MEKELLE UNIVERSITY
COLLEGE OF ETHIOPIA INSTITUTION OF TECHNOLOGY- MEKELLE
SCHOOL OF COMPUTING

DEPARTMENT-SOFTWARE ENGINEERING(SE)
COURSE-NAME: Software Testing and Quality Assurance
COURSE-CODE: (SENG4551)
INDIVIDUAL ASSESEMENT: ON TEST-PLANNING
Name – Binyam Tesfay
ID –ugr/170210/12
Year- 4th

SUBMITTED TO-Instructor MESELE
SUBMISSION DATE-JAN 13/2025

Table of Contents

1.Testing Planning and Control

1.1 Test Planning

1.1.1 Objective

The goal my of test plan is to ensure the quality and reliability of the Student Registration System, define clear testing boundaries, and efficiently allocate resources..

1.1.2 Components of a Test Plan

1.1.2.1 Introduction

Designed to simplify student registration and provide a seamless user experience, this system focuses on testing its functionality, reliability, and usability while identifying defects throughout the process. Key stakeholders in this process include:
(i) testers tasked with identifying and reporting defects.
(ii) developers responsible for building the system, and

1.1.2.2 Scope of Testing

The primary focus of testing will be on the user registration process, including integration with Stripe for both successful and failed transactions, and the admin dashboard's ability to manage and sustain student data effectively. Additionally, rigorous evaluation will be conducted on user authentication and the security of sensitive data. Non-functional aspects such as response time and load handling are crucial to ensure the system is robust, reliable, and efficient. However, testing features beyond Stripe's response handling concerning third-party APIs is excluded, as these have already undergone independent testing.

1.1.2.3 Testing Objectives

The main objectives are to ensure the user registration process functions correctly and to validate that the payment processing systems are secure and reliable.

1.1.2.4 Test Strategy

Testing Levels:

- **Unit Testing:** Validate individual components such as form validations and API calls.
- **Integration Testing:** Ensure seamless interaction between Node views and the Stripe payment module.
- **System Testing:** Verify the end-to-end registration and payment workflow.
- **Acceptance Testing:** Evaluate compliance with requirements through user feedback.

Testing Types:

- **Manual Testing:** Focus on scenarios like critical path testing and creative test cases.

- **Automation Testing:** Leverage Selenium for UI workflow automation and Jest for backend code validation.
- **Load Testing:** Confirm the system's ability to handle multiple concurrent registrations and payment transactions.

Test Environment Requirements:
A staging environment (Env1) that mirrors production, including database testing and integration with payment APIs.

Testing Approaches:

1. **Black Box Testing:**
   - Test the system without knowledge of its internal implementation, focusing on inputs and expected outputs.
2. **White Box Testing:**
   - Examine the internal workings of the system, including code structure, logic, and data flows.
3. **Regression Testing:**
   - Re-run previously executed test cases to ensure that new changes do not adversely affect existing functionality.
4. **Exploratory Testing:**
   - Conduct ad-hoc, unscripted testing to identify edge cases or scenarios not covered by predefined test cases.
5. **Performance Testing:**
   - Measure system performance under varying loads to ensure it meets performance benchmarks.
6. **Security Testing:**
   - Evaluate system defenses against potential vulnerabilities, such as unauthorized data access or SQL injection attacks.
7. **End-to-End Testing:**
   - Simulate real-world workflows to verify the system's behavior from start to finish, covering all interconnected modules.

1.1.2.5 Test Deliverables

Test Cases and Scripts

Test Case ID	Scenario	Description	Preconditions	Steps	Expected Results	Priority
TC_REG_001	Successful Registration	Verify registration succeeds with valid inputs.	Registration form is displayed, all fields are visible.	1. Open registration form. 2. Enter valid first name, last name, email		High

Test Case ID	Scenario	Description	Preconditions	Steps	Expected Results	Priority
				and all other fields. 3. Click "Register".		
TC_REG_002	Invalid Email Validation	Verify error handling for invalid email inputs.	Registration form is displayed with email validation logic in place.	1. Open registration form. 2. Enter valid first name/last name but invalid email format. 3. Click "Register".	Error displayed: *"Please enter a valid email address."* Registration fails.	High
TC_REG_003	Missing Fields in Registration	Test form handling when mandatory fields are not provided.	Registration form is displayed.	1. Open registration form. 2. Leave one or more mandatory fields empty. 3. Click "Register".	System shows error highlighting empty fields. Error message: *"This field is required."*	Medium
TC_PAY_001	Successful Payment via Stripe	Verify Stripe processes payment successfully.	Student is logged in; Stripe sandbox environment is ready.	1. Login as student. 2. Fill valid registration details 3.Click ‘Register’ button 4. Enter valid payment details via Chapa. 4. Confirm payment.	Payment succeeds. Success message displayed. Receipt email sent.	High
TC_PAY_002	Payment Failure with Invalid Card	Ensure system handles payment failure	Student is logged in; Stripe sandbox is ready.	1. Login as student. 2.Fill registration details.	Payment fails. Error: *"Transaction declined.*	High

