SQA Test Plan Guideline

<Credit Card Subsystem of Online Bank System>

SQA Test Plan (STP)

Version: < 1.2 >

< 2024/12/9 >

Revision History

Date	Version	Description	Prepared / Revised By
2024/12/9	1.0	First version of the SQA	

Table of Contents

1. Introduction

1.1 Document Terminology and Acronyms

Acronyms	Documentation Terminology Explained
SQA (Software Quality Assurance)	A set of activities ensuring software processes, procedures, and standards are followed to deliver quality software.
QA (Quality Assurance)	Broad term encompassing processes and activities to ensure a product meets predefined quality standards and requirements.
Use Case	A specific scenario describing how a user interacts with the system to achieve a goal, representing applications of the software.
Functionality Testing	Testing focused on verifying that the application behaves according to the specified requirements and functions as expected.
Black-box Testing	Testing approach where the tester evaluates the system's functionality without knowledge of its internal workings or code structure.
White-box Testing	Testing approach where the tester has full visibility of the code and evaluates internal logic and structure.
Smoke Testing	A preliminary test to confirm basic system functionality before more detailed testing begins.
Regression Testing	Re-execution of test cases to ensure that new changes have not negatively impacted existing functionality.

Integration Testing	Testing conducted to ensure that different modules or systems interact correctly with each other.
Stress Testing	Testing to determine the system's behavior under extreme or breaking-point conditions
Load Testing	Testing to evaluate how the system performs under expected user load.
Bug	An error, flaw, or defect in software that causes it to produce an incorrect or unexpected result or behave in unintended ways.
Spring boot	A Java-based framework used to create stand-alone, production-ready applications. It can simplify the development process.
Vue	A JavaScript framework for building user interfaces and single-page applications (SPAs). Known for its simplicity, flexibility.
Defect Density	The number of defects found in a software module relative to its size (e.g., defects per thousand lines of code).
Test Coverage	A measure of how much of the application has been tested, often expressed as a percentage of total requirements, lines of code, or functionality.

1.2 References

[1] Software Engineering: A Practitioner's Approach 8th Edition, Roger S.Pressman, McGraw-Hill Education

[2] Software Requirements 3rd Edition, Karl Wiegers, Joy Beatty, Microsoft Press

[3] ISO 9000 for Software

[4] Testing framework documentation

[5] "Online Banking System" Requirements Specification Report

[6] "Online Banking System" Project Design Report

2. Target Test Items

*[Provide a high-level overview of the major testing scope. List what will be included and record what will explicitly **not** be included.]*

2.1 Test Inclusions

- **Module Functional Testing:**
  - Verify whether each module meets all its functional requirements.
  - Conduct detailed testing of each module to ensure all functionality, features, and conditions are addressed.
  - Include all functional combinations and edge cases in the test scenarios to maximize test coverage and detect potential issues early.
- **Boundary Value Testing:**
  - Evaluate the system's capability to handle boundary values effectively, ensuring stable behavior.
  - Test data fields and functionalities with clearly defined boundary conditions by inputting boundary values to observe system responses.
  - Verify that valid boundary values are processed correctly and invalid inputs are gracefully handled with appropriate error messages or actions.
- **Integration Testing**
  - Ensure the modules function cohesively in an integrated environment.
  - Observe interactions between the frontend, backend, and database to validate end-to-end workflows.
  - Confirm data consistency and proper synchronization between modules after integrated operations.
- **Performance Testing:**
  - Response Time: Measuring the time taken to respond to user inputs and complete.
  - Error Handling: Verifying how the system reacts to various failure scenarios, ensuring clear and actionable error messages.
  - Load and Stress Testing: Evaluating system performance under normal and peak load conditions to assess stability and scalability.
- **Security Testing:**
  - Simulate common attack scenarios like SQL injection and Cross-Site Scripting (XSS) to identify vulnerabilities.
  - Validate the implementation of security measures outlined in the requirements analysis, such as account login protections, access controls,

and secure data handling.

- **Cross-Platform Compatibility Testing**
  - Ensure the system functions consistently across various platforms, including Windows, macOS, Android, and iOS.
- **Regression Testing**
  - Re-test previously validated functionalities to ensure that new updates or changes have not introduced defects.
  - Focus on maintaining the integrity of existing features during iterative development.

