Test Plan

Invoice Management System

21st January 2025

INTRODUCTIONS

The purpose of this test plan is to define the testing strategy, approach, resources, and schedule of testing activities for the Invoice Management System (IMS). This document will serve as a guide to ensure the quality and functionality of the system by identifying the testing scope, objectives, and responsibilities.

Objectives:

- Validate that the IMS meets the specified requirements.
- Identify and rectify defects in the system.
- Ensure the system operates reliably across different environments and scenarios.
- Provide a robust, secure, and user-friendly experience for end-users.

Scope:

In-Scope: The testing will cover the following features and functionalities of the Invoice Management System:

- **User Authentication:** Testing the login, logout, and role-based access control.
- **Invoice Creation:** Verification of customizable templates and itemized details.
- **Client Management:** Maintenance and accessibility of client databases with contact and billing information.
- **Invoice Tracking:** Monitoring invoice statuses and payment due dates.
- **Payment Processing:** Recording payments and tracking outstanding balances.
- **Reporting:** Generating various reports related to invoicing activities and client performance.
- **Integration:** Testing the integration with accounting software for financial management.
- **Customization:** Testing the customization of invoices with company branding and tax information.
- **Notifications:** Verifying notifications for payment due dates and overdue invoices.
- **Multi-Currency and Language Support:** Ensuring functionality in multiple currencies and languages.

- **Security Measures:** Testing data encryption, secure payment processing, and role-based access control.

Out-of-Scope:

- Third-party accounting software functionality beyond integration.
- Performance under extreme load conditions.
- End-user training and user documentation.

TEST STRATEGY

Testing Levels:

- **Unit Testing:** Conducted by developers to test individual components or modules of the system.
- **Integration Testing:** To ensure different modules or services interact correctly, including integration with accounting software.
- **System Testing:** Full end-to-end testing of the entire system to verify that it meets the specified requirements.
- **User Acceptance Testing (UAT):** Final testing based on user requirements and business processes to ensure the system is ready for deployment.

Testing Types:

- **Functional Testing:** To validate the system against functional specifications (e.g., user authentication, invoice creation, client management).
- **Usability Testing:** To assess the system's user interface and ease of use.
- **Security Testing:** To ensure data protection, secure payment processing, and role-based access control.
- **Performance Testing:** To evaluate the system's responsiveness, stability, and scalability under various conditions.
- **Compatibility Testing:** To ensure the system functions across different browsers and devices.
- **Regression Testing:** To ensure that new code changes do not adversely affect the existing functionality.

Testing Tools:

- Functional testing tools like Selenium or Playwright for automation.
- Security testing tools like OWASP ZAP or Burp Suite.
- Performance testing tools like JMeter or LoadRunner.
- Issue tracking tools like JIRA for defect management.

Entry and Exit Criteria:

- **Entry Criteria:**
    - Requirements and design documents are finalized.
    - Development of the system is complete, and the build is stable.
    - Test environment is set up and accessible.
- **Exit Criteria:**
    - All planned test cases are executed.
    - All critical and major defects are resolved.
    - UAT is signed off by stakeholders.

TEST CASES

Please find the Test cases Excel sheet by clicking here.

1. **Verify login with valid credentials:**
   Ensure that users can log in successfully with correct username and password.

2. **Verify login with invalid credentials:**
   Validate that an error message is shown when logging in with incorrect credentials.

3. **Verify access control for different roles:**
   Confirm that users can only access the functionalities appropriate for their assigned roles.

4. **Verify creating an invoice with all mandatory fields filled:**
   Ensure that an invoice is created successfully when all required fields are completed.

5. **Verify invoice template customization:**
   Validate that customized templates are correctly applied to new invoices.

6. **Verify adding a new client:**
   Check that a new client can be added to the client list with correct details.

7. **Verify tracking the status of an invoice:**
   Ensure the invoice status updates correctly throughout its lifecycle.

8. **Verify recording a payment for an invoice:**
   Confirm that payments are recorded, and the invoice status is updated accordingly.

9. **Verify generating an invoice activity report:**
   Validate that the report displays accurate invoice activity data.

10. **Verify integration with accounting software:**
    Ensure transactions are synchronized correctly with external accounting software.

