Group 11

The ToolShop
<Iteration/ Master> Test Plan

Version <1.0>

Revision History

Date	Version	Description	Author
<08/12/2024>	<1.0>	<>	<>

Table of Contents

Test Plan

1. Introduction

1.1 Purpose

The purpose of the Iteration Test Plan is to gather all of the information necessary to plan and control the test effort for a given iteration. It describes the approach to testing the software, and is the top-level plan generated and used by managers to direct the test effort.

This *Test Plan* for the ToolShop supports the following objectives:

- Ensure the Application Under Test conforms to functional and nonfunctional requirements
- Ensure the AUT meets the quality specifications defined by the client
- Bugs/issues are identified and fixed before go live
- Regression tests will be done with automated tests.

1.2 Scope

- In Scope:
  - Functional Testing
  - GUI Testing
  - Performance Testing
  - API Testing
  - Usability Testing
- Out Scope:
  - Unit testing
  - System Testing
  - Integration Testing
  - SIT Testing
  - Supportability Testing
  - Reliability Testing

1.3 Intended Audience

- Project Managers and Product Owners: These individuals will use the Test Plan to understand the scope, objectives, and progress of testing efforts, as well as to ensure the application is aligned with business goals and user needs. They will also rely on the test results to make informed decisions regarding release timelines and prioritization of any discovered issues.
- Developers: Developers will refer to this Test Plan to better understand the functional and non-functional requirements of the system under test, as well as the test strategies in place. It will also help them understand potential areas of risk, enabling them to address issues proactively during development and troubleshooting.
- Quality Assurance (QA) Engineers and Testers: The primary audience for the Test Plan, QA engineers and testers will utilize this document to guide their testing efforts. It provides them with clear objectives, expected results, and the criteria for assessing the application's quality. This audience will be most concerned with ensuring the application meets the specified requirements and performs as expected.
- Stakeholders and End Users: While not directly involved in testing, this group will ultimately rely on the Test Plan's outputs to gauge the product's readiness for deployment. They are interested in ensuring the application is stable, secure, and user-friendly. Their feedback may influence future iterations and test objectives.

1.4 Document Terminology and Acronyms

The following are terms, acronyms, and abbreviations used within this Test Plan that require specific definitions for proper interpretation. For general project-related terms, please refer to the project's Glossary.

Terms and Acronyms

- GUI: Graphical User Interface – The interface through which users interact with the Tool Shop Application,

involving elements like buttons, menus, and forms.

- RUP: Rational Unified Process – A software development process framework that provides guidelines, templates, and tools for application development. This process is used to structure the testing workflow for the project.
- Test Case: A set of conditions or actions used to verify that a feature or function of the application works as expected. Each test case defines specific input, expected output, and execution steps.
- Test Script: A set of automated instructions used to execute a particular test case or scenario within a testing tool.
- Katalon: A test automation tool used for executing automated test scripts, particularly for UI and functional tests.
- Smoke Testing: A preliminary testing phase that checks whether the most critical functions of the application are working as expected after a new build or deployment.
- Regression Testing: Testing the existing functionality of the application to ensure that new changes (e.g., bug fixes, feature additions) do not adversely affect previously working features.
- Integration Testing: Testing the interaction between different modules or components of the application to ensure that they function together as expected.
- Exploratory Testing: A type of testing where testers explore the application, often without predefined test cases, to identify defects, particularly edge cases.
- Stress Testing: Testing the system under heavy load to determine how it behaves under stress, such as high traffic or large volumes of data.
- Test Environment: The hardware, software, network configurations, and other settings necessary to conduct tests on the application.
- Test Artifact: Any work product or document produced as part of the testing process, such as test cases, test scripts, logs, and reports.
- Oracle: A source of expected results that helps in determining the outcome of a test, which can be a set of requirements, a database, or an external validation system.

Acronyms

- TBD: To Be Determined – Used when certain details or tools are yet to be decided upon at the time of writing the document.
- API: Application Programming Interface – A set of functions and protocols that allow the Tool Shop Application to interact with other software systems.
- CI/CD: Continuous Integration / Continuous Deployment – A practice of frequently integrating code changes into a shared repository and automatically deploying the latest version of the application.
- UAT: User Acceptance Testing – A phase in the testing process where the final product is tested in a real-world environment to verify that it meets user requirements and expectations.

Reference to Project Glossary

For a complete list of terms, acronyms, and abbreviations that are used in the project, refer to the project's Glossary document, which provides more general definitions applicable to the entire project scope.

