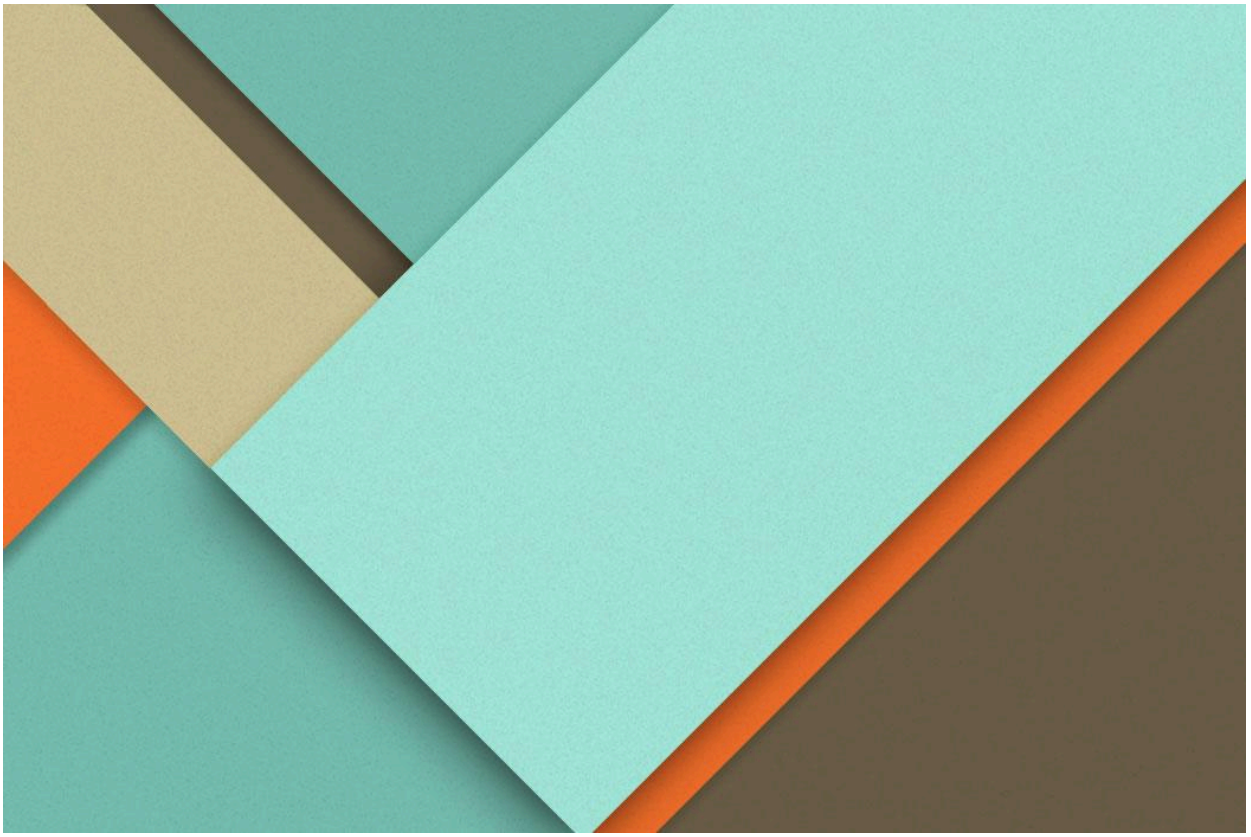


Gravity Assignment

07.24.2024

Bhavin Sangani
+91 9099921026

Requirement Analysis

To analyze requirements for testing the e-commerce platform it should include a clear understanding of the system, identification of stakeholders,scope of testing (Detailed Test Plan), and the fact that all critical functionalities are taken care of which are

