Spree Ecommerce Test Strategy

Table of Contents

1. Introduction

1.1 Overview

Spree Ecommerce is an open source ecommerce framework. Spree has been used by numerous companies from different domains (Fashion, beauty , Health etc..) to build products like marketplace , ecommerce sites quickly by leveraging the underlying solution , thus enabling them to release their products to market faster.

Spree helps user to do shopping. User can search for products, add to cart and buy the products. User can create an account by registering in the application.

Spree Ecommerce is facing issues with increased development time and bugs while building new features. It has plans to enter new markets/regions and build omni-channel capability.

2. Objectives of Test Strategy

The objectives of the test strategy are to do the following:

- Provide a framework for testing any product developed using the Spree Ecommerce framework so that the effort stays focused and on schedule.
- Document the different types of testing that needs to be done to deliver a fully functional high quality product.
- Provide recommendations on usage of appropriate tools, test approach, automation framework, defect management process.

3. Scope and Limitations

3.1 Scope

- Test the enhancements to Spree Ecommerce to enter new markets/regions.
- Test the Omnichannel capability of Spree Ecommerce.
- Includes manual and automation testing of backlog sprints till current sprint.

3.2 Limitations

Automation of third party tools like payment gateways cannot be performed.

4. Test Approach

4.1 Summary of Test Approach

Application development should be done using the **Agile** Methodology with **Test Driven Development**. Test Driven Development helps in providing basic safety net for the product. **Test Pyramid** model is followed for testing. We should focus on producing many more low-level unit tests than high-level UI-based automated and manual tests.

Unit tests increase the safety net of the application. Apart from working on creating unit tests for new code, developers should work on creating extensive unit tests for existing code over time. **Regression test** suite containing a list of high level functionalities across all the modules should be executed to ensure a new feature does not break existing features. While verifying any bug fix, impacted areas should be identified and retested. Automated tests (Unit, API, UI) are run through CI/CD.

Testing for Enhancements : Spree Ecommerce plans to support new market/regions. Localizability consists of **internationalization** (i18n) and **localization** (l10n) testing. Once the content is translated, it is important to do content verification to ensure material is properly translated as per the language and culture. Need to test that the UI, default language, currency, date, time format, validation messages and documentation are designed as per the targeted country or region.
Integration with new payment and shipping vendors needs to be tested.
Database testing needs to be performed since support for localizability may include some changes to the database schema. Existing data migration to the new database schema needs to be tested.
Scalability testing needs to be performed. **Cross Browser and Cross device testing** needs to be carried out.

4.2 Scope

1. Test the user stories at hand

2. Regress the previous stories and features
3. Ensure environment coverage

4.3 Out of Scope

Third party payment tools used in application cannot be automated.

4.4 Manual test process

- Manual test cases will be drafted for each user story within the sprint
- Testcase will be reviewed before they are executed
- Any defects found during testing will be raised and assigned to developer
- Test Report will be sent to all the stake holders along with defects found during the testing phase at the end of each sprint

4.5 Test Types

4.5.1 Unit Testing

Automated tests on code level which run every night or after new code is added. Unit tests should be written for both functional and UI classes(Javascript). Unit tests should also extensively test methods updating database tables. Unit tests should also test boundary values and edge cases. Unit tests increase the safety net of the application. Test Driven Development(TDD) approach needs to be followed. Code coverage tools can be used to validate the effectiveness. This will be performed by the developer. However QA could be part of unit testing as well. Bugs found are fixed immediately.

4.5.2 API/Service Testing

Perform API testing or service testing to test code earlier in the cycle instead of relying only on “black box” testing. Automated API testing is done using RestAssured and TestNG. All new APIs involved in the project has to be tested individually.

4.5.3 Integration Testing

Testing interactions between different modules of the system. Integration testing will be formed on different modules of the application in each sprint. Integration between APIs is also tested. In case any third party integration is not available early, stubs or mocks should be used to perform integration testing. Whenever the third party integration is available, we need to use that for testing.

In our product, we could mock new payment gateways/shipment gateways if it is not available.

While testing integration with payment gateway, we need to test scenarios like

- Different values for parameters that are passed through the payment gateway

- Test error codes related to payment gateway

- Check the language of the payment gateway and the application

- Check the currency format of the payment

- Verify if transaction can be refunded

While testing integration with shipment gateway, we need to test scenarios like

- Different shipping methods

- Different shipping error messages

- Get quote for shipping

4.5.4 Smoke/Acceptance Testing

Smoke/Acceptance test suite contains only high-level functionality to make sure the application is stable enough for further development or testing.

The automated acceptance tests include Integration Tests, Service Tests , UI tests which aim to prove the software works at a functional level and that it meets user's requirements and specifications.

4.5.5 Database Testing

Database testing should be performed. We need to test scenarios like

- Verify changes to database schema, tables(additional fields to support localization), field constraints

- Checking the default value for fields

- Existing data migration to the new database schema

4.5.6 Functional Testing

Functional testing of the application is performed in each sprint and defects will be raised.

4.5.7 UI Testing

Testing done to verify the UI of different pages in the application.

4.5.8 Regression Testing

Tests to detect side effects from changes to the system. Regression testing will be performed after each sprint and defect will be reported.