1.5 References

The following documents have been referenced within this Test Plan:

1. **Tool Shop Application Requirements Document**
    - Version: 1.1
    - Author: Tool Shop Development Team
    - Description: Contains the functional and non-functional requirements for the Tool Shop Application, including user stories, business rules, and use cases, source code
    - Source: https://github.com/testsmith-io/practice-software-testing/?tab=readme-ov-file
2. **Tool Shop API Swagger**
    - Version: 5.0
    - Author: Tool Shop Development Team

- Description: Details the customized API specialization for the Tool Shop Application project.
- Source: https://api-with-bugs.practicesoftwaretesting.com/api/documentation

2. Evaluation Mission and Test Motivation

2.1 Background

- The testing effort defined in this Test Plan is driven by the need to ensure the delivery of a robust, user-friendly, and secure solution. The project focuses on developing a web application that addresses needs of searching and buying tool.

2.2 Evaluation Mission

- Identify as many functional and usability issues as possible to ensure the application is reliable and meets user expectations.
- Assess and address quality risks associated with critical functionalities such as password encryption, local settings storage, and score calculation in quizzes.
- Verify that all implemented features meet the specified functional and non-functional requirements.
- Provide actionable feedback to stakeholders regarding perceived risks, product quality, and the overall readiness for deployment.

2.3 Test Motivators

- Quality Risks: Ensuring the application's security measures, like password encryption and local settings, function as intended without introducing vulnerabilities.
- Technical Risks: Verifying integration across modules, particularly the smooth transition between pages and data consistency.
- Project Risks: Identifying any potential delays or blockers that could affect the overall delivery timeline.
- Use Cases and Functional Requirements: Validating scenarios like login/logout with 'Remember Me' functionality, proper image display, and accurate quiz scoring.
- Non-Functional Requirements: Evaluating performance, responsiveness, and user interface consistency across different devices and resolutions.
- Change Requests: Incorporating and testing recent feature enhancements or bug fixes to confirm their implementation aligns with user feedback and requirements.

3. Target Test Items

The listing below identifies those test items_software, hardware, and supporting product elements _that have been identified as targets for testing. This list represents what items will be tested.

- Authentication: Sign in, Sign up, Sign out, Forgot your password (Nam)
- Categories (Kiên)
- Catalog: Sort, Filter, Search, Price range, Pagination (Kiên)
- Cart: Cart content, Add product, Delete product (Thu)
- Contact (Nam)
- Product: Inc/Dec pieces (+/-), Add to cart, Add to favorites, Related products, More information (Hường)
- Checkout: SignIn, Address, Payment (Thư)
- *Account management*
- *Admin dashboard*
- *Admin management*

4. Outline of Planned Tests

4.1 Outline of Test Inclusions

- Functional Testing
- GUI Testing
- Performance Testing
- API Testing
- Usability Testing

4.2 Outline of Other Candidates for Potential Inclusion

- Integration Testing

4.3 Outline of Test Exclusions

Outline of test exclusions:

- Unit testing
- System Testing
- SIT Testing
- Supportability Testing
- Reliability Testing

These test techniques have been explicitly excluded from this plan due to:

- These tests do not help achieve the evaluation mission due to the requirement.
- There are insufficient resources to conduct these tests.

5. Test Approach

5.1 Initial Test-Idea Catalogs and Other Reference Sources (Hường)

Below is a list of existing documents and reference sources that will be used to support the identification and selection of specific test cases:

- **Software Requirements Document:** A document detailing the functional and non-functional requirements of the application, including user stories, business rules, and use cases.
- **API Documentation:** Detailed documentation of the Application Programming Interface (API) used in the project, including information about methods, endpoints, and input/output parameters.
- **Test Ideas Catalog:** A catalog compiling fundamental testing ideas, including functional testing, GUI testing, performance testing, and usability testing.
- **Industry Standards and Best Practices:** Source: Software testing standards (e.g., ISTQB) and recommendations from the software development community.
- **Previous Test Results:** Test reports from previous versions of the application, used to identify potential risks and critical test cases that require repetition.
- **End User Feedback:** Feedback collected from actual users, providing valuable insights into areas for improvement and new requirements.

