MEKELLE UNVIERSITY

EITM

SCHOOL OF COMPUTING

DEPARTMENT OF SOFTWARE ENGINEERING

TEST PLAN OF STUDENT REGISTRATION SYSTEM

- Name: Semere Herruy
- ID: ugr/177912/12

Submission date: 08/05/2017 E.c

Submitted to : Inst. Mesele

TEST PLAN

1.1 TEST PLANNING

1.1.1 OBJECTIVE:

The objective of the test planning phase is to define the testing strategies, processes, and resources required to validate the functionality and performance of the **Student Registration System**. This phase ensures that all functional and non-functional requirements specified in the SRS document are adequately tested and meet quality standards before deployment.

1.1.2 COMPONENTS OF A TEST PLAN:

1. **Scope of Testing:**
    - Validate all functional requirements, including course registration, payment integration, and session management.
    - Verify non-functional requirements like scalability, reliability, and security.
2. **Test Objectives:**
    - Ensure that students can register for courses successfully.
    - Confirm secure and accurate payment processing.
    - Verify session management reliability during high traffic.
3. **Test Environment:**
    - Use a staging environment replicating the production setup.
    - Technologies: Laravel, MySQL, and API Gateway with necessary middleware and microservices.
4. **Test Deliverables:**
    - Test cases and scripts for functional, performance, and security testing.
    - Test execution reports and logs.
5. **Roles and Responsibilities:**
    - **Test Manager:** Oversee planning, execution, and reporting.
    - **Testers:** Create and execute test cases.
    - **Developers:** Support defect resolution.

1.1.3 TOOLS FOR TEST PLANNING:

- **Test Management:** JIRA or TestRail for managing test cases and tracking progress.
- **Automation Tools:** Selenium for functional testing and JMeter for load testing.
- **Version Control:** GitHub for managing test scripts and code.

1.2 TEST CONTROL

1.2.1 OBJECTIVES:

The objective of test control is to monitor and manage testing activities to ensure they align with the test plan. Adjustments are made to address deviations and ensure the system meets the defined requirements and quality benchmarks.

1.2.2 ACTIVITIES:

1. **Progress Monitoring:**
    - Track test case execution status.
    - Monitor defect resolution rates and timelines.
2. **Risk Management:**
    - Identify potential risks like delayed defect fixes or system instability.
    - Develop contingency plans for critical issues.
3. **Change Management:**
    - Manage updates to test cases due to changes in requirements.
4. **Communication:**
    - Regular updates to stakeholders through status reports.

1.2.3 TOOLS FOR TEST CONTROL:

- **Progress Tracking:** JIRA dashboards or TestRail analytics.
- **Communication:** Slack or Microsoft Teams for team collaboration.
- **Defect Tracking:** JIRA for logging and managing bugs.

1.2.4 DELIVERABLES:

1. **Testing Metrics:**
    - Pass/Fail rates for test cases.
    - Defect density and resolution times.
    - Performance benchmarks.
2. **Test Reports:**
    - Detailed reports summarizing executed tests, results, and identified issues.
3. **Updated Test Cases:**
    - Revised test scripts based on defect fixes or requirement changes.
4. **Final Test Summary:**
    - Comprehensive documentation of testing outcomes to inform deployment readiness.

2.1 TEST ANALYSIS

2.1.1 **OBJECTIVES:**

The objectives of the test analysis phase are to:

1. Identify testable requirements from the SRS document.
2. Define the scope and boundaries of testing based on system components.
3. Establish test conditions and criteria for success.
4. Ensure all functional and non-functional requirements are covered in the test scenarios.

2.1.2 **ACTIVITIES:**

1. **Requirement Analysis:**

- Analyze functional requirements, such as course registration, payment integration, and session management.
- Assess non-functional requirements like performance, scalability, and security.

2. **Risk Assessment:**
   - Identify high-risk areas, such as payment security and session reliability.
   - Prioritize testing for critical components, including API Gateway and Payment Service.
3. **Defining Test Objectives:**

   - Validate user scenarios like:

     - Student login/future consideration/ and account creation
     - Registration process (validating user input such as name, email, phone number)
     - Course registration and payment process
     - Chapa gateway interaction and successful payment.

