Verification and Validation Report: MES-ERP

Team #26, Ethical Pals
Sufyan Motala
Rachid Khneisser
Housam Alamour
Omar Muhammad
Taaha Atif

March 10, 2025

1 Revision History

Date	Version	Notes
March 10	1.0	Sufyan added Comparison to Existing Implementation and Automated Testing
March 10	1.1	Housam added Functional Requirements Evaluation and Trace to Modules
March 10	1.2	Taaha added Nonfunctional Requirements Evaluation and Trace to Requirements
March 10	1.3	Rachid Added Unit Testing and Changes Due to Testing
March 10	1.3	Omar added Symbols, Abbreviations and Acronyms and Code Coverage Metrics

2 Symbols, Abbreviations, and Acronyms

Abbreviation	**Description**
MES	McMaster Engineering Society
V&V	Verification and Validation
SRS	Software Requirements Specification
UI	User Interface
API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Deployment

This section defines the symbols, abbreviations, and acronyms used throughout the document to ensure clarity and consistency.

Contents

List of Tables

List of Figures

This document ...

3 Functional Requirements Evaluation

This section presents the detailed outcomes of functional testing for our six core functional tests (test-id1 through test-id6). We focus on (a) the inputs used, (b) observed vs. expected behavior, (c) issues discovered, and (d) improvements made to the code. All tests map directly to the requirements described in our SRS, and traceability is discussed in Section Trace To Modules.

3.1 Test 1: Submission Confirmation (test-id1)

Goal and Input: A user, already logged in, submits a valid reimbursement form *RB-2025-0002*. For instance, the form includes:

- **Expense Name:** Office Supplies
- **Date:** 2025-02-10
- **Amount:** $120.00 USD
- **Attached Receipt:** `receipt-office-supplies.pdf`

Expected Outcome: The system displays a "Submission Successful" message and updates the ledger with a corresponding entry.
Results and Analysis: We tested ten separate reimbursement submissions, each referencing distinct receipts. In nine submissions, the ledger reflected the new transaction instantly and displayed a clear confirmation message. However, one submission with a missing `budgetLine` field produced a generic error message, indicating partial fulfillment of the requirement but lacking robust error handling.
Code Improvements:

- Added front-end checks (React form validation) preventing the user from submitting the form if required fields (like `budgetLine`) are empty.
- Updated the `SubmitController` to return a descriptive error message ("Missing budget line") instead of the vague "400 Bad Request."

These changes improved usability by alerting users to incomplete inputs before reaching the back end, better aligning the submission workflow with SRS clarity requirements.

3.2 Test 2: Approval Workflow (test-id2)

Goal and Input: An administrator views a pending reimbursement request in the system (example ID: *RB-2025-0003*, "Travel Reimbursement," $200.00). **Expected Outcome:** Upon clicking "Approve," the request status changes to `Approved` and the ledger is updated; an email notification is sent to the user.
Results and Analysis: During testing, the system correctly updated the reimbursement status and updated the ledger with "Approved." However, cases where the submitter's email address was missing or misformatted led to silent notification failures, leaving the user uninformed.
Code Improvements:

- Implemented an email format validator within a new `NotificationService`.
- Logged invalid email attempts so administrators could correct user contact data.
- Refined multi-level approval for requests over $500, satisfying the SRS's requirement for stricter governance on higher-value transactions.

This ensures that all users now receive a reliable status update, and administrators have a systematic way to track any failed email notifications.

3.3 Test 3: Expense Addition (test-id3)

Goal and Input: Starting with a budget that has zero transactions, a user adds a new expense (**Expense Name: "Software License"**, **Amount: $50.00**, **Date: 2025-03-01**). **Expected Outcome:** The "Budget Overview" section in the UI should update in real time, reflecting this added expense in the total used amount.
Results and Analysis: On modern hardware, the Budget Overview updated within one second. On older machines, the update occasionally took two to three seconds, suggesting a minor performance bottleneck.
Code Improvements:

- Created a database index on `expenses.date` and `clubID` to speed up queries.
- Moved certain calculations from the front-end to a lightweight server endpoint, reducing rendering times on less powerful devices.

These optimizations brought the update time on older laptops down to about one to two seconds, meeting our real-time budget tracking goals stated in the SRS.

3.4 Test 4: Budget Creation and Categorization (test-id4)

Goal and Input: A user with appropriate permissions creates a new budget with a **Category: "Equipment"** and a **Cap: $500.00**. **Expected Outcome:** The system confirms creation of a "Equipment" budget category and displays it correctly in the dashboard.
Results and Analysis: Budget creation worked as expected, but inconsistent capitalization in the category field ("equipment," "EQUIPMENT," etc.) created multiple labels in the UI.
Code Improvements:

- Modified `BudgetManager` to normalize category inputs to lowercase, preventing duplicates.
- Added auto-suggest to the front end for known categories to further reduce user confusion.

