****

**MEKELLE UNIVERSITY**

**EITM**

**SCHOOL OF COMPUTING**

**DEPARTMENT OF SOFTWARE ENGINEERING**

**INDIVIDUAL ASSIGNMENT ON TEST PLAN FOR STUDENT REGISTRATION SYSTEM**

**ABDELAZIZ YASSIN**

**CHS/UR175850/12**

**SUBMITTED TO: INST. MESSELE**

**Test Plan for Student Registration System**

**Table of Contents**

1. Testing Planning and Control 1.1 Test Planning
   * Objective
   * Components of a Test Plan
   * Tools for Test Planning 1.2 Test Control
   * Objectives
   * Activities
   * Tools for Test Control
   * Deliverables
2. Testing Analysis and Design 2.1 Test Analysis
   * Objectives
   * Activities 2.2 Test Design
   * Objectives
   * Activities
   * Deliverables
   * Tools for Analysis and Design

**1. Testing Planning and Control**

**1.1 Test Planning**

Test planning defines the scope, approach, resources, and schedule for testing activities, serving as the blueprint for ensuring quality and reliability.

**1.1.1 Objective**

* Validate that the Student Registration System meets functional and non-functional requirements.
* Detect and resolve defects to ensure a smooth user experience.
* Ensure system compliance with performance, scalability, and security standards.

**1.1.2 Components of a Test Plan**

1. **Introduction**
   * Overview: Testing the functionalities and performance of the Student Registration System.
   * Stakeholders: Project team, QA team, and end users.
2. **Scope of Testing**
   * **Features to be tested**: User registration, login, course management, enrollment, and reporting.
   * **Features not to be tested**: Integration with third-party services beyond APIs provided.
3. **Test Objectives**
   * Ensure all requirements are met with complete coverage.
   * Validate system stability under varying loads.
4. **Test Strategy**
   * Levels: Unit, integration, system, and acceptance testing.
   * Types: Manual and automated testing, regression, performance, and security testing.
   * Environments: Replication of production setup.
5. **Test Deliverables**
   * Test cases, test data, defect logs, and test summary reports.
6. **Entry and Exit Criteria**
   * **Entry Criteria**: Test cases and test environments are ready.
   * **Exit Criteria**: All critical defects are resolved, and test objectives are met.
7. **Resources**
   * Team: Testers, developers, and product owners.
   * Tools: PHPUnit
8. **Schedule**
   * Define timelines for test preparation, execution, and reporting.
9. **Risk Management**
   * Risks: Delayed test case preparation, environment issues.
   * Mitigation: Early preparation and continuous monitoring.
10. **Approval**
    * Approved by QA lead and project manager.(In our case ,our peer testers)

**1.1.3 Tools for Test Planning**

* Documentation: Jira, Confluence.
* Test Management: TestRail, TestLink.

**1.2 Test Control**

Monitoring and managing test activities to ensure alignment with the test plan.

**1.2.1 Objectives**

* Ensure testing stays on track.
* Adapt to changes in requirements or schedule.
* Provide visibility to stakeholders.

**1.2.2 Activities**

1. **Monitoring**
   * Track test execution progress and defect metrics.
   * Check adherence to the test schedule.
2. **Reporting**
   * Daily and weekly reports with test coverage and defect trends.
3. **Issue Management**
   * Identify bottlenecks and resolve conflicts.
4. **Change Management**
   * Adjust test plans for requirement changes.
5. **Defect Tracking**
   * Prioritize and ensure timely defect resolution.

**1.2.3 Tools for Test Control**

* Test Management: TestRail.
* Defect Tracking: Jira.
* Reporting Dashboards: Power BI.

**1.2.4 Deliverables**

* Test Plan Document.
* Test Metrics Reports.
* Final Test Report.

**2. Testing Analysis and Design**

**2.1 Test Analysis**

**2.1.1 Objectives**

* Identify test conditions and ensure complete coverage.
* Establish traceability between requirements and test cases.

**2.1.2 Activities**

1. **Requirement Analysis**
   * Review functional and non-functional requirements.
   * Identify ambiguities or missing details.
2. **Test Basis Identification**
   * Artifacts: SRS, use cases, architecture documents.
3. **Derive Test Conditions**
   * Map conditions to requirements.
4. **Prioritization**
   * Rank conditions based on risk and importance.
5. **Entry Criteria**
   * Approved requirements.

**2.2 Test Design**

**2.2.1 Objectives**

* Develop test cases and data to validate test conditions.

**2.2.2 Activities**

1. **Test Case Design**
   * Write detailed test cases using a structured template.

