Test plan

StyleMate Website e-commerce platform

Introduction

The document aims to give an overview of the testing approach, strategies, and scope for the StyleMate. This document includes details like the scope of the project, objectives, test schedule and resource allocations, test deliverables, and reports.

Objectives

Primary Objective

- Validate the functionality, usability and performance of the StyleMate.

Objectives

- Ensure that the app meets the specified requirements and user expectations.
- Identify and mitigate potential risks to ensure a smooth user experience.

Scope

This test plan only covers functional testing, performance testing and compatibility testing of the StyleMate across different browsers.

Testing Features

- User registration
- Login
- Password

Testing Approach

Testing Types

- **Functional Testing**: Functional testing is a type of software testing that verifies the functionality of an application against its specified requirements. It focuses on testing each feature or function of the

software to ensure it behaves as expected, including user interactions, data processing, and integration with other systems, without considering internal code structure.

- **Performance Testing**: Performance testing is a type of software testing that evaluates how a system performs under various conditions, such as load, stress, and volume. It aims to identify performance bottlenecks, ensure stability, and measure response times, throughput, and resource usage to ensure the system meets required speed and scalability standards.
- **Usability Testing**: Usability testing is a technique used to evaluate how easy and effective a product is for users. It involves observing real users as they interact with the product to identify any issues or areas for improvement, ensuring the product meets user needs and expectations.

Testing Methodology

- **Black-box testing**: Black-box testing is a software testing method that examines the functionality of an application without knowing its internal code or structure. Testers focus on input and output, verifying that the software behaves as expected, ensuring it meets user requirements and specifications. It's useful for identifying issues like usability and performance defects.
- **White box testing**: White box testing is a software testing method where the internal structure, design, and implementation of the application are known to the tester. The tester uses this knowledge to design test cases that thoroughly examine the code, logic, and data flow, ensuring all paths and conditions are tested for correctness.
- **Regression testing**: Regression testing is a software testing practice that ensures that recent code changes haven't adversely affected existing features. It involves re-running previously passed test cases to confirm that new code modifications don’t introduce new bugs or regressions in the software's functionality. This helps maintain software quality.
- **User Acceptance Testing**: User Acceptance Testing (UAT) is the final phase of software testing where actual users test the software in a real-world

environment to ensure it meets their requirements and works as expected. UAT focuses on validating the functionality, usability, and overall performance of the software before it is released.

Operating System

- Windows 11 or above.
- Browser: Chrome, Microsoft Edge, Firefox
-

Automation Requirements

Requirements:

1. Programming Language: Java.
2. Testing Tools: Cucumber, Selenium.
3. IDE: Eclipse.

Folder Structure:

Project Name
|-> Features(Folder of mavenProject) -> CreateAccount.feature
|-> pageObjects(Package of src/test/java) -> Register.java
|-> stepDefinitions(Package of src/test/java) -> Steps.java
|-> testRunner(Package of src/test/java) -> RegisterRun.java
|-> Utilities(Package of src/test/java)
|-> Drivers(Folder of mavenProject)
|-> target(for Reports already present)
|-> pom.xml

Guide:

Create the maven Project:
File -> New -> Others -> Maven(Folder) -> Maven Project -> Next -> Next -> In the New Maven Project -> Select Catalog as Internal -> provide same name of the Project in the Group id and Artifact id -> Finish

Delete the Packages present in the src/main/java and src/test/java Which is created by default

Dependencies need to add:

a. Cucumber-core
b. Cucumber-html
c. cobertura code coverage
d. Cucumber-java
e. Cucumber-junit
f. Cucumber-jvm-deps
g. Cucumber-reporting
h. Hamcrest-core
i. Gherkin
j. Junit
k. Selenium-java
l. com.sun tools

Create Folder Structure:

Project Name
- |-> Features(Folder of mavenProject)
- |-> pageObjects(Package of src/test/java)
- |-> stepDefinitions(Package of src/test/java)
- |-> testRunner(Package of src/test/java)
- |-> Utilities(Package of src/test/java)
- |-> Drivers(Folder of mavenProject)
- |-> target(for Reports already present)
- |-> pom.xml

In the Drivers folder -> add the necessary drivers.

Right Click -> Features Folder -> New -> File -> Name: Login.feature -> Finish
Write in Login.feature.