Test Case ID	Scenario	Description	Preconditions	Steps	Expected Results	Priority
		gracefully for invalid card details.		3. Click ‘register’button. 4. Enter invalid/expired card details in Stripe. 4. Confirm payment.	*Use a valid payment method."*	
TC_PAY_003	Payment Failure with Network Interruption	Test system response when network connection is lost during payment.	Student is logged in and proceeds to payment page; network can be interrupted.	1. Start payment process. 2. Disable network connection during transaction. 3. Observe behavior.	Payment fails gracefully. Message: *"Network error. Please try again."* No amount deducted.	Medium
TC_SEC_001	SQL Injection in Registration Form	Verify system security against SQL injection attacks.	Registration form is displayed.	1. Enter malicious SQL command (e.g., DROP TABLE users;) in input fields. 2. Submit the form.	Input sanitized. No harmful effect occurs. Error message: *"Invalid input."*	High
TC_SEC_002	HTTPS Payment Security	Validate payment transactions use HTTPS for secure communication.	Payment page is displayed; Browser displays URL and SSL certificate details.	1. Access payment page. 2. Check URL starts with https://. 3. Verify SSL certificate is valid.	Payment communication is secured via HTTPS.	High
TC_PERF_001	High Load During Registration	Verify system performance during 2 simultaneous user	Registration module is deployed to a staging server; load testing tool	1. Simulate 2 users accessing the registration page at the	System remains stable. Response times < 2s. No	High

Test Case ID	Scenario	Description	Preconditions	Steps	Expected Results	Priority
		registrations.	is ready.	same time. 2. Monitor response times and server behavior.	significant error rates or crashes.	
TC_PAY_004	Duplicate Payment Attempts	Ensure the system prevents duplicate payments for the same registration.	Student is logged in and has already successfully paid for the course.	1. Login as student. 2. Attempt to pay again for the same course.	Duplicate payment is blocked. Message: *"Payment for this course already made."*	Medium

Test Data
Types of Data: valid Email Ids invalid email ID, payment Entrance (valid and invalid situations)

Test Reports
Key Statistics: pass rates/Fail rates

Defect Logs
Categorized as: Severity levels (low,medium,high)

1.1.2.6 Entry and Exit Criteria

Entry Criteria

1. **Requirements Documentation Final —** All requirements documentation should be finalized, reviewed and signed off by relevant stakeholders.
2. **Development of all Modules Completed**: Proper development of all modules must be in 100% and they are ready for testing.
3. Setup Test Environment with Stripe Sandbox: A standalone test environment should be already created and must verify all Stripe integration totally functional.

Exit Criteria

1. **Fix all high and critical severity defects**: Any defect identified to be of high or critical severeness during testing must get fixed and verified as closed.

2. **Functional Coverage Test at Least 93% (and ideally more)** You need to ensure that the functional tests are at least testing 93% of the required requirements and functions.

1.1.2.7 Resources

Team Size: with one classmate.

Tools and Systems

Tools:

- Selenium: Functional & regression tool for automation.
- Jest: a Framework for both unit and API testing.
- Postman: Systems to check an API temporarily, validate

Systems: Desktops supplied, having access to live servers for running and monitoring tests expeditiously.

1.1.2.8 Schedule

Task	Start Date	End Date	Responsible
Test case creation	Jan 13, 2025	Jan 15, 2025	Tes
Environment setup	Jan 14, 2025	Jan 17, 2025	Tes
Execution of tests	Jan 19, 2025	Jan 23, 2025	Team mate
Defect logging/fixing	Jan 22, 2025	Jan 27, 2025	Tes
Final test report	Jan 28, 2025	Jan 31, 2025	Tes

1.1.2.9 Risk Management

Risks:
a) Stripe API Downtime: Outages may disrupt payment testing.
b) Loss of Transaction Data due to Network Disruption: Network instability could lead to compromised transaction data.

Mitigation Strategies:
A. **Implement API Fallbacks:**

- Develop and thoroughly test fallback APIs to ensure internal functionality remains unaffected during API downtimes. This could include queuing failed transactions for retry once the service is restored or using mock responses in test environments to ensure consistency.

B. **Continuous Backup of Sensitive User Data:**

- Perform regular backups to prevent data loss and ensure users can be quickly restored once their network connection is re-established. Implement real-time data synchronization to maintain consistency between the user data and backend systems.