These fixes ensure the final system meets the SRS requirement for coherent budget organization and accurate categorization.

3.5 Test 5: Notification on Reimbursement Status Change (test-id5)

Goal and Input: An admin changes the status of a reimbursement request (e.g., *RB-2025-0004* "Conference Fee") from `pending` to `approved`. **Expected Outcome:** The user's dashboard reflects the updated status immediately, and an email notification is sent confirming approval.

Results and Analysis: All tested requests updated the user interface within one second of the admin's action, and email notifications were consistently sent without error. Thus, the system performed as expected with no disruptions or confusion.
Code Improvements:

- Added an optional delivery-time log to the `NotificationService`, helping us monitor any future latency spikes.
- Maintained a uniform approval message template, aligning with the SRS's requirement for consistent user communication.

3.6 Test 6: Audit Trail Logging (test-id6)

Goal and Input: Starting with an idle system, we performed a sequence of actions—budget creation, expense submission, and approval—to check the audit trail's completeness. **Expected Outcome:** Each action should be recorded with the correct user ID, timestamp, and request reference number for compliance tracking.
Results and Analysis: Every action was indeed captured (*e.g.*, "User `u1002` created Budget ID `BG-0003` at 2025-03-02T14:05Z"), but the timestamps initially used local server time (EST), confusing for staff in other time zones.
Code Improvements:

- Standardized timestamps to UTC in `AuditLog`.
- Extended the admin panel with a time-zone converter, ensuring a clear and unified audit trail per SRS compliance requirements.

3.7 Overall Observations and Next Steps

Across Tests 1–6, our system demonstrated strong alignment with the SRS specifications. We uncovered minor usability gaps (form validations, inconsistent category naming) and performance issues on older hardware, each resolved through targeted code improvements. Moving forward, we will:

- Conduct additional stress tests for budgets with a high number of expenses to ensure sustained performance.

- Evaluate the updated `NotificationService` more extensively with user acceptance testing to confirm email reliability.
- Gather stakeholder feedback on the new category auto-suggest to ensure it fosters clarity without cluttering the UI.

These refinements maintain consistency with our planned VnV approach while *not* duplicating the procedures from the plan. We will re-run these tests after each major update to verify no regressions occur and that new features continue to meet the functional requirements set forth in the SRS.

4 Nonfunctional Requirements Evaluation

This section summarizes the results of our nonfunctional requirements testing (Tests 7–10). As before, we do not restate each test procedure from the VnV Plan, but rather highlight the observed outcomes, any issues discovered, and the improvements made to address them.

4.1 Usability

Overview: We conducted informal user trials where new users (e.g., club leaders unfamiliar with the system) performed basic tasks such as budget creation and reimbursement submission. Our goal was to gauge ease of navigation, clarity of interface elements, and overall user satisfaction.
Findings and Improvements:

- **Clarity of Labels:** Some users found certain labels ambiguous (e.g., "Add Reimbursement" vs. "Submit Request"). We updated form labels to match simpler, more intuitive phrasing ("Submit Reimbursement").
- **Tooltip Guidance:** We added short tooltips over key fields (like "Budget Line") to reduce confusion, especially for first-time users.
- **Success Indicators:** After an action, a quick "Success!" toast message is now displayed in the corner of the screen, reinforcing the system's responsiveness and user confidence.

Overall, feedback was positive, indicating that the interface was straightforward to learn and navigate. Further structured usability testing is planned with a larger group of student leaders in future sprints.

4.2 Performance (Tests 7 & 9)

4.2.1 Test 7: Response Time Under Load (test-id7)

Goal and Input: We simulated multiple users (15 concurrent) simultaneously submitting reimbursement requests. **Expected Outcome:** The system should respond within our target threshold (under 3 seconds for form submission) despite heavy load.
Results and Analysis:

- Average response time was about 2.2 s under 15 concurrent submissions, which meets our 3 s goal.
- A small fraction (about 8%) of requests took slightly longer (3.5 s) when older machines joined the test.

Code Improvements:

- We added an in-memory cache for frequently accessed club/budget data, reducing repeated database lookups.
- Improved concurrency handling in the our reimbursement controller, ensuring that read-write locks do not block each other unnecessarily.

These optimizations helped keep the average response times near 2 seconds, even under moderate stress, aligning with our performance requirements from the SRS.