I. How you would gather and analyze the testing requirements for the e-commerce platform.

- In this role, I would understand the requirement of the e-commerce product by attending meetings with all the different project stakeholders: product owners, business analysts, developers, and other QA members who are already working on that product.
- Will gather the details of the key functionalities of an e-commerce platform and any customizations that have been asked for by the client, other than what are available from a typical market e-commerce app/website.
- Also will study any existing documentation if available, such as requirement specifications, JIRA user stories to gain knowledge about system and user journey.
- Will Identify Key Functionalities and Components fo E commerce Web Application and Mobile Application along with the all APIs like core features such as user registration, login, product browsing, search/Filter/Sort functionality, shopping cart, checkout process, payment processing, order history, and user profile management and fo mobile-specific features like push notifications, device compatibility.
- Will finalize the functional, non-functional, and performance requirements..
- Based on above will strat defining Testing Scope and Objectives to know what to test and how to test and what are the entry and exit criteria and risk assessment and which type of testing will be carried out like

    - Functional Testing (End To End)
    - Integration Testing (API Suit)
    - Performance Testing (Non Functional)
    - Compatibility Testing (Browser/Device)
    - Security Testing

- Will collaborate with team and review all documented scope and requirements for testing

II. Identify the key functionalities and areas that need to be covered by automated tests.platform .

Creating an automated testing framework for an e-commerce platform involves covering a wide range of functionalities across web applications, mobile applications, and APIs so it's important to identify and prioritize key functionalities and areas that ensure a seamless and reliable user experience. (By doing impact analysis and Risk assessment)

Note: We should only take those test to automate which are repetitive in nature and first priority for automate should be prioritized based on user journeys like browsing products, adding them to the cart, and completing the purchase which should cover happy path by Simulate real user scenarios from start to finish.

Broader Level Key Functionalities and Areas for Automation

1. User Registration and Authentication

- User registration process
- Login and logout functionality
- Password reset and recovery
- Social media logins (if applicable)
- Two-factor authentication (Authentication frameworks oAuths,Bearer Token)

2. Product Management

- Product search functionality (Predictive/Normal)
- Product filtering and sorting
- Product detail page display
- Adding products to the wishlist
- Removing products from the wishlist

3. Shopping Cart

- Adding products to the cart
- Removing products from the cart
- Updating product quantities in the cart
- Cart persistence across sessions (if applicable)

4. Checkout Process

- Address selection or entry (Shipping address scenarios)
- Payment method selection
- Applying discount codes or coupons
- Order review and confirmation
- Handling different payment gateways

5. Order Management

- Order history display
- Order status tracking
- Order cancellation and returns
- Generating invoices and receipts

6. User Account Management

- Updating personal information
- Managing saved addresses
- Viewing and managing payment methods
- Notification and email preferences (Push Notifications in Mobile)

7. Performance Testing

- Load testing for product catalog
- Load testing for checkout process
- Load testing for user authentication
- Load testing scenarios for less product inventory and high demand

8. Security Testing

- SQL injection
- Authorization and Authentication
- Data encryption and secure data transfer

9. Browser/Device Compatibility

- Device compatibility Test
- Browser compatibility Test
- Can integrate test with third party cloud service BrowserStack, pCloudy, LamdaTest to achieve this.

Test Automation Strategy

To develop a comprehensive test automation strategy for the E-commerce platform we need to take approach that encompasses web, mobile and API Testing by integrating the mentioned tools and technologies.

Approach to developing an automated testing strategy.

- **Requirement Analysis and Planning**

- Clear automation goals should be defined in a way that reduces manual testing efforts and ensures better test coverage and fast feedback.
- Identify Key Functionalities and Scenarios based on the requirement analysis, prioritize and identify critical test scenarios for web, mobile, and APIs that we did above

- **Tools & Technology Selection For Automation**

- Select tool and technology for your automation by considering various factors like language supported by tool, reporting, learning curve, your team strength, Open source, community support and integration with CI/CD

- **Framework Design**

- You can create master framework which cover all WEB/Mobile and API but this is not recommended due to high maintenance and readability for new comers, so it would be better to manage three different projects if possible for all three components.
- Use data-driven framework to separate test logic from test data and actions by using external data from JSON/ CSV or DB
- Create reusable components and functions to avoid redundancy and enhance maintainability. (By Leveraging Page Object Model design pattern)
- Design a modular framework that can easily scale and adapt to changes in the application/Website (High readability and low maintenance)

- **Integration with CI/CD:**

- Integrate the automation framework with CI/CD pipelines (e.g., Jenkins, GitLab CI) to enable continuous testing and Ensure automated tests run on every code commit or on a scheduled basis.