2.2 Test Exclusions

- **Out-of-Scope Features:**
  - Features unrelated to the credit card module, such as loan processing, mortgage management, or investment tracking, are not tested.
  - Modules that have been marked for future phases of development or are under maintenance.
- **Unsupported Platforms:**
  - Any device or operating system not listed in the compatibility testing section, such as obsolete versions of Windows, macOS, or outdated browsers like Internet Explorer.
  - Certain mobile platforms, such as older Android or iOS versions no longer officially supported by the system requirements.
- **Legacy System Interfaces:**
  - Testing integration with older systems that have been deprecated or are no longer actively used in production.
  - APIs or services scheduled for replacement and no longer part of the main workflow.
- **Non-priority Edge Cases:**
  - Highly improbable user scenarios, such as simultaneous use of the system on hundreds of unique browsers under non-standard configurations.
  - Rare or exotic combinations of inputs and outputs that are unlikely to occur in practical use.
- **Performance Under Extreme Conditions:**
  - Stress testing scenarios that simulate millions of concurrent users, which exceed the system's intended scalability targets.
  - Load testing on environments or hardware that does not meet the

baseline specifications defined in the test environment requirements.

3. Test Approach

[The Test Approach presents an overview of the recommended strategy for analyzing, designing, implementing and executing the required tests.]

3.1 Test Identification and Justification.

1. **Impact Analysis:**
    - Review recent code changes or updates to determine which components are affected and require testing.
    - Focus on modules with significant new implementations or modifications, such as the addition of a new credit card management feature.
    - Analyze past defect trends to identify areas prone to issues.
2. **Requirement Mapping:**
    - Create a one-to-one mapping of test cases to functional and non-functional requirements outlined in the documentation.
    - Ensure each critical requirement is tested, including security, usability, and performance standards.
3. **Risk Assessment:**
    - Identify high-risk areas, such as payment processing, which could have severe consequences if defects arise.
    - Emphasize security-related tests to ensure compliance with regulatory standards.
    - Assess potential failure points in integration workflows, such as data sync between frontend and backend systems.
4. **Customer Priorities:**
    - Highlight features and workflows most critical to user satisfaction, such as the speed and reliability of fund transfers or the accuracy of credit card statements.
    - Prioritize scenarios heavily used by end-users, e.g., querying transaction history.
5. **Test Types Justified by Importance:**
    - **Functional Testing:** Verifying that the system behaves as required across all specified scenarios.
    - **Regression Testing:** Ensuring new changes do not break existing

functionality.

- **Integration Testing:** Testing cross-module and system-wide functionality to confirm seamless interactions.
- **Security Testing:** Validating protection against potential threats like unauthorized access.

Justification:

- Optimizing test efforts to deliver maximum coverage within the available time frame.
- Aligning testing priorities with business goals, technical risks, and user requirements.
- Addressing critical workflows to ensure system reliability and customer satisfaction.

3.2 Conducting Tests

3.2.1 *Build Acceptance Testing （smoke）*

Objective:
To verify the stability and readiness of each building of the online banking system (credit card part) for further testing. This is a preliminary test to ensure the build is functional enough to proceed to in-depth testing phases.

Scope:

- Focus on critical workflows and features (e.g., login, account access, and fund transfers).
- Ensure essential functionalities are operational without errors.

Test Approach:

a. **Test Environment:**
    - Set up the system in the staging environment, mirroring production settings.
    - Use sanitized or dummy test data.

b. **Key Areas Tested:**
    - **Login and Authentication:** Verify that users can log in securely using valid credentials.
    - **Homepage Accessibility:** Ensure key elements load, including account summaries and navigation menus.
    - **Core Features:** Check basic functionality of credit cards, including application, payment and management.

    - **Error Handling:** Verify appropriate error messages for failed logins or system errors.

c. **Pass/Fail Criteria:**
    - The build is marked as "pass" if no critical or blocking issues are found.
    - A "fail" indicates that significant bugs prevent further testing.

d. **Tools/Resources:**
    - Smoke test scripts prepared by the QA team.

e. **Deliverables:**
    - A build acceptance report summarizing results.
    - Issues logged into the defect tracking system.

3.2.2 ***Functionality testing***

Objective:
To validate the credit card part of the online banking system's functionality against the defined requirements and ensure all features perform as intended.

Scope:

- Comprehensive testing of all features across modules of the credit card part, including application, payment/repayment, management and querying functionalities.

