Abdulhaq Zulfiqar(22i-2585)

Shahmeer(22i-1522)

Usman Asif(22i-8802)

SE-G

SQE

Project

Contents

Test Plan Identifier 6

References: 6

Introduction: 6

- Purpose 6
- Objectives 6
- Scope 7

Test Items 7

- Modules to Be Tested 7
  - Account Management 7
  - Transaction Management 7
  - Search and Help Modules 7
  - Database Integration 7
  - Test Classes 7

Software Risk Issues 7

- Database Connection Failure 8
  - Risk: 8
  - Impact: 8
  - Mitigation Strategy: 8
- Complexity of Transaction Logic 8
  - Risk: 8
  - Impact: 8
  - Mitigation Strategy: 8
- User Authentication and Authorization Failures 8
  - Risk: 8
  - Impact: 8
  - Mitigation Strategy: 9
- Data Integrity and Consistency Issues 9
  - Risk: 9
  - Impact: 9
  - Mitigation Strategy: 9

Performance Under Load 9
- Risk: 9
- Impact: 9

Mitigation Strategy: 9
Inadequate Error Handling 9
- Risk: 9
- Impact: 9

Mitigation Strategy: 10
Misunderstanding of Requirements 10
- Risk: 10
- Impact: 10

Mitigation Strategy: 10
Features to Be Tested 10
Functional Features 10
Non-Functional Features 11
Features Not to Be Tested 11
Test Approach 11
Levels of Testing 11
Unit Testing 11
Integration Testing 11
System Testing 11
Database Testing 11
Performance Testing 11
Regression Testing 12
Testing Techniques 12
Black-box Testing: 12
White-box Testing: 12
Boundary Testing: 12
Pass/Fail Criteria 12
Pass Criteria 12
Fail Criteria 12
Suspension Criteria and Resumption Requirements 12

Suspension Criteria 12
1. Critical Defects: 12
2. Lack of Test Data: 12
3. System Unavailability: 12
Resumption Requirements 13
1. Critical Defects Resolved: 13
2. Test Data Availability: 13
3. System Restored: 13
Testing Deliverables 13
Test Case Document: 13
Bug Report: 13
Test Execution Report: 13
Test Summary Report: 13
Remaining Test Tasks 13
Dependencies: 14
Testing Environment 14
Hardware Requirements 14
Software Requirements 14
Data Requirements 15
STAFFING AND TRAINING NEEDS 15
Staffing Needs 15
1. Test Lead 15
2. Test Engineers: 15
3. Database Administrator: 15
4. Performance Testing Specialist: 15
5. QA Analysts: 15
Training Needs 15
Test Engineers: 15
Performance Testing Specialist: 15
Database Administrator: 16
QA Analysts: 16
Roles and Responsibilities 16

Schedule 16
Risks and Contingencies 17
Approval 17
Glossary 17

Test Plan Identifier

Project Name: Core Java Banking System
Test Plan Identifier: BankingSystem-Test-Plan-001
Date of Issue: 2024-12-09
Prepared by: Abdul Haq Zulfiqar

References:

- **Project Plan**: Describes the overall goals, timelines, and scope of the banking system project.
- **Requirements Specifications**: Detailed document outlining the functional and non-functional requirements for the banking system.
- **Design Documents**: High-level and detailed design documents for the system architecture and user interfaces.
- **Test Process Standards**: Internal standards for conducting software tests, including methodologies and reporting guidelines.
- **Tools and Frameworks**: Information on the tools used for testing (e.g., JUnit for unit testing, Apache JMeter for performance testing).
- **Coding Standards**: Reference for the coding practices and guidelines followed in the development of the banking system.

Introduction:

Purpose

This test plan defines the testing approach, objectives, scope, and deliverables for the Core Java Banking System. The aim is to ensure that all features, including account management, transaction handling, database integration, and error handling, meet functional and non-functional requirements.

Objectives

- Validate core functionalities like account creation, deletion, deposits, and withdrawals.
- Ensure database integration works as expected, maintaining data integrity.
- Identify software defects and address them before deployment.
- Confirm the system's performance and scalability.