- **Reporting and Monitoring:**

- Generate detailed reports for each test run, highlighting passed/failed tests, errors, and execution time. We can leverage Extent/Allure reports for WEb and Mobile automation and for API automation with Postman we can use Newman reporting.

- **Maintenance and Improvement::**

- Regularly update and maintain test scripts to accommodate application changes.
- Continuously gather feedback from test executions to improve and optimize the test cases and framework.

Tools & Technology Selection

A. Web Testing: Selenium WebDriver /PlayWright

- **Selenium WebDriver:** For automating browser interactions and testing web applications. (Open source and Large community support)
- **PlayWright**: with Java/JavaScript - Very popular tool this days
- **TestNG:** For test management, assertions, and generating test reports.
- **Maven:** For managing project dependencies and build automation.
- **Extent Report:** Generate detailed reports for each test run, highlighting passed/failed tests,

B. Mobile Testing: Appium

- **Appium 2.0:** For automating mobile application testing across different platforms (iOS, Android).

C. API Testing: RestAssured, Postman

- **RestAssured:** For automating RESTful API testing using Java, allowing detailed verification of API responses.
- **Postman:** For creating, managing, and executing API tests, and for manual exploratory testing.
- **Newman:** Postman's command-line tool for running Postman collections in CI/CD pipelines.

D. Performance Testing: JMeter

- **JMeter:** For performance and load testing of web applications and APIs.
- **Plugins:** Various plugins for extended functionalities, such as reporting and real-time monitoring.

Integration into a Cohesive Testing Framework

A. Framework Structure:

Organize the project into distinct modules for web, mobile, and API tests, with shared utilities and resources.

- `src/test/java/web`
- `src/test/java/mobile`
- `src/test/java/api`
- `src/test/resources (configs, Constant, Utilities, Data)`

B. Continuous Integration (CI) Setup:

- **JenkinsCI:** Configure CI servers to pull the latest code from the repository and trigger automated test execution. (Free style project)
- **Pipeline Configuration:** Define CI pipelines with stages for building the project, running tests (web, mobile, API), and generating reports. (Require DevOps team)

C. Test Execution and Reporting:

- **Parallel Execution:** Configure the framework to support parallel execution of tests to reduce overall test execution time.
- **Reporting Tools:** Use tools like Allure or ExtentReports to generate comprehensive test reports.
- **Notifications:** Integrate with communication tools (e.g., Slack, email) to send test execution summaries and alerts.

D. Version Control and Collaboration:

- **Git:** Use a version control system (Git) to manage and version test scripts.
- **Branching Strategy:** Implement a branching strategy (e.g., feature branches, pull requests) to ensure code quality and facilitate collaboration.

