SMART LIVING SHOP WAVE

TEST PLAN

PREPARED BY

STEFCHE GJORGIEVSKI

DOCUMENT REVISION HISTORY			
DATE	DOCUMENT VERSION	REVISION DESCRIPTION	AUTHOR
09/01/2024	1.0	Initial Version	Stefche.G

TABLE OF CONTENTS

1. INTRODUCTION

This test plan outlines the strategy and approach for validating the Smart Living software product, version 1.0, through automated testing. Smart Living is an innovative concept that combines lighting, furniture, and decorative hardware into a cohesive shopping experience. Designed as both an architectural concept studio and a furniture shop, it aims to provide customers with a modern, user-friendly interface that simplifies the selection and purchasing of home decor items. Our goal is to ensure the software meets all specified requirements and functions as intended before release.

This document details the testing objectives, scope, resources, and deliverables, providing a framework for evaluating performance, reliability, and user experience. Automated testing will efficiently cover a wide range of scenarios, allowing for quicker feedback and thorough validation.

Collaboration with stakeholders from various departments, including product management, development, design, and marketing, will be crucial to ensure alignment with quality standards and business objectives. Engaging these stakeholders throughout the testing process will help address concerns early and foster a shared understanding of project goals. Their insights and feedback will be invaluable in refining the software and enhancing the overall user experience.

By following this test plan, we aim to deliver a robust, user-friendly solution that enhances the Smart Living experience.

1.1 TEST OBJECTIVES

Our automated testing efforts aim to enhance the website's effectiveness, ensuring it reflects the company’s dedication to a modern, user-friendly interface. We anticipate a significant increase in customer satisfaction over the next quarter through rigorous automated testing.

The test plans for the Smart Living website support the following measurable objectives:

1) Ensure Functional Integrity: Validate that all website features, including product displays, navigation, and checkout processes, function properly and meet the specified requirements, aiming for a 95% pass rate on all automated test cases.

2) Enhance User Experience: Assess the usability and accessibility of the website to guarantee a seamless and intuitive experience for all users. The baseline customer satisfaction score is currently 70%. We target an improvement to at least 80% in follow-up surveys.

3) Optimize Performance: Evaluate the website's load times and responsiveness under varying traffic conditions to ensure optimal performance, with a goal of maintaining page load times under 2 seconds for 95% of users. Current average load times will be documented as a baseline for comparison.

4) Verify Compatibility: Test the website across multiple devices, browsers, and operating systems to ensure a consistent experience for all users, targeting 100% coverage of major platforms.

5) Assess Security Measures: Identify potential vulnerabilities and ensure that user data is protected through robust security protocols, aiming for zero critical security vulnerabilities in API tests.

6) Facilitate Search Engine Optimization (SEO): Ensure that the website is optimized for search engines, aiming for a minimum score of 80% on SEO audit tools, using baseline scores from existing audits for comparison.

7) Gather Analytics and Reporting: Implement tracking to collect user interaction data for continuous improvement, with the goal of analyzing user behavior trends quarterly and establishing a baseline for current interaction metrics.

8) Support Continuous Improvement: Establish a framework for ongoing testing and feedback, aiming for bi-weekly reviews of test results to implement improvements based on baseline performance data.

9) Align with Brand Identity: Ensure that the website design reflects Smart Living's brand values and aesthetic, with at least 90% positive

feedback from brand stakeholders, comparing feedback against a baseline survey.

10) Increase Conversion Rates: Identify and fix barriers in the user journey to boost conversions. The current conversion rate is 5%, and we are targeting a 10% increase within three months after testing implementation, using baseline data for comparison.

2. SCOPE

The testing scope for the Smart Living website encompasses the following areas:

2.1 IN-SCOPE AREAS

❖ Functionality Testing

Validate core features and functionalities of the Smart Living software to ensure they meet specified requirements.

Examples:

- Testing product search functionality to ensure users can easily find items by keywords.
- Verifying the accuracy of product details displayed, including price, specifications, and images.
- Ensuring the checkout process works smoothly, from cart to payment confirmation.