1.1 Testing Techniques and Types

1.1.1 Function Testing

Technique Objective:	Exercise the Tool Shop Application's functionality, including navigation through categories, adding tools to the cart, placing orders, and managing user accounts, to observe and log application behavior.
Technique:	Execute each use-case scenario's individual use-case flows or functions and features, using valid and invalid data, to verify that: ● The expected results occur when valid data is used (e.g., placing an order with sufficient stock). ● The appropriate error or warning messages are displayed when invalid data is used (e.g., trying to purchase out-of-stock items). ● Each business rule is properly applied (e.g., discounts for bulk orders)

Oracles:	● **Requirement Specifications**: Validate outcomes against the documented requirements, such as product catalog browsing, checkout process, and stock management rules. ● **Test Case Expected Results**: Compare actual results against predefined expected outcomes for each use-case scenario, such as correct pricing and tax calculations. ● **Automated Verification**: Use tools like Katalon to verify whether actions like adding items to the cart or applying coupon codes yield the correct results. ● **System Logs and Outputs**: Analyze server logs to ensure correct processing of orders, payment, and inventory adjustments.
Required Tools:	● **Automation Tool**: ○ Katalon, for automating test cases like user login, item search, ○ Cypress for cart content and checkout processes ● **Data-Generation Tool**: A tool like ChatGPT or create automation source code to generate realistic test data for customer accounts, orders, and product inventories. ● **Installation-Monitoring Tools**: Docker to ensure consistency in deployment and monitoring of system configurations. ● **Base Configuration Imager**: To standardize test environments across devices and users.
Success Criteria:	● All key use-case scenarios (e.g., searching for products, placing an order, managing accounts) are tested successfully. ● All major application features, such as cart management, checkout, and inventory updates, function as expected without critical issues. ● Business rules, like stock level checks and discount calculations, are correctly implemented. ● Invalid data inputs (e.g., exceeding available stock or using an expired coupon) trigger appropriate error or warning messages.
Special Considerations:	● Tool Setup and Integration: Ensure that Katalon and Docker are properly configured for seamless testing. ● Data Availability: Ensure sufficient data is available for testing scenarios like bulk orders and various user roles (e.g., admin vs. customer). ● Environment Variability: Consistent test environments must be maintained across all team members and devices for reliable test results. ● Environment Variability: Consistent test environments must be maintained across all team members and devices for reliable test results. ● Dependency on Development Progress: Certain features, such as advanced payment integration, may need to be completed before testing can commence.

1.1.1 User Interface Testing

Technique Objective:	The goal is to exercise the following elements of the application to observe and log standards conformance and target behavior: ● Navigation through the target-of-test reflecting business functions and requirements, including window-to-window, field-to-field, and the use of access methods (e.g., tab keys, mouse movements, and accelerator keys). ● Window objects and characteristics such as menus, size, position, state, and focus
Technique:	● Create or modify tests for each window to verify proper navigation and object states for each application window and object. ● The tests should simulate user interactions, such as clicking, scrolling, and typing in various fields, ensuring that all objects respond correctly. ● Navigation should be tested across all UI elements to ensure seamless transitions and user-friendly flow.
Oracles:	● **Visual Comparison**: Verify that the user interface matches the expected design and layout as per the design specifications or wireframes. This will be done visually by comparing screenshots with the approved design. ● **Automated Checks**: Use automated tests to check for elements like menu accessibility, button functionality, and input fields. For example, Katalon will verify if each element is interactive, visible, and properly positioned. ● **Interaction Verification**: Ensure that each UI element responds as expected (e.g., text boxes accept input, dropdowns show choices) and the correct error or confirmation message appears when invalid or valid data is entered. ● **Standards Compliance**: Check whether UI elements conform to corporate or industry standards (e.g., color scheme, fonts, consistency across screens)
Required Tools:	● **Browser Stack**: For cross-browser testing, ensuring that the UI behaves consistently across different platforms and browsers. ● **Katalon**: For automation of user interactions such as clicks, typing, and navigation between elements.

Success Criteria:	The technique supports the testing of each major screen or window that will be used extensively by the end user, ensuring that: ● Navigation is intuitive and adheres to the expected user flow. ● All UI elements (buttons, input fields, dropdowns, etc.) function as intended, providing the appropriate feedback. ● The UI layout is consistent and visually conforms to design specifications. ● The UI meets accessibility standards (e.g., keyboard navigation, screen reader compatibility).
Special Considerations:	Not all properties for custom and third-party objects (such as certain JavaScript-based elements or embedded widgets) may be accessible for automated testing, and manual verification might be necessary for these. UI tests should also account for responsiveness (ensuring the UI adapts to different screen sizes). Some complex UI elements may require deeper investigation if they do not behave consistently across all devices and platforms.

5.1.1 Load Testing

Technique Objective:	Exercise designated transactions or business cases under varying workload conditions to observe and log target behavior and system performance data under load conditions.
Technique:	Ensure that the system can handle anticipated user activity and workload without compromising functionality, reliability, or user experience when system receives a huge amount of requests (GET, POST)
Oracles:	● **Validation**: Confirms that the system meets performance criteria. ● **Identification**: Highlights performance bottlenecks, resource issues, or scaling inefficiencies. ● **Decision Making**: Provides data to support go/no-go decisions for production deployment.
Required Tools:	● **Test tool:** Performance testing tool like Jmeter, LoadRunner,... to test and collect results in such ways: graph, report. ● **Data generation tool**: A tool like ChatGPT or create automation source code to generate realistic test data for customer accounts, orders, and product inventories. ● **Installation-monitoring tools**: Docker to ensure consistency in deployment and monitoring of system configurations. ● **Base Configuration Imager**: To standardize test environments across devices and users.