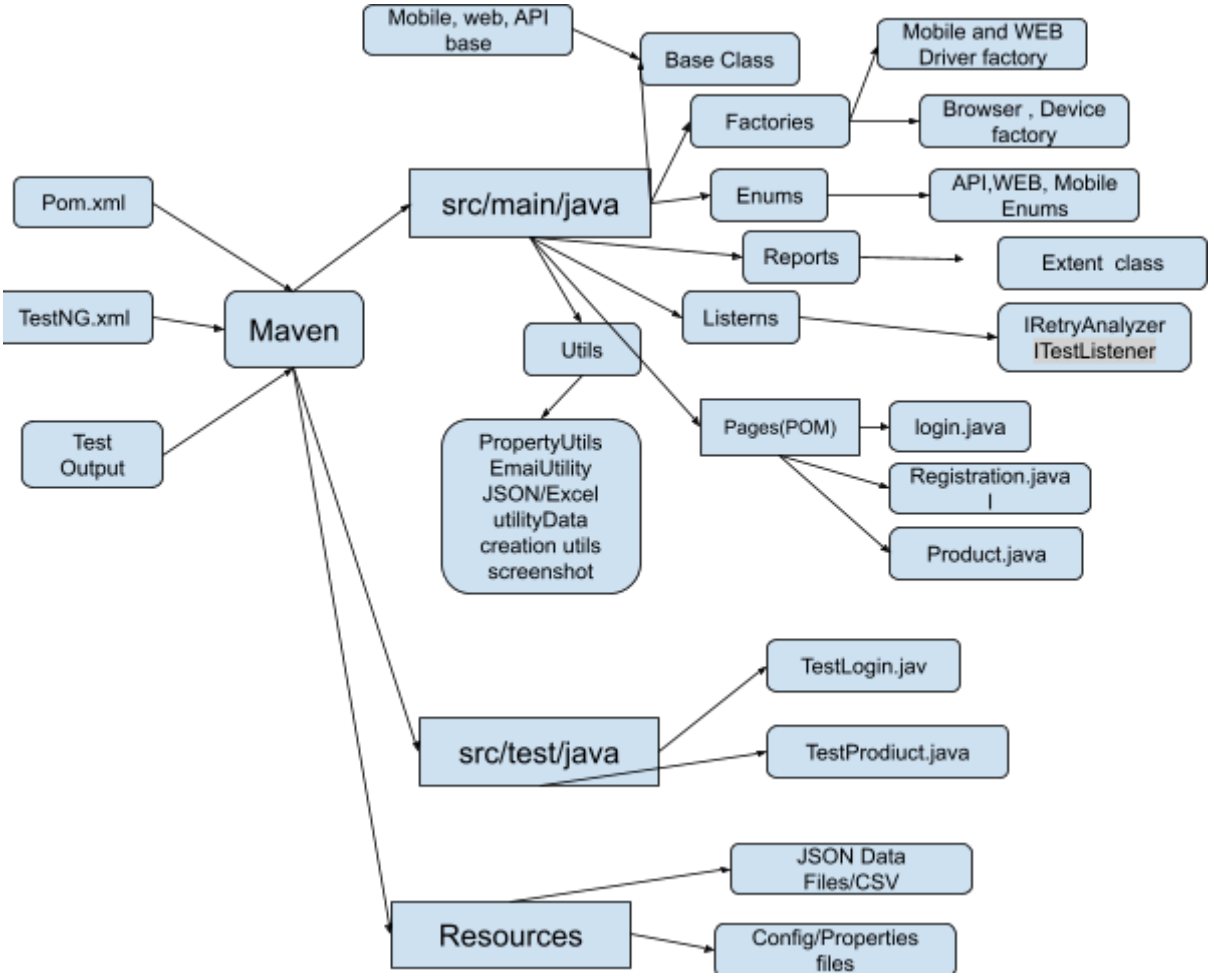
E. Test Environment Management:

- **Test Data Management:** Use databases or data files (JSOn/CSV to manage test data. Ensure data consistency and reusability across different test cases.
- **Environment Configuration:** Use environment variables and configuration files to manage different test environments (development, staging, production).

F. Regular Review and Updates:

- **Code Reviews:** Conduct regular code reviews to ensure code quality and adherence to standards.
- **Retrospectives:** Regularly review the automation strategy and framework for improvements and optimizations.