❖ Regression Testing

Automated tests are performed for existing features to ensure that any recent code changes don't introduce new defects.

Examples:

- Automated tests are performed on previously validated features like user registration and login to ensure that, they remain unaffected by new code updates.

- Validating that promotional codes and discounts are applied correctly after updates to pricing logic**.**

❖ Performance Testing

Assess the application's performance under various load conditions to ensure it meets performance benchmarks. This will involve:

- Integration with Performance Tools: While Selenium is primarily for functional testing, it will be complemented by performance testing tools like JMeter or Gatling. These tools will simulate multiple users interacting with the application to measure load times, response times, and resource utilization.

- Performance Scenarios: Define specific scenarios to test, such as peak load conditions and user interactions during high traffic. Analyze the results to identify bottlenecks and optimize the application accordingly.

Examples:

- Simulating 1,000 concurrent users browsing the product catalog to measure page load times.

- Testing the response time of the server when users are checking out simultaneously during a promotional event.

❖ User Interface Testing

Verify that the user interface elements display correctly and function as intended across different devices and screen sizes.

Examples:

- Ensure that buttons and images are aligned and visible on both mobile and desktop versions of the site.

- Ensures that dropdown menus and interactive elements respond appropriately on touch devices.

- ❖ API Testing

Test the application programming interfaces (APIs) for functionality, reliability, and security.

Examples:

- Verifying that the API returns correct product information when queried with valid parameters.
- Testing the API's response time and error handling for invalid requests.

- ❖ Integration Testing

Ensure that different modules and services within the software interact seamlessly and function correctly together.

Examples:

- Testing the integration between the payment gateway and the checkout system to confirm transactions are processed accurately.
- Validating that inventory updates, occur in real-time when purchases are made.

- ❖ Smoke Testing

Verify the basic functionality of the application after each new build to ensure critical features are operational.

Examples:

- Ensuring users can successfully log in, search for products, and add items to the cart.

- Checking that the homepage loads without errors and all primary navigation links work.

❖ Sanity Testing

Confirm that key functionalities of the application work correctly after each new build.

Examples:

- Verifying that the user profile works properly after a new deployment.
- Verifying that the logout function works properly after backend changes.

2.2 OUT OF SCOPE AREAS

❖ Usability Testing

Evaluating overall user experience is excluded from automated tests, as this requires human judgment and subjective analysis that automation cannot accurately capture. Human insights are essential for understanding user interactions and preferences, making this type of testing better suited for manual evaluation.

❖ Security Testing

In-depth security assessments are excluded; however, basic API security checks may be included. This exclusion is due to the complexity of comprehensive security evaluations, which typically require specialized tools and expertise.Given the importance of security, we recommend conducting thorough security audits separately.

❖ End-to-End Testing

Full end-to-end testing that involves multiple systems or external dependencies will not be automated, as these scenarios often require complex configurations and manual oversight. Our automated testing will concentrate on specific functionalities to ensure efficiency and comprehensive coverage.

❖ Mobile Testing

Mobile purchasing through devices will not be tested as our current focus is on optimizing the desktop web experience. Additionally, mobile testing would require a separate set of automated tests and resources that are currently not allocated.

❖ Stress Testing

Specific stress testing scenarios are not included in the automated scope because these tests often require unique configurations and conditions that go beyond standard functional testing. For now, we will focus on performance under normal load conditions.

3. TEST STRATEGY

3.1 AUTOMATION FRAMEWORK

We will utilize the Selenium framework in conjunction with C# to automate the testing of the Smart Living website. Selenium provides robust support for web application testing, allowing us to interact with the user interface as a real user would. The choice of C# as our programming language ensures seamless integration with our existing development environment.