**Test Case: Student Registration System**

* + **Test Case ID:**
  + SRS-001
  + **Test Case Name:**
  + Student Registration and Payment Process
  + **Precondition:**
  + The student is logged into the system.
  + A registration fee is configured in the system.
  + Payment gateway (e.g., Stripe) is integrated.
  + **Test Steps:**
  + Navigate to the "Student Registration" page.
  + Verify that the registration fee is displayed correctly.
  + Click on the "Proceed to Payment" button.
  + Enter valid payment details (e.g., credit card number, expiry date, CVV).
  + Submit the payment.
  + Wait for the system to process the payment.
  + Verify that a confirmation message is displayed upon successful payment.
  + Check the student dashboard to ensure the registration status is updated.
  + Verify that the system generates a receipt and sends it to the student's email.
  + **Expected Results:**
  + The registration fee is displayed correctly.
  + Payment is processed successfully.
  + A success message is shown on the screen.
  + The registration status is updated in the system.
  + A receipt is generated and sent to the student's email.
  + **Negative Test Cases:**
  + Attempt payment with invalid card details:
  + Expected Result: The system displays an error message, and the payment is not processed.
  + Try to register without paying the fee:
  + Expected Result: The system prevents registration and displays a warning message.
  + **Postcondition:**
  + Student registration is completed and recorded in the system database.
  + Payment details are stored securely.
  + A receipt is available for the student.

1. **Test Data Design**
   * Define valid, invalid, and boundary data inputs.  
     **2. Test Data Design**

Valid Data Inputs:

* Student Information:
* Ethiopian names (e.g., "Abebe Kebede", "Mensura Abdu ").
* Names with titles (e.g., "Dr. Yassin Ibrahim ", "Prof. Gebre Yohannes").
* Names in Amharic (e.g., "አብድቡይ አላምዋ", "ሃንሪን ተቅሪና").
* Names in Tigrigna (e.g. “ሓጎስ” , ”በርሀ”).
* Names in Ge’ez (eg. "ጥአር ሐርቲር", "ኢንዳው አዳላጋ")
* Correct email formats (e.g., "abebe.kebede@example.com").
* Valid phone numbers (e.g., "+251911234567").
* Payment amounts matching the course fee.

**Invalid Inputs:**

* Names with no last name (e.g., "Abebe").
* Names with special characters (e.g., "@bebe #Kebede").
* Names in unsupported scripts (e.g., Cyrillic, "Абебе").
* Excessively long names (e.g., "Abebe" repeated 50 times).
* Invalid email formats (e.g., "abebe@.com", "@example.com").  
  ampersand (&), equals sign (=), underscore (\_), apostrophe ('), dash (-), plus sign (+), comma (,), brackets (<,>), and more than one period (.).
* Invalid phone numbers (e.g., "12345", "+999999999999999").
* Payment amounts exceeding or below the course fee.

**Boundary Inputs:**

* Names with exactly 255 characters.
* Emails with the maximum allowed length.(320 characters )
* Payment amounts at the upper and lower boundaries of allowed fees.
* Phone numbers with exactly 15 digits (maximum length).

**Edge Case Inputs:**

* Names with diacritics (e.g., "Méşfiné Gébré Tésfa").
* Names with hyphens or apostrophes (e.g., "Selam-Omer", "Lili'o").
* Emails with uncommon domain extensions (e.g., "abebe@university.ac.ke").
* Inputs containing leading or trailing whitespace

1. **Test Environment Setup**
   * Prepare environments mirroring production.

**3. Test Environment Setup**

* Prepare a staging environment mirroring production:
  + Clone the production database structure.
  + Ensure test data is isolated and anonymized.
  + Configure Stripe sandbox API keys for payment testing.
  + Enable debugging logs for tracing issues.
  + Set up email testing tools (e.g., Mailtrap) to verify email notifications.
* Ensure proper permissions and roles are configured for test accounts

1. **Test Automation Design**
   * Identify cases suitable for automation.
   * **Form Validation:**
   * Ensure all required fields are validated.
   * Verify valid and invalid inputs.
   * **Payment Processing:**
   * Simulate successful and failed payment scenarios.
   * Test network interruptions during transactions.
   * **Course Enrollment:**
   * Verify that the student is enrolled in the selected course upon payment.
   * **Email Notifications:**
   * Check that confirmation emails are sent with correct details.
   * **Receipt Generation:**
   * Verify receipt details and PDF download functionality.
   * **Test Automation Tools:**
   * Selenium for UI testing.
   * PHPUnit for backend logic.
   * Postman/Newman for API testing.

1. **Traceability Matrix**
   * Map test cases to requirements.

**2.2.3 Deliverables**

* Test Cases.
* Test Data.
* Requirements Traceability Matrix (RTM).

**2.2.4 Tools for Analysis and Design**

* Test Management: TestRail.
* Automation: Selenium, PHPUnit.
* Traceability Matrix: Excel.