Scope

This test plan covers all core and auxiliary modules, including database integration, user interactions, and transaction handling. Non-critical features such as theming and splash screens are excluded unless they affect core functionality.

Test Items

Modules to Be Tested

Account Management

- NewAccount.java
- DeleteCustomer.java
- ViewCustomer.java

Transaction Management

- DepositMoney.java
- WithdrawMoney.java

Search and Help Modules

- FindAccount.java
- BankHelp.java

Database Integration

- Ensure proper schema design, CRUD operations, and data consistency.

Test Classes

- NewAccountTest.java: Unit tests for account creation logic.
- DeleteCustomerTest.java: Unit tests for customer deletion functionality.
- TransactionTest.java: Tests for transaction processing (deposit/withdraw).
- FindAccountTest.java: Unit tests for account search functionality.
- DatabaseTest.java: Tests to ensure database connections and CRUD operations function correctly.

Software Risk Issues

Risk Identification

Identifying potential risks associated with the banking system helps in preparing

mitigation strategies. The following software risk issues have been identified for the Core Java Banking System:

Database Connection Failure

- ***Risk:*** The system may face issues when connecting to the database due to incorrect configurations or connection timeouts.
- ***Impact:*** This would prevent access to critical customer and transaction data, halting core system functionalities.

Mitigation Strategy:

- Ensure that database configurations are tested in various environments before deployment.
- Set up automatic fallbacks or retries in case of database connection failures.

Complexity of Transaction Logic

- ***Risk:*** The transaction management logic for deposits, withdrawals, and balance updates may be error-prone, especially with concurrent transactions.
- ***Impact:*** Inaccurate balances or incorrect transaction records could cause significant issues, leading to potential financial discrepancies.

Mitigation Strategy:

- Use transaction isolation techniques to handle concurrent transactions.
- Implement extensive unit and integration tests specifically for transaction handling.

User Authentication and Authorization Failures

- ***Risk:*** Inadequate handling of user login and access control could lead to unauthorized access or account breaches.
- ***Impact:*** This could compromise customer data and lead to security vulnerabilities.

Mitigation Strategy:

- Ensure that authentication mechanisms are rigorously tested using security best practices (e.g., hashing passwords, token-based authentication).
- Regularly conduct penetration testing and security audits.

Data Integrity and Consistency Issues

- ***Risk:*** If data is not synchronized properly between the application and the database, it could lead to inconsistencies, especially after transactions.
- ***Impact:*** Data corruption or loss could occur, causing severe trust issues with the banking system.

Mitigation Strategy:

- Implement robust validation checks during database interactions.
- Use database transactions to ensure atomicity of critical operations like deposits and withdrawals.

Performance Under Load

- ***Risk:*** The system may experience performance degradation when processing a high volume of transactions or concurrent users.
- ***Impact:*** Slow response times or downtime could affect the user experience, particularly during peak times.

Mitigation Strategy:

- Conduct stress and load testing early in the development process to identify bottlenecks.
- Optimize database queries and use caching mechanisms for frequently accessed data.

Inadequate Error Handling

- ***Risk:*** Poor error handling could lead to unhandled exceptions, crashes, or incorrect system responses in case of invalid inputs or failures.
- ***Impact:*** A lack of graceful error recovery could result in poor user experience or application crashes.

Mitigation Strategy:

- Implement detailed exception handling and logging mechanisms.
- Ensure that errors are presented to the user in a user-friendly manner and are logged for troubleshooting.

Misunderstanding of Requirements

- ***Risk:*** Ambiguities or changes in the functional or non-functional requirements could lead to incomplete or incorrect implementation.
- ***Impact:*** This could result in features not meeting user expectations or missing critical functionalities.

Mitigation Strategy:

- Conduct regular meetings with stakeholders to ensure requirements are well understood.
- Use clear and comprehensive documentation to track requirements and changes.