C. **Graceful Error Handling and User Feedback:**

- Design the system to handle errors gracefully by providing clear, user-friendly error messages in case of payment failures or network disruptions. This can guide users to retry or check their connection without confusion.

D. **Implement Redundancy for Payment Processing:**

- Set up alternative payment gateways alongside Stripe to provide redundancy and ensure that the system can process payments even if Stripe experiences downtime.

E. **Monitor System Performance in Real-Time:**

- Utilize monitoring tools to continuously track the health of APIs and network performance. This allows for quick detection of issues, enabling preemptive mitigation before they impact the system or user experience.

1.1.2.10 Approval

Test plan and results to be reviewed and approved by instructor Mesele

1.1.3 Tools for Test Planning

Documentation Tools:

- **MicrosoftWord:** For writing and sharing the test plan collaboratively, with easy integration into the project management system and seamless updates from all team members.

Test Management Tools:

- **TestLink:** An open-source test management tool that allows for organizing and managing test cases, tracking execution progress, and reporting results effectively.

Version Control Tools:

- **GitHub:** A comprehensive platform for version control and CI/CD integration, ensuring test scripts, documentation, and configurations are properly versioned and tracked throughout the testing process.

Additional Tools and Ideas:

- **Continuous Integration Tools:**
  - **Jenkins or CircleCI:** Automate test execution through CI/CD pipelines to continuously validate the system with every code change and ensure consistent quality.
- **Collaboration Tools:**
  - **Slack or Microsoft Teams:** Enable real-time communication among the testing team for quick issue resolution, feedback sharing, and status updates.
- **Defect Tracking Tools:**
  - **Bugzilla:** A dedicated tool for defect tracking that helps in logging, prioritizing, and managing defects found during the testing phase.
- **Automation Tools:**
  - **Cypress or Puppeteer:** These tools can be used for end-to-end testing, automating complex workflows and providing faster execution times compared to traditional Selenium.

1.2 Test Control

1.2.1 Objectives

. 1. **Adherence to the Test Plan:** Ensure that testing processes align with the defined test plan, following the established scope, resources, and timelines.
2. **Flexibility and Adaptability:** Maintain flexibility within the testing schedule to accommodate any changes in project scope, user requirements, or unforeseen challenges. This includes adjusting test priorities and resources as needed.

3. **Stakeholder Transparency:** Keep stakeholders informed and engaged by providing regular updates on testing progress, key findings, and any risks or issues encountered during the process. Transparency in the form of reports, dashboards, and status meetings ensures that all parties remain aligned on objectives and expectations.
4. **Continuous Improvement:** Continuously evaluate testing processes and outcomes to identify areas for improvement. This could include refining test cases, enhancing automation coverage, or improving communication strategies based on feedback.
5. **Risk Mitigation:** Proactively identify potential risks or roadblocks early in the testing process and establish contingency plans. This ensures that any critical issues are addressed before they impact the project timeline or delivery.
6. **Efficient Resource Management:** Ensure optimal use of testing resources by monitoring workload distribution, prioritizing testing tasks, and ensuring that key areas (e.g., critical functionality, security) are tested thoroughly.
7. **Quality Assurance:** Guarantee that the system not only meets functional requirements but also adheres to quality standards, ensuring the software is reliable, secure, and user-friendly.

1.2.2 Activities

Monitoring

1. **Test Case Execution:** Track progress by comparing the number of executed test cases to the planned total. This allows for early detection of delays and ensures that testing milestones are being met.
2. **Defect Trends:** Monitor defect density, severity levels, and recurring issues over time. Analyzing trends helps identify systemic problems, areas requiring further attention, and potential risks to the quality of the system.
3. **Schedule Adherence:** Continuously monitor the testing schedule to ensure all milestones are achieved on time. This includes identifying any deviations from the original timeline and proactively addressing causes to prevent delays.

Reporting

1. **Comprehensive Reports:** Provide clear, concise, and actionable status reports that outline test progress, current risks, defect trends, and key metrics such as pass/fail rates, test coverage, and defect resolution. These reports should be tailored to different stakeholders for better understanding and decision-making.
2. **Real-Time Dashboards:** Utilize real-time dashboards to visualize progress and current test results, allowing stakeholders to access up-to-date information at any moment. Dashboards can include metrics on test case execution, defect status, and resource utilization.

Issue Management

1. **Conflict Resolution:** Escalate bottlenecks such as delays in defect fixes or resource allocation. Ensure timely intervention and resolution to prevent issues from negatively affecting the testing process.
2. **Resource Allocation:** Reassign resources as needed to address bottlenecks and ensure that critical testing tasks (e.g., performance testing, security testing) are not delayed due to lack of personnel or expertise.
3. **Escalation Protocols:** Define clear escalation paths for issues that cannot be resolved within a reasonable time frame, ensuring that higher-level management is involved when necessary to prioritize or allocate additional resources.