Success Criteria:	The system after testing meets the following requirements: ● Average response time: < 2 seconds ● Error rate: < 1% ● Throughput: 50 requests/sec
Special Considerations:	● Load testing should be performed on a dedicated machine or at a dedicated time. This permits full control and accurate measurement. ● The databases used for load testing should be either actual size or scaled equally.

5.1.2 Stress Testing

Technique Objective:	Push the system beyond its anticipated workload capacity to observe and log its behavior under extreme or abnormal conditions. Stress testing aims to identify breaking points, failure modes, and system recovery mechanisms when subjected to excessive demands, such as high transaction volumes or minimal available resources.
Technique:	Ensure that the system can handle extreme user activity or workload scenarios without compromising its critical functionalities. Stress testing involves simulating conditions such as: ● Receiving a **very large number of requests** (e.g., GET, POST) in a short period. ● Severely **limiting available server resources**, such as memory, disk space, or CPU cycles. ● Concurrent execution of **identical or conflicting transactions** by multiple users. ● Testing under **unexpected transaction mixes** or data access patterns.
Oracles:	**Validation**: Confirms the system's ability to withstand stress conditions without catastrophic failure or data corruption. **Failure Analysis**: Highlights system components that fail first or exhibit instability under stress. **Recovery Insight**: Evaluates the system's capacity to recover gracefully after stress conditions subside.
Required Tools:	● **Stress Simulation Tools**:JMeter ● **Resource Limitation Tools**: Docker ● **Data Generation Tools**: Faker in Python

Success Criteria:	The system must satisfy the following criteria during or after stress testing: 1. **Controlled Failure**: System components must degrade gracefully without crashing entirely. 2. **No Critical Data Loss**: Ensure data integrity is preserved even under extreme conditions. 3. **Resource Utilization Analysis**: Provide detailed insights into resource bottlenecks for optimization.
Special Considerations:	Stress testing should be conducted in a controlled environment with dedicated machines to avoid interference with other processes. Databases and test environments should closely mimic the real-world production scale. Network stress may require tools that simulate high traffic, such as packet injection or flooding techniques. Ensure adequate logging mechanisms to capture all relevant data during stress conditions.

5.1.3 Spike Testing

Technique Objective:	To observe and log system behavior under high-volume scenarios, including: ● Maximum number of simultaneous clients performing worst-case transactions. ● Maximum database size reached with simultaneous queries and report transactions.
Technique:	● Use tests from Performance Profiling or Load Testing. ● Simulate maximum clients executing high-load scenarios for extended periods. ● Utilize representative data to create scaled database conditions. ● Execute simultaneous queries and reports to simulate peak conditions.
Oracles:	● Log and monitor response times, throughput, and error rates. ● Verify data accuracy in query and report results under high loads ● Ensure no unexpected crashes, stalls, or data corruption. ● Compare actual results to expected outcomes, flagging deviations.
Required Tools:	● Test Script Automation Tool (JMeter). ● Monitoring tools (CPU, memory, disk usage, etc.). ● Data-generation tools for CSV Config
Success Criteria:	● Large user and data volume emulated successfully. ● System maintains stability during extended testing under high volume. ● No critical failures or significant performance degradation observed. ● Clear insights gained about system limits and bottlenecks.
Special Considerations:	● Define acceptable duration for high-volume conditions (e.g., 2 hours). ● Consider recovery time and resource replenishment post-test.

5.1.4 API Testing

Technique Objective:	Exercise the target APIs under high-volume scenarios: ● Exercise the API under various input scenarios to validate correct functionality, schema compliance, error handling, and response codes without focusing on performance testing.
Technique:	● Use functional testing methods to validate inputs and outputs. ● Perform negative testing by providing invalid inputs to check error handling. ● Ensure schema compliance by comparing actual responses to expected formats. ● Validate edge cases like empty inputs or extreme values.
Oracles:	● Automated validations in Postman using assertions. ● Manual verification of response content, status codes, and headers. ● Validation of JSON response structure and data against expected schema using tools like JSON Schema Validator.
Required Tools:	● Postman for executing test cases and verifying responses. ● Mock servers (if necessary) to simulate API responses. ● JSON Schema Validator for automated schema checks.
Success Criteria:	● All test cases pass with expected outputs. ● The API handles valid inputs correctly and rejects invalid inputs gracefully. ● The response adheres to the documented schema. ● Error messages are clear and descriptive.
Special Considerations:	● Ensure test cases cover boundary conditions and edge cases. ● Focus on functional correctness rather than load or stress scenarios. ● Validate behavior for optional query parameters (e.g., when omitted or partially used).