3.2 TEST TOOLS

1) Automation Tools

- Web Automation:
  - ➢ Selenium.WebDriver.ChromeDriver
  - ➢ Selenium.WebDriver.MicrosoftDriver

2) Test Framework:
    - ➢ NUnit

3) Continuous Integration

- CI/CD Tool:
    - ➢ Azure DevOps

4) Reporting Tools

- Reporting Frameworks:
    - ➢ Allure Reports
    - ➢ ExtentReports

5) API Testing Tools
    - ➢ Postman
    - ➢ RestAssured

6) Performance Testing Tools
    - ➢ JMeter
    - ➢ Gatling

7) Code Coverage Tools
    - ➢ Coverlet Collector

8) ORM Tools

- Object-Relational Mapping:
    - ➢ Entity Framework

3.3 TEST DEVELOPMENT

To enhance our test development process, we will implement the Page Object Model (POM) design pattern, which will promote maintainability and reusability of test scripts. Additionally, we recognize the importance of effective test data management in ensuring comprehensive test coverage.

- ❖ Dynamic Test Data Management:

We will utilize various external data sources to manage test data dynamically, enabling us to run tests under different scenarios without hard coding values. This approach not only improves the flexibility of our test scripts but also ensures that they remain relevant as the application evolves.

Examples of Data Sources:

CSV Files:

We will use CSV files to provide input data for our tests, allowing easy updates and modifications without altering the test scripts themselves. For instance, a CSV file containing user credentials can be utilized for login tests.

JSON Files:

JSON files will be employed to manage complex data structures. For example, we can store product information in a JSON format to test various e-commerce functionalities, such as adding items to a cart.

Databases:

We will leverage our test database to pull data dynamically during test execution. This can include retrieving user profiles, product listings, or other relevant data needed for comprehensive testing.

Environment Variables:

Utilizing environment variables can help manage sensitive information, such as API keys or passwords, securely. This way, we can keep our test data secure while allowing tests to access necessary credentials during execution.

By implementing dynamic test data management strategies, we will enhance the robustness and adaptability of our automated testing process, ensuring that it remains aligned with changing application requirements. This will ultimately contribute to higher quality and more reliable software delivery.

3.4 CONTINOUS INTEGRATION

To ensure that our automated tests contribute effectively to the development process, we will integrate them into our Continuous Integration/Continuous Deployment (CI/CD) pipeline. This integration is crucial for providing immediate feedback on code changes and maintaining high software quality.

Test Frequency:

We will schedule our automated tests to run after every build, allowing us to catch issues as soon as code changes are introduced. This frequent testing will help maintain a reliable codebase and facilitate quicker identification of defects.

Additionally, we will implement nightly test runs to perform more extensive testing scenarios, including integration and performance tests. This dual approach ensures that we not only validate every change immediately but also conduct thorough evaluations on a regular basis.

This structured testing regimen aligns with agile development principles and helps our team deliver robust and reliable software efficiently.

4. TEST DELIVERABLES

1. Test Plans:

- Description: Detailed documentation outlining the testing strategy, objectives, and approach.
- Timeline: To be completed within the first week of the testing phase.

2. Test Cases:

- Description: A comprehensive suite of automated test scripts designed for various functionalities.
- Timeline: Development to begin in the second week, with completion expected by the end of week three.

3. Test Data:

- Description: External data files (CSV, JSON) used for dynamic test execution.
- Timeline: To be prepared concurrently with test case development, finalized by week three.

4. Test Execution Reports:

- Description: Reports summarizing the results of each test run, including pass/fail status and any issues identified.
- Timeline: Generated after each test execution, with initial reports available by the end of week four.

5. Defect Reports:

- Description: Documentation of any defects found during testing, including severity levels and steps to reproduce.
- Timeline: To be compiled in real-time as defects are identified, with weekly summaries provided.

6. Test Coverage Reports:

- Description: Insights into the extent of code covered by automated tests, using tools like Coverlet.
- Timeline: Generated alongside test execution reports, with a comprehensive report available by the end of week four.