Test Approach:

a. **Test Planning:**
    - Identify test cases based on the functional requirements document (FRD).
    - Prioritize high-risk and high-impact functionalities.

b. **Test Execution:**
    - Perform **positive testing** to ensure functionalities work as expected under normal conditions.
    - Perform **negative testing** to verify the system handles invalid inputs gracefully.

c. **Key Features Tested:**
    - **Application:** Apply for a credit card.
    - **Payment**: Use credit card to pay or make transactions.
    - **Reyment**: Reypay the loan money of the card.
    - **Management**: Mange the basic information of a credit card including password, user information, reporting loss, cancellation etc.
    - **Querying**: Query transaction flow.

d. **Defect Management:**

- Document any discrepancies between expected and actual outcomes in a defect tracking system.
- Assign severity levels (Critical, Major, Minor) to defects.

e. **Tools/Resources:**

- Functional test cases written in Feishu.
- Automation scripts (if applicable) for repetitive tests.

f. **Pass/Fail Criteria:**

- Functionality is deemed successful if it meets all defined acceptance criteria without critical defects.

g. **Deliverables:**

- Functional test case execution report.
- List of identified defects with severity and resolution status.

3.2.3 *Regression Testing*

- Regression Testing is to confirm that a new release of software version doesn't adversely affected existing features. It's done to make sure that new changes should have no side effects on existing functionalities. And it ensures that the features still works.
- After each bug is identified and then fixed, we must perform regression testing to ensure our changes do not affect the rest of the system.

3.2.4 *Volume/Performance/Failover testing*

3.3 Test Automation Strategy

3.4 Defect Management

3.5 Test Metrics

3.6 Reporting

4. Entry and Exit Criteria

4.1 Test Execution Entry Criteria

- Test Plan has been reviewed and base-lined
- Test Cases have been reviewed and base-lined.
    - Any QA environment requirements have been clearly defined.
- Build Verification Testing and the Smoke testing is successful.

4.2 Test Execution Exit Criteria

- Test Case execution status
- Sign off.
- Defects

4.3 Suspension and Resumption Criteria

5. Environmental Needs

5.1 System Hardware

For server of the back-end, we have the following requirements:

- Power supply is sufficient
- Memory is more than 16GB
- CPU main frequency is more than 2.6GHZ with more than 10 cores
- Hard disk capacity is greater than 1TB, with hard disk speed more than 5000MB/s
- Network cable can be used normally and has good data transmission capability
- Network card speed is more than 100megabit

For client computer, we have the following requirements:

- Power supply is sufficient
- CPU main frequency is more than 2.0GHZ
- Memory is more than 8GB
- Hard disk capacity is greater than 128GB, with hard disk speed more than

2000MB/s

- Network card speed is more than 100megabit
- Network cable can be used normally and has good data transmission capability
- Mouse, keyboard, monitor, host is not damaged, can be used normally

5.2 Software Elements in the Test Environment

Type	Description
Operation System	Any major distribution, including Ubuntu v20.04+, macOS v11.2.3+, and Windows v10+.
Database	MySQL v8.0.22
Browser	Google Chrome v90+, Firefox v88+, Safari v14+, Edge v91+
Programming Luaguage	Backend Development Environment: Java OpenJDK 18 Frontend Development Environment: JavaScript (latest ES6 specification)

6. Responsibilities, Staffing, and Training Needs

6.1 People and Roles

people	**role**
	test case design, functional test and system test of the "System administrator" module.
	test case design, function test and system test of "Modify credit card" module.
	test case design, function test and system test of "credit card payment, flow query" module.

	test case design, functional test and system test of the module "Repayment, reporting loss and cancellation of credit card".
	test case design, function test and system test of "Query personal Information, Credit Card Examiner query" module.

6.2 Staffing and Training Needs

- **Test staffing:** HongYu Cai, MengYue Lv, Rui Yang, XinYi Zhou, JiaMu Wang
- **Training content:**
  - Systematically read the documentation, and test with a thorough understanding of its functions and system requirements;
  - Learn to master basic unit testing methods;
  - Learn skills in test automation to ensure the quality and reliability of software, such as JMeter;
  - Learn various software testing techniques and best practices to improve testing efficiency;
  - Learn the basic concepts of common security problems and ways to attack systems

7. Key Project/ Phase Milestones

Milestone	**Planned Start Date**	**Actual Start Date**	**Planned End Date**	**Actual End Date**
Project/ Phase starts	2024.12.2	2024.12.2	2024.12.2	2024.12.2
SQA Test Plan agreed	2024.12.3	2024.12.3	2024.12.8	2024.12.8
Testing resources requisitioned	2024.12.9	2024.12.9	2024.12.11	2024.12.11