Features to Be Tested

Functional Features

User Account Management

- Creation, deletion, and viewing of customer accounts.
- Validation of input for account details.

Transactions

- Deposit and withdrawal processes.
- Accurate balance updates post-transaction.

Search and Help

- Search accounts by ID or name.
- Provide user-friendly help documentation.

Database Integration

- Validate connection and ensure data integrity during CRUD operations.

Non-Functional Features

1. *Performance:* System should handle 100 concurrent transactions without failures.
2. *Error Handling:* The system should handle invalid inputs gracefully and display appropriate error messages.
3. *Scalability:* Ensure the system supports future enhancements with minimal changes.

Features Not to Be Tested

- Aesthetic functionalities of themes.
- Splash screen animations or transitions.
- Modules/files not directly used in the application (e.g., placeholder classes).

Test Approach

Levels of Testing

Unit Testing

- Validate individual methods and functionalities using JUnit.
- Examples: Test account creation in NewAccount.java.

Integration Testing

- Assess interactions between modules (e.g., deposits updating balances in the database).

System Testing

- End-to-end testing of all workflows, simulating real-world banking scenarios.

Database Testing

- Validate schema, data consistency, and performance of database queries.

Performance Testing

- Assess system behavior under load using tools like Apache JMeter.

Regression Testing

- Revalidate existing functionalities after updates or bug fixes.

Testing Techniques

Black-box Testing: Validate system behavior without knowledge of internal implementation.

White-box Testing: Examine the internal logic and database interactions.

Boundary Testing: Test inputs at the edge of acceptable ranges (e.g., maximum withdrawal limits).

Pass/Fail Criteria

Pass Criteria

- Actual outcomes match expected results.
- All critical functionalities achieve 100% success during testing.

Fail Criteria

- Actual outcomes deviate from expected results.
- Critical bugs are discovered during testing.

Suspension Criteria and Resumption Requirements

Suspension Criteria

Testing will be suspended if the following conditions are met:

1. ***Critical Defects:*** If a critical defect is found that prevents further testing or compromises the system's core functionality (e.g., database connection failure, authentication failure), testing will be paused until the issue is resolved.
2. ***Lack of Test Data:*** If necessary test data or resources are unavailable or corrupted, testing will be suspended until the data is restored or generated.
3. ***System Unavailability:*** If the test environment becomes unavailable (e.g., server crashes, network issues), testing will be paused until the environment is restored.

Resumption Requirements

Testing will resume once the following requirements are met:

1. ***Critical Defects Resolved:*** Once critical defects are fixed, testing will continue from the point where it was suspended.
2. ***Test Data Availability:*** Testing will resume once the missing or corrupted data has been made available.
3. ***System Restored:*** After the system is restored and the environment is operational, testing will resume with the previously tested cases to ensure no new issues have arisen.

Testing Deliverables

Test Case Document:

Comprehensive list of test cases, including inputs, expected outcomes, and actual results.

Bug Report:

Detailed records of identified defects, including severity and resolution status.

Test Execution Report:

Pass/fail status of all executed test cases.

Test Summary Report:

Overview of testing activities, results, and recommendations.

Remaining Test Tasks

The following tasks are yet to be completed and will be handled in subsequent phases:

Task	Assigned To	Status
Finalize Test Cases for All Modules	Test Engineers	Pending

Database Testing and Validation	Database Tester	Pending
Performance Testing Setup (JMeter)	Performance Tester	Pending
Regression Test Automation	Test Engineers	Pending
Error Handling and Exception Testing	Test Engineers	Pending
Test Execution and Bug Reporting	Test Engineers	Pending
Final Test Summary Report	Test Lead	Pending
User Acceptance Testing (UAT)	QA Analyst	Pending

Dependencies:

- Availability of test data.
- Resolution of any critical bugs or defects.
- Completion of earlier test phases (Unit, Integration, System Testing).

Testing Environment

Hardware Requirements

- Minimum: 4 GB RAM, 500 GB HDD.
- Recommended: 8 GB RAM, SSD storage.