TEST ENVIRONMENT

Hardware Requirements:

- **Client Machine:**
  - Processor: Intel Core i5 or equivalent
  - RAM: 8 GB or more
  - Storage: 500 GB or more
  - Operating Systems: Windows 10, macOS 11, Linux (Ubuntu 20.04)
- **Server Machine:**
  - Processor: Intel Xeon or equivalent
  - RAM: 16 GB or more
  - Storage: 1 TB or more
  - Operating Systems: Windows Server 2019, Linux (CentOS 8)

Software Requirements:

- **Browsers:**
  - Google Chrome (latest version)
  - Mozilla Firefox (latest version)
  - Microsoft Edge (latest version)
  - Safari (for macOS)
- **Databases:**
  - MySQL 8.0 or higher
  - PostgreSQL 12 or higher

- **Web Server:**
    - Apache 2.4 or higher
    - Nginx 1.18 or higher
- **Other Tools:**
    - JIRA for issue tracking
    - Selenium WebDriver for automation testing
    - JMeter for performance testing
    - OWASP ZAP for security testing

Network Configuration:

- Stable internet connection with at least 10 Mbps speed for client machines.
- Secure network setup for server machines, including firewalls and VPN.

TEST EXECUTIONS

Test Execution Approach:

- Test cases will be executed in a phased manner, starting with unit testing, followed by integration, system, and user acceptance testing.
- Prioritize test cases based on their criticality and impact on the system.
- Execute automated tests using tools like Selenium for regression and functional testing.
- Manual testing will be conducted for usability, security, and exploratory testing.

Test Execution Schedule:

- **Unit Testing:** Week 1-2
- **Integration Testing:** Week 3
- **System Testing:** Week 4-5
- **User Acceptance Testing:** Week 6

Defect Reporting and Tracking:

- All identified defects will be reported and tracked in JIRA.
- Defects will be categorized based on severity and priority.
- Each defect report will include a description, steps to reproduce, expected vs. actual results, and screenshots if applicable.

Metrics for Test Execution:

- **Test Case Execution Status:** Number of test cases passed, failed, or blocked.
- **Defect Density:** Number of defects per module.
- **Defect Resolution Time:** Average time taken to resolve defects.
- **Test Coverage:** Percentage of requirements covered by test cases.

DEFECT MANAGEMENT

Defect Lifecycle:

1. **New:** Defect is identified and logged in the defect tracking tool.
2. **Assigned:** Defect is assigned to the relevant developer or team for investigation.
3. **In Progress:** Developer is working on fixing the defect.
4. **Fixed:** Developer resolves the defect and marks it as fixed.
5. **Retesting:** Tester verifies the fix by retesting the defect.
6. **Closed:** If the defect is resolved successfully, it is closed. If not, it is reopened for further investigation.

Defect Severity Levels:

- **Critical:** Defects that cause system crashes or prevent major functionality from working.
- **High:** Defects that impact significant functionality but have a workaround.
- **Medium:** Defects that affect less critical functionality or occur under specific conditions.
- **Low:** Minor defects that do not significantly impact functionality.

Defect Priority Levels:

- **P1 (Highest):** Must be fixed immediately.
- **P2:** Should be fixed in the next release.
- **P3:** Fix can be scheduled for future releases.
- **P4 (Lowest):** Minor issues that may be fixed if time permits.

Defect Reporting Template:

- **Defect ID:** Unique identifier for the defect.
- **Summary:** Brief description of the defect.
- **Steps to Reproduce:** Detailed steps to replicate the defect.
- **Expected Result:** What should happen.
- **Actual Result:** What actually happens.
- **Severity:** Level of severity assigned to the defect.

- **Priority:** Level of priority assigned to the defect.
- **Status:** Current status of the defect (New, Assigned, In Progress, Fixed, Retesting, Closed).
- **Environment:** Details of the environment where the defect was found.
- **Attachments:** Screenshots, logs, or any relevant files.

RISKS AND ASSUMPTIONS

Risks:

1. **Incomplete Requirements:** There is a risk that requirements may not be completely defined, leading to missed test cases or functionality.
    - *Mitigation:* Regular communication with stakeholders and requirement reviews.
2. **Integration Issues:** Integration with external accounting software may not work as expected.
    - *Mitigation:* Early integration testing and collaboration with external system teams.
3. **Security Vulnerabilities:** Potential vulnerabilities in the system could compromise sensitive data.
    - *Mitigation:* Conduct thorough security testing and regular audits.
4. **Performance Bottlenecks:** The system may not perform well under high load conditions.
    - *Mitigation:* Perform extensive performance testing and optimize the system for scalability.
5. **Resource Availability:** Limited availability of testing resources could delay the testing process.
    - *Mitigation:* Proper planning and resource allocation, and potential use of automation to cover more ground.

Assumptions:

1. The functional requirements provided are complete and final.
2. The development team will provide stable builds for testing.
3. The test environment will be set up and configured before the start of testing.
4. All necessary testing tools and software will be available and functioning properly.
5. Test data will be provided or will be created in alignment with the test scenarios.