In the pageObjects package -> Create class -> Name: LoginPage -> Finish
Write in LoginPage.java

Right click -> stepDefinitions -> New -> Class -> Name -> Steps -> Finish

Right Click -> MavenProject -> Maven -> Update Project -> OK
Login.feature -> Run as -> Cucumber Feature

Right Click -> mavenProject -> Run as -> Run Configurations
Name Correct -> Project Name
Enter the correct Feature Path if needed
-> Apply -> Run

In the Login.feature -> Right click -> Run as -> Cucumber Feature
Copy the methods shown at Console -> Paste in the Steps.java

Then remove the following from the Steps.java:
// Write code here that turns the phrase above into concrete actions
throw new io.cucumber.java.PendingException();

Write -> in -> Steps.java

Inside the testRunner -> Create Class -> TestRun.java
Write -> in -> TestRun.java

refresh the project -> In the target Folder -> Open the htmlreport.html
-> Open with System Editor

In the Login.feature -> add the Scenario Outline and examples
in the Testrun.java -> make dryRun = true -> run the TestRun.java -> Junit Test
watch the output

In the Login.feature -> make dryRun = false -> run the TestRun.java -> Junit Test
watch the output

Dependencies of [pom.xml]:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>mavenProjectV001_nopCommerce</groupId>
  <artifactId>mavenProjectV001_nopCommerce</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>mavenProjectV001_nopCommerce</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
```

```
   <!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.13.2</version>
   <scope>test</scope>
</dependency>

   <!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-core -->
<dependency>
   <groupId>io.cucumber</groupId>
   <artifactId>cucumber-core</artifactId>
   <version>7.20.1</version>
</dependency>

<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-html -->
<dependency>
   <groupId>io.cucumber</groupId>
   <artifactId>cucumber-html</artifactId>
   <version>0.2.7</version>
</dependency>

<!-- https://mvnrepository.com/artifact/net.sourceforge.cobertura/cobertura -->
<dependency>
   <groupId>net.sourceforge.cobertura</groupId>
   <artifactId>cobertura</artifactId>
   <version>2.1.1</version>
   <scope>test</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-java -->
<dependency>
   <groupId>io.cucumber</groupId>
   <artifactId>cucumber-java</artifactId>
   <version>7.20.1</version>
</dependency>

<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-junit -->
<dependency>
   <groupId>io.cucumber</groupId>
   <artifactId>cucumber-junit</artifactId>
   <version>7.20.1</version>
   <scope>test</scope>
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-jvm-deps -->
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-jvm-deps</artifactId>
    <version>1.0.6</version>
    <scope>provided</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/net.masterthought/cucumber-reporting -->
<dependency>
    <groupId>net.masterthought</groupId>
    <artifactId>cucumber-reporting</artifactId>
    <version>5.8.4</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.hamcrest/hamcrest-core -->
<dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest-core</artifactId>
    <version>3.0</version>
    <scope>test</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/io.cucumber/gherkin -->
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>gherkin</artifactId>
    <version>30.0.4</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.27.0</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.sun/tools -->
<dependency>
    <groupId>com.sun</groupId>
    <artifactId>tools</artifactId>
    <version>1.5.0</version>
    <scope>system</scope>
```

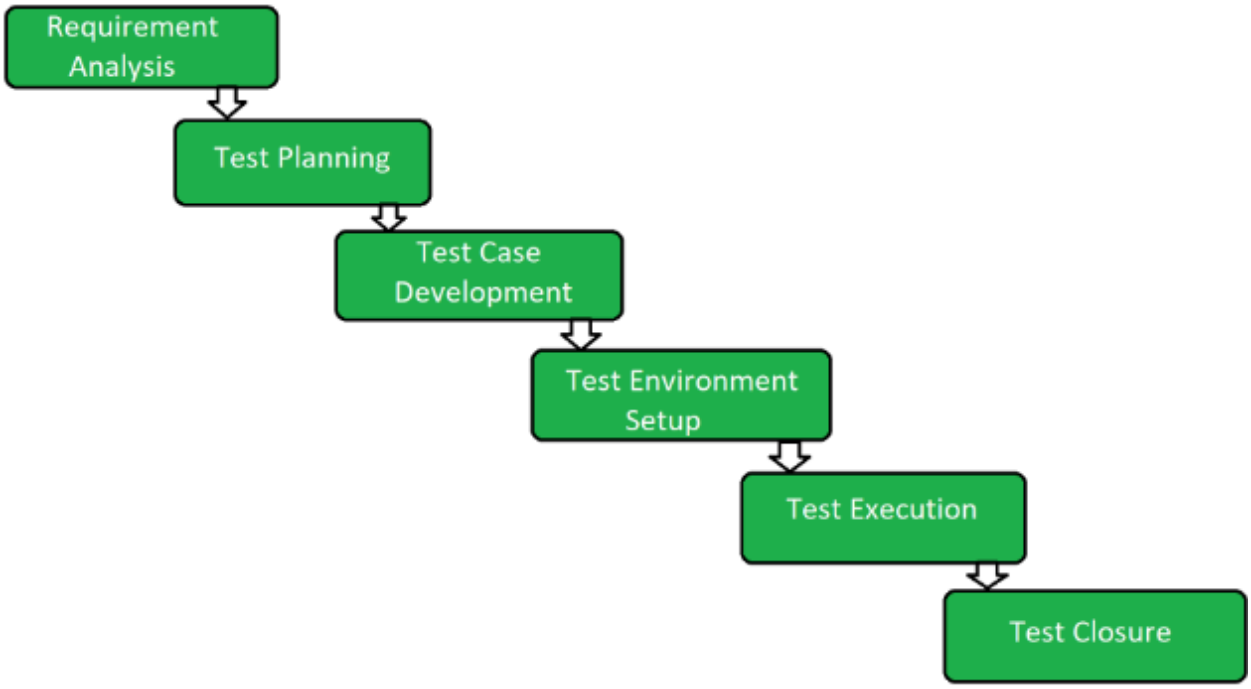
```
    <systemPath>C:\Program Files\Java\jdk-23\lib\tools.jar</systemPath>
</dependency>