   - Test session handling during high-traffic scenarios to avoid system downtime.

4. **Mapping Requirements to Test Cases:**
   - Create traceability between system requirements and corresponding test cases.

2.2 TEST DESIGN

2.2.1 **OBJECTIVES:**

The objectives of the test design phase are to:

1. Develop detailed test cases and scripts based on identified test conditions.
2. Design test data to cover normal, edge, and error scenarios.
3. Prepare the testing environment and establish necessary configurations.

2.2.2 **ACTIVITIES:**

1. **Test Case Development:**

   - Create test cases for each system feature, ensuring comprehensive coverage:

     - **Student Registration**: Testing the input fields for name, email, and phone number. Include validation for format, non-emptiness, and boundary cases.
     - **Course Registration**: Validate course details (name, description, amount).
     - **Payment Processing**: Test valid and invalid payment scenarios, including interactions with the Chapa gateway.

   - Include test cases for edge cases (e.g., invalid inputs like a missing email, phone number format issues, and payment failures).

◆ Detailed Test Cases:

Test Case 1: Student Registration - Name Field Validation

- **Test Case ID**: TC-REG -001
- **Test Case Description**: Verify that the name field accepts valid input and rejects invalid input (e.g., numbers or special characters).
- **Test Steps**:
    1. Navigate to the registration page.
    2. Enter a valid name (e.g., "Abel Fisha").
    3. Enter an invalid name (e.g., "Abel123").
    4. Submit the registration form.
- **Expected Results**:
    - The form should accept valid names.
    - The form should reject names with numbers or special characters and display an error message.
- **Preconditions**: User must be on the registration page.
- **Postconditions**: The user should receive appropriate feedback for name validation.
- **Priority**: Critical

Test Case 2: Student Registration - Email Format Validation

- **Test Case ID**: TC-REG-002
- **Test Case Description**: Verify that the email field accepts valid email format and rejects invalid email formats.
- **Test Steps**:
    1. Navigate to the registration page.
    2. Enter a valid email (e.g., "abelfisha@example.com").
    3. Enter an invalid email (e.g., " abelfisha @example").
    4. Submit the registration form.
- **Expected Results**:
    - The form should accept valid emails.
    - The form should reject invalid emails and display an error message.
- **Preconditions**: User must be on the registration page.
- **Postconditions**: User gets appropriate feedback about the email format.
- **Priority**: High

Test Case: Validate Age Input

- **Test Case ID**: TC-AGE-001
- **Description**: Ensure the age field accepts valid inputs and rejects invalid ones.
- **Preconditions**: The user must be on the student registration form.
- **Test Steps**:
    - Enter a valid age (e.g., 18, 25, 50).
    - Enter an invalid age (e.g., -5, 0, 200, "abc").
    - Leave the age field empty.
- **Expected Results**:
    - For valid age: The system accepts the input and proceeds without error.

    - For invalid age: The system displays an error message like *"Invalid age. Please enter a valid number between 1 and 120."*
    - For empty age: The system displays an error message like *"Age is a required field."*
- **Postconditions**: The system prevents the form submission for invalid or empty inputs.

Test Case: Validate Phone Number Input

- **Test Case ID**: TC-PHONE-002
- **Description**: Ensure the phone number field accepts valid inputs and rejects invalid ones.
- **Preconditions**: The user must be on the student registration form.
- **Test Steps**:
    - Enter a valid phone number (e.g., +251912345678, 0912345678).
    - Enter an invalid phone number (e.g., 12345, abcdefg, 09123!45678).
    - Leave the phone number field empty.
- **Expected Results**:
    - For valid phone numbers: The system accepts the input and proceeds without error.
    - For invalid phone numbers: The system displays an error message like *"Invalid phone number. Please enter a valid format."*
    - For empty phone numbers: The system displays an error message like *"Phone number is required."*
- **Postconditions**: The system prevents the form submission for invalid or empty inputs.