4.5.9 Cross Browser/Cross Device Testing

Testing needs to be done across different supported browsers and supported devices. Cross browser testing should cover the main platforms like Linux, Windows, Mac etc… We could test scenarios like performing few actions in a browser and continuing the usage in a mobile device (like Add a product to cart using a browser and placing order for that product in a mobile device).

4.5.10 Usability Testing

Testing of the application to ensure that the intended users of a system can carry out their tasks efficiently, effectively, and satisfactorily.

4.5.11 Exploratory Testing

Exploratory testing will be performed keeping in mind the user actions.

4.5.12 Accessibility Testing

Testing that will be done to ensure that the product/component is compliant with accessibility standards. Testing will complete with the submission of a compliance report.

4.5.13 Localizability Testing

Localizability consists of internationalization (i18n) and localization (l10n) testing. Some of the test scenarios would include :

- Test double-byte languages

- Test Time and date format, Phone number format, Currency, Weights and measures is displayed properly for the target region

- Test validation Message for Input Fields for different regions

- Verify Currency conversions are handled properly

- Test switching regions

- Test Keyboard usage/layout for different locale

4.5.14 Compliance Testing

Since this product involves handling financial data and personal data, testing for compliance with PCI and GDPR need to be performed. Chosen 3rd party payment providers shall adhere to PCI compliance as well.

4.5.15 Security Testing

Security Tests should check for basic security vulnerabilities derived from OWASP. Penetration testing should be done. Security testing is done across all the layers – UI, API, Integration.

4.5.16 Performance Testing

Performance tests should cover tests to test response time performance. Load testing should be done. Scalability analysis should be done. JMeter could be used for performance tests.

4.5.17 User Acceptance Testing

User Acceptance Tests are tests to confirm the built product is what was expected and that it meets user's expectations.

4.6 Test Levels

We follow the **Test Pyramid** model. We should focus on producing many more low-level unit tests than high-level UI-based automated and manual tests, including exploratory and usability testing.

4.7 Automation Test Process

1. Test cases will be identified in each sprint which can be automated and the same will be automated
2. Regression suite covering the critical functionalities of the application from Sprint-1 to Sprint-N will be identified
3. Regression suite identified in step-2 will be automated and test scripts will be added after each sprint
4. Regression suite will be executed in each sprint to check the health of the application
5. Automated scripts will be executed as part of Continuous Integration process

5. Test Environment

TYPES OF TESTING	AUTOMATED/MANUAL	ENVIRONMENT	EXECUTED BY
Unit Tests	Automated	Development	CI
API/Service Tests	Both	Development/QA env	QA
Integration Tests	Automated	Development	CI
Smoke/Acceptance Tests	Automated	All env	CI
Database Tests	Manual	Development/QA env	QA
Functional Tests	Both	QA env	Manual - QA, Automation - CI
UI Tests	Both	QA env	Manual - QA, Automation - CI
Regression Tests	Both	QA/Regression env	Manual - QA, Automation - CI
Cross Browser/Cross Device Tests	Both	QA/Regression env	Manual - QA, Automation - CI
Usability Tests	Manual	UAT Env	QA/BA + Business Use
Exploratory Tests	Manual	QA env	QA
Accessibility Tests	Both	QA env	QA
Localizability Tests	Both	QA env	QA
Compliance Tests	Both	QA env	QA
Security Tests	Both	Dev env, QA env, Pre-prod/UAT env	QA
Performance Tests	Both	Dev env, QA env, Pre-prod/UAT env	Dev, QA
User Acceptance Tests	Manual	UAT Env	Business users

6. Defect Life Cycle

All the Defects/Suggestions will be raised in Jira. Report can be pulled out anytime to see the status of any defect. Defect report will be broadcasted to all the stake holders.

7. Resource

Four QA resources will be available to test the project.

8. Tools/Framework Used

Tools/Framework	Purpose
IntelliJ	Opensource development tool for developing the scripts
Selenium Webdriver(Java)	Functional and UI Automation
Appium	Mobile Automation
Junit	Unit Testing
TestNG	Automation and Reporting
PostMan	Manual API Testing
RestAssured	Automated API Testing
Jmeter	Performance Testing
JIRA	Test and Defect Management
Maven	Build tool to enable CI/CD
Jenkins	CI/CD
GIT	Configuration/Version management

9. Assumptions

Manual test cases will be used as reference while automating the features.

10. Control Procedure

1. Automated features will be delivered sprint wise
2. Every item discussed on call will be documented
3. Deadline of testing might change based on complexity of the feature being tested
4. Test Documents – Test Plan, Test Strategy, Test Case Document, Automated Scripts, reports etc. will be uploaded in GitHub.
5. QA team will be responsible for managing the test documents in GitHub.

11. Deliverables

1. Manual TCD
2. Automated Scripts
3. Test execution report

12. Key Metrics

1. Code Coverage
2. Test Coverage - Manual Test coverage, Automation test coverage
3. Traceability Matrix
4. Defect leakage

5. Defect re-open rate
6. Percentage of non-defects

13. Risks and mitigation plans

Risk	**Mitigation plan**
Environment unavailability	Use containerized deployment (such as docker) to enable quick spin up of new environments
Resource unavailability	Weekly meetings to assess resource capacity vs requirements
Increase in project scope	Effort for the new project scope is estimated and communicated