6. Entry and Exit Criteria

6.1 Test Plan

6.1.1 Test Plan Entry Criteria

- All development and coding tasks for the Tool Shop Application have been completed to the point where the first round of testing (e.g., functional testing) can begin.
- The required testing environment, including hardware, software, and network configurations, has been successfully set up and is ready for testing

6.1.2 Test Plan Exit Criteria

- All planned testing phases, such as functional, integration, UI, and regression testing, have been executed according to the schedule.
- All high-priority defects and critical issues have been identified and resolved, or appropriate mitigation strategies are in place (e.g., workarounds or deferrals).
- All major functionality and user scenarios have been tested. The test cases executed must cover at least 95% of the application's major features, based on the defined requirements.

- All test results, including pass/fail status, issues encountered, and the final assessment of the application, have been documented
- If included in the plan, all critical performance and stress testing scenarios have been executed, and results are satisfactory

6.1.3 Suspension and Resumption Criteria

- Suspension Criteria: If your team members report that there are 40% of test cases failed, you should suspend testing until the development team fixes all the failed cases
- Delays in obtaining necessary resources, such as test data, development fixes, or third-party components, may necessitate the suspension of testing.
- If key resources (testers, equipment, or tools) become unavailable due to unforeseen circumstances (e.g., team member illness, tool failure), testing may be paused.

Testing can be resumed once:

- The testing environment, tools, or required resources become available again after a temporary disruption.
- If data, testers, or other resources were unavailable, they are now ready for testing, and the necessary preparations have been completed.
- When the project resumes its normal schedule after delays or reprioritization, testing will continue according to the updated timeline.

6.2 Test Cycles

6.2.1 Test Cycle Entry Criteria

- **Build Readiness**:
  - No critical defects are present in the current build.
- **Test Environment Setup**:
  - Load testing environment is configured and matches the production environment.
  - Required tools are installed and operational.
- **Test Data**:
  - Test data sets are prepared, validated, and loaded into the environment.
- **Resources Availability**:
  - Testers, tools, and necessary documentation are in place.

6.2.2 Test Cycle Exit Criteria

- **Test Coverage**:
  - All planned test scenarios have been executed
- **Defect Threshold**:
  - All critical and high-severity defects have been resolved or deferred with stakeholder approval.
  - Medium and low-severity defects are documented for future cycles.
- **Report Submission**:
  - Test results, logs, and reports have been submitted for review.

6.2.3 Test Cycle Abnormal Termination

- **Critical Issues in the Build**:
  - Discovery of a critical defect or blocker that prevents further testing.
  - Application crashes or becomes unresponsive under testing.
- **Environment Failures**:
  - Test environment is unstable or does not replicate the production setup.
  - Unavailability of critical resources such as servers or tools.
- **Scope Changes**:

- Significant changes to the test requirements or application functionality.

7. Deliverables (Hường)

7.1 Test Evaluation Summaries

Provide concise reports summarizing the results of the test effort, including key findings, defects identified, and any deviations from expected outcomes. These summaries will be produced weekly and at the conclusion of major testing phases to ensure stakeholders remain informed about progress and system performance.

1.1 Reporting on Test Coverage

Test coverage reports will detail the percentage of requirements, functions, and test cases executed, passed, and failed. These reports will be generated using tools such as JIRA, Zephyr, or TestRail, and will be produced bi-weekly. Visual elements like charts and graphs will be included for clarity.

1.2 Perceived Quality Reports

Perceived quality reports will include metrics and insights on user satisfaction, usability issues, and overall system performance. These will be based on metrics such as defect density, resolution time, and user feedback. Reports will be produced monthly using tools like Grafana or Google Sheets for analysis and documentation.

1.3 Incident Logs and Change Requests

Incidents and change requests will be tracked and managed using tools like JIRA. Each incident will be recorded with details such as the defect description, steps to reproduce, severity, priority, and resolution status. Change requests will also include impact analysis and approval workflows.

1.4 Smoke Test Suite and Supporting Test Scripts

The smoke test suite will consist of automated scripts designed to validate critical functionalities in new builds. Test scripts will be maintained using tools like Selenium, Playwright, or TestNG, ensuring their compatibility with subsequent product iterations.