7. Final Test Summary Report:

- Description: A comprehensive report summarizing all testing activities, results, and recommendations for release.
- Timeline: To be completed within one week following the final round of testing

Deliverable	Description	Due Date	Responsible
Test Plan	Detailed documentation outlining the testing strategy.	[Date: Week 1]	[Team Member/Role]
Test Cases	Automated test scripts for various functionalities.	[Date: End of Week 3]	[Team Member/Role]
Test Data	Dynamic test data files (CSV, JSON).	[Date: End of Week 3]	[Team Member/Role]
Test Execution Reports	Summary of results for each test run.	Ongoing; initial by Week 4	[Team Member/Role]
Defect Reports	Documentation of identified defects.	Ongoing	[Team Member/Role]
Test Coverage Reports	Insights into code coverage.	By End of Week 4	[Team Member/Role]
Final Test Summary Report	Comprehensive summary of testing activities.	[Date: End of Week 6]	[Team Member/Role]

5. ENVIRONMENT AND RESOURCES

5.1 INTRODUCTION

The following specifications outline the environment required for conducting our automated tests:

❖ Physical Machines:

These will be used for performance-intensive tests and scenarios requiring direct access to hardware resources. This setup is ideal for testing applications that demand high computational power and resource management, ensuring accurate assessments of performance under realistic conditions.

❖ Virtual Machines (VMs):

VMs provide an efficient way to create isolated testing environments that can be quickly provisioned or decommissioned as needed. This is particularly useful for testing different OS configurations or software versions without the need for dedicated hardware.

❖ Cloud-Based Solutions:

Utilizing cloud services (like AWS, Azure, or Google Cloud) allows for scalable resources that can be adjusted based on testing needs. This option supports remote access and collaboration among distributed teams.

5.2 HARDWARE

For testing purposes, the following hardware requirements will be utilized:

1. Processor: Intel Core i5 or equivalent.
2. RAM: 16 GB or higher.
3. Storage: Minimum 100GB of available disk space.
4. Display: Screen resolution of 1280x720 or higher.

5. Graphics: Integrated graphics card or dedicated GPU with DirectX 11 support.
6. Network: High-speed network connectivity.
7. Input: Keyboard and mouse.
8. Processor: 64-bit Intel 4-core 2.0 GHz or higher processor.
9. Configured PC Workstations.

5.3 SOFTWARE

For testing purposes, the following software requirements and tools will be utilized:

Server:

1. Operating system – Windows 11
2. Web server –
   - Apache HTTP Server – version 2.4.43
   - Tomcat – version 10.1.30
   - Nginx – version 1.26.0
3. Database server - Oracle or MS-SQL Server

Client:

1. Windows operating system – Windows 11.
2. Internet browser – Google Chrome (current version) & Microsoft Edge (current version).

6. TESTING SCHEDULE

6.1 ANALYZE THE REQUIRMENTS

The QA team must thoroughly understand and analyze the project requirements, focusing on both functional and non-functional aspects. Addressing bugs early in development is significantly more cost-effective than fixing them during testing or production stages.

Understanding of Requirements:

1) Requirement specifications will be provided by the client.
2) The QA team will collaborate with the project lead and developers to ensure comprehensive understanding.

3) Queries raised during analysis will be communicated to the client for clarification.

4) Responses to queries will be documented and shared by the client.

6.2 PREPARING TEST CASES

QA/Test Engineers will prepare automated test cases based on requirement specifications, ensuring coverage of all scenarios.

Each automated test case will typically include:

1) Test Case Name: A descriptive title indicating its purpose.

   Input Data: The necessary data required for execution.

2) Expected Result: The anticipated outcome upon execution.

The following table lists the identified test scenarios prioritized by the Smart Living Project Manager:

Test Scenarios	Priority/Risk
• Home Page	High
• Navigation	Medium
• Product Category List	Medium
• Model List	Medium
• Product Detail	High
• Content List	Low
• Content Display	Medium
• Contact Us	Medium
• Login	High
• Logout	Low
• Account Information	Medium
• Registered Products	Medium
• Search	High

• Site Search	Medium
• Reporting	Medium

6.3 PREPARING TEST MATRIX

The QA team will develop a test matrix that documents the relationship between system/project requirements and their corresponding automated test cases throughout the development lifecycle. This ensures that all approved requirements are designed into the solution, verified, and implemented.