Testing team training complete	2024.12.12	2024.12.12	2024.12.12	2024.12.12
Requirements baselined	2024.12.13	2024.12.13	2024.12.14	2024.12.14
Test Case Design baselined	2024.12.15	2024.12.15	2024.12.16	2024.12.16
QA – Cycle 1 Build Acceptance Test Execution	2024.12.17	2024.12.17	2024.12.18	2024.12.18
QA – Cycle 1 Functional Test Execution	2024.12.19	2024.12.19	2024.12.19	2024.12.19
QA – Cycle 2 Build Acceptance Test Execution	2024.12.20	2024.12.20	2024.12.21	2024.12.21
QA – Cycle 2 Functional Test Execution	2024.12.22	2024.12.22	2024.12.22	2024.12.22
QA Regression Test Execution	2024.12.23	2024.12.23	2024.12.24	2024.12.24
QA Performance/Failover Test Execution	2024.12.25	2024.12.25	2024.12.25	2024.12.25
QA Final Integrated Build Test Execution	2024.12.26	2024.12.26	2024.12.28	2024.12.28
Project Status Assessment review	2024.12.29	2024.12.29	2024.12.30	2024.12.30

Project/ Phase ends	2025.1.1	2025.1.1	2025.1.3	2025.1.3

8. Risks, Dependencies, Assumptions, and Constraints

8.1 Risks

In this testing project, given its relatively small size and the simplicity of the functionalities implemented, the risk can be considered as low. However, considering the lack of experience within our testing team, it is still necessary to constantly pay attention to avoid the following issues:

- Failure to complete on schedule Countermeasure: Testers should cooperate effectively and promptly to finish the project on time.

- Misunderstanding of requirements Countermeasure: Firstly, everyone should ensure a consistent understanding of the requirements across the entire team, thoroughly read the documentation, and develop a reasonable test plan based on clear use cases.

- Lack of experience with testing tools Establish a technical development group to set up the testing environment and guide each tester on how to use it.

- Specific requirements that cannot be tested in the short term Considering time constraints, high-priority test cases should be tested first.

8.2 Dependencies

- **Availability of Test Environment**: Ensure that the test environment is set up and stable before initiating any testing activities. Any delays or issues in setting up the environment can significantly affect the testing schedule.

- **Quality and Completeness of Test Data**: The availability and quality of realistic test data are critical. Incomplete or incorrect test data can lead to inaccurate test results and necessitate retesting, delaying the project.

- **Approval from Regulatory Bodies**: For certain types of systems, especially those involving financial transactions like a credit card system, regulatory approval may be necessary before full-scale testing can proceed.

- **Documentation Availability**: Comprehensive and up-to-date documentation, including design documents, requirement specifications, and user manuals, must be available for the testers. Lack of proper documentation can hinder the testing process.

- **Hardware and Software Requirements**: Ensure that all necessary hardware and software required for testing are available and compatible with the test environment. Any shortage or incompatibility can delay the testing process.

8.3 Assumptions

Proven assumptions	Impact of incorrect assumptions
Stable and Reliable Test Environment	Frequent downtimes or environmental issues can lead to delays in testing activities and affect the overall progress of the testing plan
Clear and Detailed Test Cases	Incomplete or poorly defined test cases can result in gaps in testing coverage, leading to undetected defects and potential system failures
Availability of Stakeholders for Clarifications	Unavailability of stakeholders can cause delays in resolving ambiguities or making necessary adjustments to the test plan or test cases
Consistent Performance of Third-Party Services	Unexpected issues with third-party services can disrupt testing schedules and require additional testing cycles to ensure compatibility and reliability
Continuous Integration and Deployment (CI/CD)	Issues with CI/CD pipelines can slow down the testing process, delay feedback loops, and hinder the ability to quickly identify and fix issues

8.4 Constraints

- Due to the relative isolation between testers and developers, clients may not have a clear understanding of the overall project situation.

- The timelines for development and testing can differ significantly, so project testing must be strictly implemented based on documents such as software requirements analysis reports, with assumptions made independently.

- The timeline for the testing project is tight, but the complexity of the project is relatively low, so it is possible to reduce the number of tests in appropriate situations to ensure the correct execution of functional testing.

- Due to issues such as long development times and unstandardized development environments, the environments used by developers and testers may differ.