1.5 Additional Work Products

7.1.1 Additional Automated Functional Test Scripts

Automated test scripts will be provided as source code files or compiled repositories, developed using tools like Katalon, or similar frameworks. These will be thoroughly documented for maintainability.

7.1.2 Test Guidelines

Guidelines will cover best practices, fault models, automation standards, and reusable test patterns. These documents will be shared as collaborative resources, providing structured support for the testing team.

7.1.3 Traceability Matrices

Traceability matrices will be created using tools such as Microsoft Excel. These matrices will map test cases to requirements, ensuring complete coverage and alignment with project objectives.

8. Testing Workflow (Thư)

The testing workflow for the Tool Shop Application will follow a structured, iterative process that aligns with the Rational Unified Process (RUP) principles. This workflow includes specific phases and iterations, with activities tailored to the project's objectives and development timeline.

Overview of Testing Workflow

The test team will execute the testing process in alignment with the project's Development Case, which has customized the base RUP testing workflow as follows:

Step 1: Requirements Analysis:

- Collaborate with stakeholders to finalize testable requirements.

- Identify and document use cases and business rules.
- Define test objectives and success criteria for each feature.

Step 2: Test Planning:

- Develop a detailed Test Plan based on the requirements and use cases.
- Define the scope, tools, environment setup, and test deliverables.
- Assign roles and responsibilities within the test team.

Step 3: Test Design:

- Create test scenarios, test cases, and test scripts using tools like Katalon.
- Map test cases to requirements to ensure traceability.
- Review and validate test artifacts with relevant stakeholders.

Step 4: Test Environment Setup:

- Configure test environments using Docker for consistency.
- Load sample data generated by tools like Mockaroo.
- Validate the environment's readiness with smoke tests.

Step 5: Test Execution:

- Conduct functional, integration, and regression testing based on the Test Plan.
- Log defects and anomalies in the defect management tool.
- Re-run tests after defect fixes to validate corrections.

Step 6: Test Reporting:

- Generate test execution reports summarizing test results and defect trends.
- Provide feedback to stakeholders about the quality and readiness of the application.

Step 7: Test Closure:

- Ensure all planned test cases are executed, or document any deviations.
- Archive test artifacts and data for future reference.
- Conduct a post-mortem review to identify process improvements.

Task Management

Task management will be handled manually and updated as the project progresses. Specific details of testing tasks will be managed and documented in:

- **Project Management Tools**: Tasks and schedules will be tracked using tools like file **Tracking.docx**
- **Weekly checkin**: Tasks will be discussed and updated during weekly team meetings.

Phase and Iteration Testing

- **Phase 1**:
  - Conduct functional testing for essential use cases like user login, product search, and cart operations.
  - Generate data by tool or create data automatically by source code
- **Phase 2**:
  - Perform API testing
  - Begin exploratory testing to identify edge-case scenarios.
- **Phase 3**:
  - Conduct performance and stress testing for critical operations, such as handling bulk orders or concurrent users.

- Perform automation testing for functional test cases of workflows

9. Environmental Needs

9.1 Base System Hardware

The following table sets forth the system resources for the test effort presented in this *Test Plan*.

System Resources		
Resource	**Quantity**	**Name and Type**
Database Server	1	MySQL Database Server
Network or Subnet	1	Local Area Network (LAN)
Server Name	1	TestDBServer
Database Name	1	ToolShop
Client Test PCs	4	Windows 10 or higher, Intel Core i5, 8GB RAM, 256GB SSD
Test Repository	1	Cloud-based Test Repository (e.g., GitHub)
Test Development PCs	4	Windows 10 or higher, Intel Core i7, 16GB RAM, 512GB SSD

9.2 Base Software Elements in the Test Environment

The following base software elements are required in the test environment for this *Test Plan*.

Software Element Name	Version	Type and Other Notes
Windows	10/11	Operating System
Internet Explorer	11	Internet Browser
Katalon	any	Testing tool
Cypress	any	Testing tool
Postman	any	Testing tool

Software Element Name	Version	Type and Other Notes
JMeter	any	Testing tool
Docker	any	Deployment tool
Visual Studio Code	any	IDE

9.3 Productivity and Support Tools

The following tools will be employed to support the test process for this *Test Plan*.

Tool Category or Type	Tool Brand Name
Test Management	Microsoft Excel
Defect Tracking	Microsoft Excel
ASQ Tool for functional testing	Katalon
ASQ Tool for performance testing	JMetter
Project Management	Google Drive
DBMS tools	MySQL

9.4 Test Environment Configurations

The following Test Environment Configurations needs to be provided and supported for this project.