6.4 REVIEWING TEST CASE AND MATRIX

- A review of the automated test cases and matrix will be conducted by a senior QA team member.

- For complex requirements, the lead will assist in the review process. Feedback from the reviewer will be provided to the author of each test case.
- Any suggested improvements will be reworked by the author and submitted for approval.

- The revised test cases will undergo another round of review and approval.

6.5 CREATING TEST DATA

Test data will be generated by the respective QA members based on the identified scenarios and test cases. Emphasis will be placed on structuring test data for reusability across multiple test cases.

6.6 EXECUTING TEST CASE

- Automated test cases will be executed using the Selenium framework on the client’s development/test site.

- Results, including actual outcomes and pass/fail status, will be documented in the test case records.

6.7 DEFECT LOGGING AND REPORTING

QA will log defects found during automated test execution using the Bugzilla bug tracking tool. Each defect will be assigned a unique Bug ID, which will be referenced in the test case documentation. The QA team will promptly inform the relevant developer about any logged defects.

6.8 RETESTING AND REGRESSION TESTING

Once bugs are resolved, the QA team will rerun automated tests to confirm the fixes. Regression testing will also be conducted to ensure that new changes do not adversely affect existing functionality.

6.9 DEPLOYMENT AND DELIVERY

- After all reported defects are resolved and no new issues are identified, the report will be deployed to the client’s test site by the developer.
- QA will conduct a final round of testing on the client’s test site, if required.
- The final report, along with sample output, will be emailed to the project lead and relevant stakeholders.
- The QA team will submit a completed delivery slip to the developer.
- Upon receipt of the signed delivery slip from both QA and the developer, the lead will send a delivery confirmation email to the client.

6.10 AUTOMATION TESTING MILESTONES

Milestone 1: Setup and Initial Test Development (1 Month)

- Establish the automation framework (Selenium with C#).
- Create the initial set of automated test cases for critical functionalities.
- Complete initial environment setup.

Milestone 2: Complete Core Test Suite Automation (2-3 Months)

- Automate additional test cases covering key user journeys and features.
- Develop test data management strategies.
- Conduct initial runs and gather metrics.

Milestone 3: Full Test Coverage and Optimization (4-5 Months)

- Achieve comprehensive test coverage, including edge cases and negative scenarios.
- Optimize test scripts for performance and maintainability.
- Begin integrating tests into the CI/CD pipeline.

Milestone 4: Final Review and User Acceptance Testing (5-6 Months)

Conduct thorough regression testing.

- Gather feedback from stakeholders and make necessary adjustments.
- Prepare for deployment and final delivery.

7. ENTRY CRITERIA

1) Application Stability: The application should be stable and free of critical bugs.

2) Test Environment Setup: The testing environment must be fully configured and operational.

3) Test Data Availability: Necessary test data must be prepared and accessible.

4) Test Cases Defined: All relevant test cases should be documented and approved.

5) Test Scripts Development: Automated test scripts should be developed and ready for execution.

6) Documentation: All necessary documentation, including requirements and specifications, must be complete.

7) Version Control: The correct version of the application must be under version control and deployed in the test environment.

8) Review and Approval: All entry criteria must be reviewed and approved by relevant stakeholders.

9) Team Readiness: The testing team should be prepared and trained for the testing phase.

8. EXIT CRITERIA

1) All critical and high-severity defects have been resolved: No outstanding high-priority issues that affect functionality.

2) Test Coverage: At least 95% of the automated test cases must pass, covering key functionalities.

3) Stakeholder Approval: Obtain sign-off from key stakeholders after review of test results and reports.

4) Performance Metrics: All performance benchmarks (load times below 2 seconds) must be met.

5) User Acceptance Testing: Successful completion of user acceptance testing with positive feedback from stakeholders.