Test Case: Validate Course Selection

- **Test Case ID**: TC-COURSE-001
- **Description**: Ensure the course type selection displays the correct course description and price automatically.
- **Preconditions**: The user is logged in and on the course registration form.
- **Test Steps**:
    - Select a valid course type from the dropdown menu (e.g., "Web Development").
    - Select an invalid option (e.g., blank or non-existent course type, if allowed).
    - Change the course type multiple times to verify dynamic updates.
- **Expected Results**:
    - For valid selection: The description and price of the selected course are displayed automatically (e.g., "Web Development: $500").
    - For invalid selection: The system displays an error message like *"Invalid course selection."*
    - Dynamic updates occur seamlessly as the user changes the selection.
- **Postconditions**: The course type, description, and price fields are populated correctly for valid inputs.

Test Case: Validate Payment Confirmation Process

- **Test Case ID**: TC-PAYMENT-002
- **Description**: Ensure the payment process works as expected after course selection.
- **Preconditions**:
    - User has successfully selected a course type.

    - Payment gateway (e.g., Chapa) is functional.
- **Test Steps**:
    - Click on "Pay"
    - Provide valid payment details (e.g., card number, CVV, expiry date).
    - Provide invalid payment details (e.g., incorrect card number, expired card).
    - Leave payment details empty and attempt to submit.
- **Expected Results**:
    - For valid payment details: The payment is processed successfully, and a confirmation message like *"Payment successful. Thank you for registering!"* is displayed.
    - For invalid payment details: The system displays an error message like *"Invalid payment details. Please check your information and try again."*
    - For empty payment details: The system displays an error message like *"Payment details are required."*
- **Postconditions**:
    - Successful payment transitions the user to the confirmation page or dashboard.
    - Unsuccessful payment prevents registration and prompts the user to retry.

Test Case: Validate Course Registration Flow

- **Test Case ID**: TC-COURSE-003
- **Description**: Ensure the entire flow of course registration, from selection to payment confirmation, works as intended.
- **Preconditions**:
    - The user is logged in.
    - All course types, descriptions, and prices are predefined in the system.
- **Test Steps**:
    - Select a course type and verify that the description and price populate automatically.
    - Click "Proceed to Payment."
    - Enter valid payment details and confirm payment.
- **Expected Results**:
    - The system successfully registers the user for the selected course and provides a success message.
    - The user is redirected to the dashboard or confirmation page.
- **Postconditions**:
    - Registration data is stored in the database.
    - Payment is logged and confirmed in the system.

2. **Test Data Design:**

    - Develop datasets for valid, invalid, and boundary conditions:
        - **Valid Data**:
            - Valid student credentials (e.g., correct name, email, and phone number format)
            - Correct course details (e.g., course name, description, and amount)
            - Valid payment information (e.g., card number, ...)
        - **Invalid Data**:
            - Missing required fields (e.g., missing email or payment details)
            - Invalid email or phone format

        - Payment gateway failures
    - **Boundary Conditions**:
        - Maximum number of course registrations
        - Maximum session duration

3. **Test Environment Setup:**
    - Configure a **staging environment** to simulate production, including the **API Gateway**, **Payment Service**, **Session Store**, and the **Database**.
    - Integrate tools and libraries required for testing.
4. **Review and Validation:**
    - Peer-review test cases to ensure completeness and accuracy.
    - Validate test cases against requirements for traceability.

2.2.3 DELIVERABLES:

1. **Test Case Documentation**:
    - Detailed test case descriptions, including:
        - **Test Case ID**
        - **Test Case Description**
        - **Test Steps**
        - **Expected Results**
        - **Preconditions** (e.g., logged-in student)
        - **Postconditions**
        - **Priority** (Critical/High/Medium/Low)
2. **Test Data**:
    - Sample datasets for valid, invalid, and boundary scenarios.
3. **Test Scripts**:
    - Automated test scripts for functional and performance testing.
4. **Test Environment Report**:
    - Documentation of the testing environment setup and configurations.

2.2.4 TOOLS FOR ANALYSIS AND DESIGN:

1. **Test Case Management:**
    - JIRA or TestRail for organizing test cases and mapping them to requirements.
2. **Automation Tools:**
    - Selenium for functional testing.
    - JMeter for performance and load testing.
3. **Database Tools:**
    - MySQL Workbench for managing and preparing test data.
4. **Version Control:**
    - GitHub for storing and versioning test scripts and related files.