Configuration Name	Description	Implemented in Physical Configuration
Average user configuration	Simulate typical end-user system with 8GB RAM, Intel i5 processor	Standard desktop environment, standard internet connection
Minimal configuration supported	Test the minimum hardware that can run the application effectively	4GB RAM, Intel i3 processor, 128GB SSD
Visually and mobility challenged	Configure accessibility tools such as screen readers and custom keyboard navigation	Windows 10 with screen reader software (e.g., NVDA) and keyboard shortcuts
International Double Byte OS	Test multi-language support and characters in languages such as Chinese, Japanese	Operating System supporting Double Byte Characters
Network installation (not client)	Test installation and use of application from a network share or remote server	Server-based application access via network

10. Responsibilities, Staffing, and Training Needs

10.1 People and Roles

This table shows the staffing assumptions for the test effort.

Human Resources		
Role	**Minimum Resources Recommended** (number of full-time roles allocated)	**Specific Responsibilities or Comments**
Test Manager		Provides management oversight. Responsibilities include: ● planning and logistics ● agree mission ● identify motivators ● acquire appropriate resources ● present management reporting ● advocate the interests of test ● evaluate effectiveness of test effort
Test Analyst		Identifies and defines the specific tests to be conducted. Responsibilities include: ● identify test ideas ● define test details ● determine test results ● document change requests ● evaluate product quality
Test Designer		Defines the technical approach to the implementation of the test effort. Responsibilities include: ● define test approach ● define test automation architecture ● verify test techniques ● define testability elements ● structure test implementation
Tester		Implements and executes the tests. Responsibilities include: ● implement tests and test suites ● execute test suites ● log results ● analyze and recover from test failures ● document incidents

10.2 Staffing and Training Needs

This section outlines how to approach staffing and training the test roles for the project.

- **Skill needs**
  - **Test Manager**:
    - Project management, risk management, and communication.
    - Familiarity with test management tools (e.g., Jira, TestRail).
  - **Test Designer:**
    - Analytical skills for designing effective test cases.
    - Expertise in domain-specific testing techniques.
  - **Test Analyst:**
    - Requirement analysis, defect reporting, and test case creation.
    - Knowledge of relevant testing tools and frameworks.
  - **Tester:**
    - Attention to detail and knowledge of testing processes.
    - Ability to follow test scripts and identify deviations.
- **Training needs:**
  - **Test Manager:**
    - Risk management and advanced reporting techniques.
    - Training on creating and maintaining test schedules using management tools.
  - **Test Designer:**
    - Advanced techniques in test case design and reusable components.
    - Domain-specific training for better alignment with project requirements.
  - **Test Analyst:**
    - Defect management training and requirement traceability.
    - Hands-on workshops for defect logging and tracking tools.
  - **Tester:**
    - Training on executing automated and manual test cases.
    - Basic understanding of defect logging tools and test reporting.

11. Iteration Milestones

Milestone	Planned Start Date	Actual Start Date	Planned End Date	Actual End Date
Iteration Plan agreed	6/12/2024	6/12/2024		
Iteration starts	6/12/2024	6/12/2024		
Requirements 1- Create Test Plan	8/12/2024	8/12/2024	10/12/2024	10/12/2024
Requirements 2- Functional Testing	10/12/2024	10/12/2024	15/12/2024	15/12/2024
Requirements 3 - Data generation	16/12/2024	16/12/2024	18/12/2024	18/12/2024
Requirements 4 - GUI Testing + Usability testing	20/12/2024	20/12/2024	23/12/2024	23/12/2024

Milestone	Planned Start Date	Actual Start Date	Planned End Date	Actual End Date
Requirements 7 - API Testing	24/12/2024	24/12/2024	26/12/2024	26/12/2024
Requirements 5 - Automation testing	26/12/2024	26/12/2024	30/12/2024	30/12/2024
Requirements 6 - Performance Testing	31/12/2024	31/12/2024	1/1/2024	1/1/2024
Requirements 7 - Test report	2/1/2024	2/1/2024	3/1/2024	3/1/2024
Iteration Assessment review			3/1/2024	3/1/2024
Iteration ends			3/1/2024	3/1/2024

12. Risks, Dependencies, Assumptions, and Constraints

Risk	Mitigation Strategy	Contingency (Risk is realized)
Prerequisite entry criteria is not met.	<Tester> will define the prerequisites that must be met before Load Testing can start. <Customer> will endeavor to meet prerequisites indicated by <Tester>.	● Meet outstanding prerequisites ● Consider Load Test Failure
Test data proves to be inadequate.	<Customer> will ensure a full set of suitable and protected test data is available. <Tester> will indicate what is required and will verify the suitability of test data.	● Redefine test data ● Review Test Plan and modify ● components (that is, scripts) ● Consider Load Test Failure
Database requires refresh.	<System Admin> will endeavor to ensure the Database is regularly refreshed as required by <Tester>.	● Restore data and restart ● Clear Database