<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-java -->
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>7.20.1</version>
</dependency>
  </dependencies>

</project>
```

Entry and Exit Criteria

The below are the entry and exit criteria for every phase of the Software Testing Life Cycle:



1. Requirement Analysis

Entry Criteria

Once the testing team receives the Requirements documents and user documents of the project. Includes what functions need to be tested and how the software should behave.

Exit Criteria

The testing team thoroughly explore and understand each requirement listed in the documents. Any doubts and uncertainty regarding the requirement should be addressed and clarified to ensure that the testing team has a clear understanding of what need to be tested and how it should behave.

2. Test Planning

Entry Criteria

The planning phase begins once the project plan is approved and detailed requirements are available.

Exit Criteria

The phase concludes with the approval of the test plan document by the Client.

3. Test Designing

Entry Criteria

The design phase starts after the test plan is approved. It requires detailed functional and non-functional requirements and access to design tools and environments.

Exit Criteria

The phase ends when all Test Cases and Test Scenarios are reviewed and approved.

4. Test Execution Phase

Entry Criteria

Execution starts when Test Cases and Test Scenarios are ready, and the test environment is set up. Test data must be prepared and validated, and the test team must be trained.

Exit Criteria

Test Case Reports, Defect Reports are ready.

5. Test Closure

Entry Criteria

Test Case Reports, Defect Reports are ready.

Exit Criteria

It concludes with the preparation and review of the test summary reports.

Tools

The following are the list of Tools we will be using in this Project:

- **Selenium** is a widely-used **open-source automation testing tool** for web applications. It allows software testers to automate browser actions, such as navigating to web pages, clicking buttons, filling out forms, and verifying outcomes. Selenium is primarily used for functional and regression testing of web applications across different browsers and operating systems.
- **Cucumber** is a **Behavior-Driven Development (BDD) tool** used by software testers to write automated test cases in plain language that is easy for both technical and non-technical stakeholders to understand. It allows you to describe the application's behavior using a format called **Gherkin**, which uses natural language syntax.

Risks and Mitigation Plans

The following are the list of risks possible and the ways to mitigate them

Risk	Mitigation plans
Lack of Expert Automation Testers	Backup Resource Planning
No detailed Requirements are available	Subject matter experts available for a deep understanding of functionalities

Approvals

Masai will send different types of documents for Client Approval like below:

1. Test Plan
2. Test Scenarios
3. Reports

Testing will only continue to the next steps once these approvals are done.