Change Management

1. **Dynamic Adaptation:** Revise test plans or adapt the testing scope in response to requirement changes, particularly for areas like third-party API integrations (e.g., Stripe API). Ensure that the test cases remain relevant and aligned with the updated requirements.
2. **Impact Assessment:** Evaluate the potential impact of any changes to requirements, and adjust the testing strategy accordingly to address new risks or functionalities that may require additional testing or different approaches.

Defect Tracking

1. **Tracking Tools:** Use defect tracking tools like Jira to efficiently manage the defect lifecycle. Track each defect's status, assign severity levels, and ensure timely resolution.
2. **Priority Assignment:** Assign defects based on their severity, impact, and priority, and monitor their resolution closely. Focus on critical defects that affect core functionality or security, ensuring these are addressed first.
3. **Root Cause Analysis:** Conduct root cause analysis for recurring defects or major issues to prevent them from reoccurring in the future. Use these findings to refine test cases and improve overall product quality.
4. **Defect Reporting Metrics:** Track metrics such as defect density, defect resolution time, and defect recurrence to evaluate the efficiency of the testing process and defect management efforts.

1.2.3 Tools for Test Control

1. **Test Management Tools:**
    - **Jira or TestRail:** Manage test cases, track execution progress, and log defects. These tools provide a centralized platform for organizing testing activities and allow for detailed reporting on test coverage and defect status.
2. **Defect Tracking Tools:**
    - **Bugzilla or Jira:** Track and prioritize defects with detailed reporting on severity, resolution time, and defect trends. These tools enable clear communication between testers and developers, ensuring timely fixes.

3. **Version Control Tools:**
    - **GitLab or GitHub:** Track changes to test scripts and ensure version control for test artifacts. These platforms support collaboration and facilitate the integration of test scripts into continuous integration pipelines.
4. **Continuous Integration Tools:**
    - **Jenkins or CircleCI:** Automate the execution of tests with each code change, ensuring immediate feedback on the system's functionality and performance.
5. **Collaboration Tools:**
    - **Slack or Microsoft Teams:** Use these tools for real-time communication, sharing updates, and collaborating on defect resolution. These platforms foster quick decision-making and immediate attention to issues.
6. **Monitoring and Reporting Tools:**
    - **SonarQube or New Relic:** Monitor system performance and detect any anomalies, ensuring that the system meets quality standards throughout the testing phase. These tools also provide insights into code quality, helping identify potential areas for improvement.

1.2.4 Deliverables

1. **Test Metrics Reports:**
    - **Analysis of Key Metrics:** Provide quantitative clarity on test coverage, execution status, and defect statistics. Include metrics like test pass/fail rate, defect density, test execution progress, and defect resolution time. Use these metrics to assess testing effectiveness and system quality.
    - **Performance Metrics:** Include key performance indicators (KPIs) such as response time, load handling, and scalability during testing phases to ensure the system meets non-functional requirements.
2. **Final Test Report:**
    - **Holistic Summary:** Summarize the overall outcomes, challenges faced during testing, and any unresolved issues. Highlight key findings, such as critical defects or performance bottlenecks, and provide a recommendation for deployment readiness based on test results.
    - **Risk Assessment:** Provide an analysis of any remaining risks that could impact deployment or user experience. This should include unresolved defects, performance concerns, or integration issues that need further attention before the system can be fully released.
    - **Actionable Recommendations:** Offer recommendations for post-deployment monitoring, including areas that may require additional focus or testing post-launch. This could include suggested improvements or ongoing testing cycles to ensure system stability.

2. Test Analysis and Design

2.1 Test Analysis

2.1.1 Objectives

- Extract exact test conditions and scenarios that are coming form formally written project requirements.
- Link every test condition to a single requirement for full requirement coverage.
- Review Stripe API and integration docs to see if we are missing any edge cases.

2.1.2 Activities

Requirement Analysis

Analyze requirements from sources like user stories, technical documentation, and API guides.

Categorize tests into functional, security, and non-functional aspects such as scalability and performance.

Test Basis Identification

Use key project artifacts:

System design documents (registration and payment workflows).

Use cases and personas (e.g., new student registrations with payment).

Risk assessment reports (e.g., API integration failures).

Deriving Test Conditions

Define high-level scenarios such as:

Validation of the student registration process (valid/invalid inputs).

Payment handling including transaction failure cases.

Prioritization

Assign testing priorities (high, medium, low) depending on business impact and risk levels

2.2 Test Design

2.2.1 Objectives

2.2.2 Activities

2.2.3 Deliverables

2.2.4 Tools for Analysis and Design