Framework Design



A web application login functionality using Selenium and Java

```
public class EcommerceLoginTest {

        private WebDriver driver;

        @BeforeMethod
        public void setUp() {
           // Set path to the ChromeDriver executable (we can also use Webdriver Manager)
           System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
           driver = new ChromeDriver();
           driver.manage().window().maximize();
           driver.get("https://example.com/login");

        }
        @Test
        public void testLogin() {
          WebElement username_Text = driver.findElement(By.id("username"));
           WebElement password_Text = driver.findElement(By.id("password"));
           WebElement login_Button = driver.findElement(By.id("loginButton"));
           username_Text .sendKeys("testuser");
           password_Text .sendKeys("password123");
           login_Button.click();
           // Check if login was successful by Asseretions
           WebElement welcomeMessage = driver.findElement(By.id("welcomeMessage"));
           Assert.assertTrue(welcomeMessage.isDisplayed(), "Login failed or welcome message
not found.");
        }

        @AfterMethod
        public void tearDown() {
           driver.quit();
        }
    }
```

Note: This is standalone code, if we implement framework it would be different and will have higher readability. After that our test will look like this

```
  @Test
        public void testLogin()
{
          LoginPage loginPage=New LoginPage(driver)
        loginPage.enterUsername("testuser");
        loginPage.enterPassword("password123");
        loginPage.clickLoginButton();
        homePage = new HomePage(driver);
```

```
        Assert.assertTrue(homePage.isWelcomeMessageDisplayed(), "Login failed or welcome
message not found.");

}
```

A mobile application search functionality using Appium.

```
public class MobileSearchTest {

        Private AndroidDriver driver;
        @BeforeMethod
        public void setUp() throws Exception {
                UiAutomator2Options options = new UiAutomator2Options();
                options.setPlatformName("Android"); // optional
                options.setAutomationName(AutomationName.ANDROID_UIAUTOMATOR2);// optional
                options.setAppActivity(".MainActivity");
                options.setAppPackage("com.shaipwork.android");
                options.setDeviceName(device);
                options.setApp(System.getProperty("user.dir") + "\\resources\\apkbuilds\\shaipwork.apk");
                driver = new AndroidDriver(new URL("http://127.0.0.1:4723"), options);
        }
        @Test
          public void testSearch() {
            // Locate the search box and perform a search
            driver.findElement(By.id("com.example.app:id/searchBox")).sendKeys("testQuesry");
             driver.findElement(By.Id("com.example.app:id/searchButton")).click();

            // Verify the search results
            MobileElement result = driver.findElement(By.Id("com.example.app:id/searchResult"));
            Assert.assertTrue(result.isDisplayed(), "Search result not found.");
          }
        @AfterMethod
        public void tearDown() {
                driver.quit();
        }
}
```

Note: This is standalone code (With framework it will look different)

An API endpoint for creating a new user using RestAssured.

```
public class ApiUserCreationTest {
        @Test
        public void testCreateUser() {

      RestAssured.baseURI = "https://amazon.in";

          // Define the API endpoint and payload
          String endpoint = "/userRegistration";
          String requestBody = "{ \"firstName\": \"newuser\", \"LastName\": \"Test\" }";
          // Send POST request
          Response response = RestAssured.given()
                .header("Content-Type", "application/json")
                .log()
                .all()
                .body(requestBody)
                .when()
                .post(endpoint);
          // Verify the response
          Assert.assertEquals(response.getStatusCode(), 201, "Expected HTTP status code 201 Created.");
          Assert.assertTrue(response.getBody().asString().contains("userId"), "Response body does not contain userId.");
        }
    }
```

Note: This is standalone code (With framework it will look different)

Coding Standards and Best Practices

1. **Separation of Logic from Test:**
    - **Page Objects:** Encapsulate page-specific actions and locators within page object classes. This separates test logic from page-specific code.
    - **Test Methods:** Write test methods that focus on the scenario being tested, leveraging page objects for interactions.
2. **Readability and Maintainability:**
    - **Clear Naming:** Use descriptive names for classes, methods, and variables.
    - **Modular Design:** Organize test code into methods and classes to promote reusability and maintainability.
    - **External Configuration:** Use external configuration files or environment variables for settings like driver paths and URLs.