9. RESUMPTIONS CRITERIA

1) The development environment is stable and reflects production settings.

2) All necessary resources (hardware, software, personnel) will be available as planned.

3) Test data will be accessible and representative of real-world scenarios.

4) Stakeholders will provide timely feedback and approvals as needed.

5) No major changes to the project scope will occur during the testing phase.

10. RISK MANAGEMENT

Potential risks associated with the testing process:

1) Resource Availability: Risk of key team members being unavailable.

- ➢ Mitigation: Cross-train team members to ensure coverage.

2) Defect Volume: High number of defects found during testing may delay the timeline.

- ➢ Mitigation: Prioritize defects and implement a triage process for efficient handling.

3) Tool Limitations: Risks related to the limitations of automation tools.

- ➢ Mitigation: Regularly evaluate tools and be prepared to adapt strategies if limitations are encountered.

4) Scope Creep: Risk of requirements changing mid-project.

- ➢ Mitigation: Establish a change control process to assess impacts and obtain necessary approvals for any changes.

11. COMMUNICATION PLAN

1) Regular Meetings: To be schedule daily stand-ups or weekly check-ins to discuss progress, issues, and next steps.

2) Reporting: Test results and defect reports will be shared with stakeholders on base of weekly summaries and immediate alerts for critical defects.

3) Stakeholder Engagement: Identify key stakeholders and specify their roles in the testing process.

12. TRAINING AND ONBOARDING PLAN

1) Training Requirements

- New member will have any necessary training sessions for tools(Selenium, Nunit) or processes.

2) Documentation

- Links to resources or documentation that can assist the team in understanding the tools and frameworks being used.

13. CONTINGENCY PLANS

❖ Fallback Strategies if automation test fails:

- Manual Verification: Critical test cases will be manually verified by the testing team to ensure that the application is functioning as expected, even if automated tests fail.

- Logs and Error Analysis: Detailed logs and error messages will be reviewed to identify the root cause of the failure. This analysis will guide further investigation and troubleshooting.

- Test Retry Mechanism: Implement a retry mechanism for transient failures (e.g., network issues or temporary unavailability) to reduce false negatives in the test results.

- Issue Tracking: Any failures will be logged in the issue tracking system for further analysis and prioritization, ensuring that the development team addresses the underlying problems.

- Test Environment Verification: Ensure that the test environment is stable and correctly configured, as environmental issues can contribute to test failures.

❖ Resource Reallocation if certain areas of testing fall behind schedule:

- Prioritization of Critical Tests: Main focus will be on high-priority test cases to ensure critical functionalities are verified first.
- Cross-Training Team Members: Team members will be encouraged to share expertise and take on tasks in different areas to maximize efficiency.
- Flexible Scheduling: Team schedules will be adjusted to allocate more time to delayed testing areas, ensuring that project milestones are met.
- Temporary External Support: External testers will be consider to assist in catching up on testing tasks.

11. APPENDIX

❖ Glossary of terms

ROLE	RESPONSIBILITIES
PM	1. Acts as a primary contact for development and QA team. 2. Responsible for Project schedule and the overall success of the project.
QA & TESTER	1. Understands requirements and acceptance criteria. 2. Writes and executes test cases. 3. Prepares the Requirement Traceability Matrix (RTM). 4. Reviews test cases and the RTM for completeness. 5. Reports and tracks defects in the defect management system.

	6. Conducts retesting and regression testing to ensure fixes are effective. 7. Participates in bug review meetings to discuss defect status and resolution strategies.
DEVELOPER	1. Write and implement quality code. 2. Code review debugging. 3. Perform test and debug errors.
SERVER ADMINISTRATOR	Responsible for maintaining the test environment, ensuring its stability and availability for testing activities.

- References to relevant documents (e.g., requirement specifications).
- Links to training materials for tools and processes.

ACRONYMS

- PM: PROJECT MANAGER
- QA: QUALITY ASSURANCE
- RTM: REQUIRMENT TRACEABILLITY MATRIX