Dependency between	Potential Impact of Dependency	Owners
Availability of system environment	Delays in setting up the environment could delay test execution.	System Admin, Test Team
Test data preparation	Lack of suitable data might result in testing delays or inaccurate results.	Customer, Tester
Network connectivity	Network issues can impact the test results, especially in stress and load testing scenarios.	IT Support, Test Team

Assumption to be proven	Impact of Assumption being incorrect	Owners
The system will handle high concurrency levels without degradation.	If the system doesn't handle high concurrency, it may fail during load and stress testing.	Tester, Development Team
Test data will be realistic and complete.	If test data is incomplete or inaccurate, it can lead to unrepresentative testing results.	Tester, Customer

Assumption to be proven	Impact of Assumption being incorrect	Owners
Environment configurations will remain stable throughout the testing period.	Instability in the environment might result in inconsistent testing results and delays.	System Admin, Tester

Constraint on	Impact Constraint has on test effort	Owners
Time available for testing	Limited time might restrict the ability to conduct thorough testing, especially for edge cases.	Test Team, Project Manager
Resource availability (e.g., hardware)	If required resources are unavailable, testing may be delayed or limited.	System Admin, Project Manager
Dependencies on other teams for data or environment setup	Delays in setup may impact the start of the testing process and the overall timeline.	Test Team, Customer

13. Management Process and Procedures

13.1 Measuring and Assessing the Extent of Testing

- **Test Metrics**:
  - Percentage of test cases executed.
  - Percentage of test cases passed, failed, or blocked.
  - Defect density (defects per test case or feature).
  - Test coverage (percentage of requirements covered by test cases).
- **Tracking Tools**:
  - Use test management tools (e.g., TestRail, Jira, Google Docs) to track progress.
- **Periodic Reporting**:
  - Generate daily or weekly test execution reports.
- **Exit Criteria Validation**:
  - Compare results against predefined exit criteria to assess testing completeness.

13.2 Assessing the Deliverables of this Test Plan

- **Test Cases**:
  - Ensure all test cases are reviewed, approved, and traceable to requirements.
- **Defect Reports**:
  - Verify that all high and critical defects are resolved or deferred with stakeholder approval.
- **Test Execution Reports**:
  - Review execution logs, defect reports, and coverage metrics to validate results.
- **Approval Meetings**:
  - Conduct formal reviews with stakeholders to assess the completeness and quality of deliverables.

13.3 Problem Reporting, Escalation, and Issue Resolution

- **Problem Reporting**:
  - Testers report issues in the defect tracking tool with detailed descriptions and logs.
- **Escalation Process**:
  - Issues unresolved within 24 hours and Critical issues impacting timelines are escalated to the Test Manager.
- **Resolution Process**:
  - Root cause analysis is conducted by the assigned team.
  - Resolutions are verified through regression testing.
- **Communication**:

- Daily stand-ups and progress reports ensure transparency on issue resolution.

13.4 Managing Test Cycles

- **Planning**:
  - Define objectives, test scenarios, and resource allocation for each test cycle.
- **Cycle Reviews**:
  - Conduct end-of-cycle reviews to evaluate progress and identify areas for improvement.
- **Continuous Improvement**:
  - Apply lessons learned to optimize subsequent test cycles.

13.5 Traceability Strategies

- **Coverage of Testing against Specifications**:
  - Maintain a requirements traceability matrix (RTM) linking test cases to specific requirements.
  - Ensure complete test coverage through periodic reviews of the RTM.
- **Motivations for Testing**:
  - Document the purpose of each test case (e.g., functional validation, regression, performance).
  - Periodically review test relevance and retire obsolete test cases.
- **Software Design Elements**:
  - Map test cases to specific design components to identify tests affected by design changes.
  - Use version control to track design updates and trigger re-testing as necessary.
- **Resulting Change Requests**:
  - Link defects to change requests and associated test cases.
  - Re-run tests post-change implementation to verify resolution.

13.6 Approval and Signoff

- **Approval Process**:
  - The Test Plan is reviewed and approved in the following stages:
    - Initial draft reviewed by Test Manager.
    - Final approval by everyone in the team.
- **Signoff Authority**:
  - **Test Manager**: [Name], Responsible for overall test strategy.
  - **QA Lead**: [Name], Verifies the completeness of test cases and execution reports.
  - **Project Manager**: [Name], Ensures alignment with project goals.
  - **Key Stakeholders**: [Names], Provide final signoff based on business requirements.