3. **Error Handling:**
    - **Assertions:** Use meaningful assertions to verify test outcomes and ensure that tests fail with clear error messages. (Soft and Hard assertion)
    - **Exception Handling:** Implement appropriate exception handling to manage unexpected issues. (Custom exception classes and try catch)
4. **Reusability:**
    - **Reusable Methods:** Create reusable methods for common actions (e.g., entering text, clicking buttons).
    - **Utility Classes:** Use utility classes for common functionality, such as WebDriver management.
5. **Scalability:**
    - **Data-Driven Testing:** Consider using data-driven approaches to test multiple scenarios with different data sets.
    - **Parallel Execution:** Configure tests to run in parallel if supported, to reduce execution time. (Implement Threadlocal and singleton patterns)

Test Execution and Reporting (already covered as a part of framework strategy)

Performance Testing with Jmeter

Conducting performance testing for an e-commerce platform using JMeter involves designing and executing tests to assess the platform's stability, and scalability under various load conditions.

1. **Test Planning and Setup**

- Identify critical user scenarios and business transactions (e.g., browsing products, adding items to the cart, checkout process, Product listing)
- Determine what aspects of the e-commerce platform you want to test (e.g., load times, response times).

2. **Design Performance Test plan**

- Design test plans that simulate user interactions with the e-commerce platform.
- Use CSV Data Set Config or User Defined Variables to parameterize inputs (e.g., user credentials, search terms).

3. **Configure Load Generation**

- Configure thread groups to simulate multiple users. Set the number of threads (users), ramp-up period (time to start all threads), and loop count (number of times to repeat the test).
- Use controllers (e.g., Loop Controller, If Controller) to define the logic for executing different parts of the test.
-

4. **Add Listeners**

- Add listeners (e.g., View Results Tree, Summary Report, Aggregate Report) to gather and visualize test results.

4. **Configure Load and Monitor**

- Set up different load levels (e.g., normal load, peak load) to test how the platform handles varying user loads.
- Use JMeter plugins or external monitoring tools to observe server resource usage (CPU, memory). (We might need DevOps team help to monitor server load and health while load testing going on)
- Parallel we can check single user experience of mobile application and web application behavior manually when there is a load on server/

key performance metrics we can observe are

1. Response time of APIs
2. Throughput
3.Concurrency Test with number of user hitting same API at same time
4. The usage of server resources (CPU, memory, disk, network).
5. Latency

E-commerce Domain Knowledge

Edge cases we should consider during E-commerce testing (Broder level)

1. Simulate a high volume of concurrent users and transactions to test performance under peak load.

2. Test the platform's behavior under slow or intermittent network conditions 2G/3G/4G/G

3.Test how the platform handles large volumes of data (e.g., thousands of products, large user base

4.Test the usability of the platform on various devices and screen sizes (responsive design).

5.Test the behavior when a user logs in from multiple devices and Synchronisation testing

6.Data Accuracy: Verify that all displayed data (e.g., prices, product descriptions) is accurate and up-to-date.

7.Test with invalid payment details and verify error handling.

8.Verify the application of discounts and coupon codes under various conditions.

9. Less no of product inventory and high traffic scenarios

10.Browser and Mobile device compatibility for UI and usability

11. Test scenarios where partial payment is made and ensure proper handling.

12. Test how the platform handles network failures during critical operations (e.g., checkout).

Explain how your e-commerce domain knowledge would enhance the quality and effectiveness of the automated tests.

So far i worked for 2 different clients E-commerce website. Performed API and end to end website functional testing. Made with Shopify.

https://my.trapo.asia/

https://jennifermillerjewelry.com/

Attaching link of Github Projects and few live reports.

https://github.com/bhavin21026?tab=repositories

Automation Live Reorts:

AutomationReports