Software Requirements

- JDK 11 or higher.
- IDE: IntelliJ IDEA or Eclipse.
- Database: SQLite or equivalent.
- Version Control: Git.

Data Requirements

- Sample data for accounts, transactions, and users.
- Test scenarios for edge cases and error conditions.

STAFFING AND TRAINING NEEDS

Staffing Needs

1. ***Test Lead***: One individual responsible for overseeing the testing process, coordinating with the development team, and ensuring the testing milestones are met.
2. ***Test Engineers:*** A team of engineers responsible for writing, executing, and maintaining test cases. Each engineer should be well-versed in Java programming and testing frameworks like JUnit.
3. ***Database Administrator:*** One DBA to support testing related to database operations, ensuring schema integrity and efficient queries during testing.
4. ***Performance Testing Specialist:*** A dedicated tester with expertise in performance testing tools like Apache JMeter to simulate load testing for the banking system.
5. ***QA Analysts:*** Two QA analysts who will help ensure quality control, verify that testing standards are met, and ensure the system meets non-functional requirements.

Training Needs

Test Engineers:

- Training on JUnit and other Java testing frameworks for unit and integration testing.
- Familiarization with the banking system's architecture and business logic.

Performance Testing Specialist:

- Training on using Apache JMeter or equivalent tools for stress testing.

Database Administrator:

- In-depth understanding of the system's database architecture, required for database testing and validation.

QA Analysts:

- Exposure to the banking system's user interface and core functionality.
- Training in bug tracking systems (e.g., Bugzilla) and defect reporting.

Roles and Responsibilities

Role	Responsibility
Test Lead	Oversee all testing activities, allocate resources, and track progress.
Test Engineers	Design and execute test cases; report defects.
Developers	Address reported bugs and clarify functionality when needed.
QA Analysts	Ensure the quality of deliverables and adherence to standards.

Schedule

Activity	Start Date	End Date	Duration
Test Plan Preparation	2024-12-09	2024-12-15	6 days
Test Case Development	2024-12-16	2024-12-22	6 days
Unit Testing Execution	2024-12-23	2024-12-27	4 days
Integration Testing Execution	2024-12-28	2024-12-31	3 days
System Testing Execution	2025-01-02	2025-01-07	5 days
Bug Fixing & Verification	2025-01-08	2025-01-12	4 days

Risks and Contingencies

Risk	Likelihood	Mitigation Strategy
Database connection failure	High	Validate configurations and use fallback options.
Unclear requirements for features	Moderate	Frequent stakeholder communication to clarify requirements.
Missing or incomplete test data	Low	Develop comprehensive test data sets during test case creation.
Resource unavailability	Moderate	Assign backup resources to critical tasks.

Approval

Name	Role	Signature	Date
Abdul Haq Zulfiqar	Test Lead	[Signature]	2024-12-11
Shahmeer	QA Analyst	[Signature]	2024-12-11
Project Manager	Project Manager	[Signature]	2024-12-11

Glossary

- **Test Plan**: A document that describes the scope, approach, resources, and schedule for testing activities.
- **Unit Testing**: Testing individual components or classes in isolation to ensure they work as expected.
- **Integration Testing**: Testing interactions between different modules to ensure they function together as expected.
- **System Testing**: End-to-end testing of the system as a whole to ensure all features and functionalities work together.
- **Regression Testing**: Testing the system after changes or bug fixes to ensure that existing functionality is not broken.
- **Black-box Testing**: Testing the system based on its functionality and expected behavior without knowledge of the internal code.

- **White-box Testing**: Testing the internal logic and structure of the system's code.
- **CRUD Operations**: Create, Read, Update, Delete operations that are fundamental to most database systems.
- **Pass/Fail Criteria**: The conditions under which a test is considered successful or unsuccessful.
- **Database Testing**: Verifying that the database operations such as schema design, CRUD operations, and data consistency are working correctly.
- **Performance Testing**: Testing the system to ensure it can handle the expected load and performance requirements.
- **Error Handling**: Testing how the system responds to errors, invalid inputs, and edge cases.