4.2.2 Test 9: Load Handling (test-id9)

Goal and Input: We manually simulated multiple users filling out various forms (expense addition, budget creation) simultaneously. **Expected Outcome:** The system should maintain functional integrity and acceptable performance (i.e., not degrade significantly under up to 20 concurrent sessions).
Results and Analysis: During a 20-user concurrency spike:

- CPU usage on the hosting server peaked at about 70%.
- Response times remained under 4 seconds for all but one outlier, which took 5 seconds due to a large image upload.

- No crashes or data inconsistencies occurred.

Code Improvements:

- We introduced a file-size check for receipt uploads to warn users if the attachment is too large, preventing slowdowns from oversize images.
- Split the monolithic `AppService` into smaller modules, allowing better scaling in future container-based deployments.

These measures help ensure that the system scales gracefully under typical usage levels expected by the MES.

4.3 Security and Access Control (Test 8)

Goal and Input: An unauthorized user attempts to access restricted sections (e.g., admin panel or high-value approval screens) using guessed URLs or direct links. **Expected Outcome:** The user is denied access and any unauthorized attempt is logged.
Results and Analysis:

- All restricted endpoints properly returned `403 Forbidden` or redirected to the login page.
- Attempts were logged under the "Security Log" with the user's IP address, time, and requested endpoint.

Code Improvements:

- Added an optional "Lockout" mechanism—if an IP attempts 5 consecutive invalid logins or access attempts within 10 minutes, that IP is temporarily blocked.
- Clarified error messages to avoid giving hints about admin URL structures or user roles.

These changes reinforce alignment with the SRS's confidentiality and integrity requirements, mitigating unauthorized access attempts.

4.4 Data Integrity and Security (Test 10)

Goal and Input: Under normal usage with data integrity checks enabled, multiple transactions and database updates are performed across different user sessions. **Expected Outcome:** The data remains consistent, properly linked to the correct user and budget lines, and no unauthorized modifications go undetected.
Results and Analysis:

- We performed 50 consecutive transactions, including expense additions, reimbursements, and budget edits. All fields remained consistent, with no partial or duplicated entries.
- The system's hash-based verification on critical fields (e.g., amounts) detected one deliberate attempt at direct DB manipulation, logging it and rolling back changes.

Code Improvements:

- Strengthened the DB layer to disallow direct updates to amount fields without valid session tokens.
- Enhanced the `DataValidator` class to maintain an immutable transaction ID log, preventing silent tampering of posted reimbursements.

Hence, the SRS's requirement for secure and tamper-evident records is fulfilled, ensuring high confidence in the system's data integrity.

4.5 Conclusion on Nonfunctional Testing

Tests 7 through 10 confirm that our system addresses key nonfunctional aspects:

- *Usability:* While not a specific test ID, our informal checks indicated a straightforward user experience, supported by better labeling and tooltips.
- *Performance:* Simultaneous submissions and multiple concurrent sessions did not significantly degrade response times, thanks to caching, concurrency tuning, and partial modularization.

- *Security and Access Control:* Unauthorized attempts were fully denied and logged, meeting the SRS requirement for robust access controls.
- *Data Integrity:* Hash-based verifications and stronger DB rules ensured that transaction data remained correct and tamper-evident.

Future work includes more aggressive load testing with greater concurrency to reflect final deployment usage and a formal usability study with a broader user base. Overall, the system's nonfunctional performance and security posture align well with SRS mandates, ensuring reliability for the McMaster Engineering Society's financial management needs.

5 Comparison to Existing Implementation

The MES-ERP system represents a significant advancement over the previous implementation, which relied on Google Forms for expense submissions and spreadsheets for budget tracking and reimbursement processing. This section compares the two implementations across several key dimensions.

5.1 Process Efficiency

- **Previous Implementation:** The Google Forms and spreadsheets approach required manual data transfer between systems. Staff had to copy information from form submissions into tracking spreadsheets, manually update status changes, and communicate updates via separate email threads.
- **MES-ERP:** The new system provides an end-to-end digital workflow where data flows automatically through the system. Once a request is submitted, it remains in the database and can be tracked, updated, and processed without manual data re-entry.

5.2 User Experience

- **Previous Implementation:** Users had to navigate to different Google Forms for different types of requests. After submission, they had limited visibility into the status of their requests and often had to follow up via email.

- **MES-ERP:** Users now have a centralized dashboard where they can submit different types of requests, track the status of all their submissions, and receive automatic email notifications when their request status changes.

5.3 Role-Based Access Control

- **Previous Implementation:** Access control was limited to basic Google permissions. Different spreadsheets were shared with different stakeholders, creating information silos and making it difficult to maintain a comprehensive view of financial activities.

- **MES-ERP:** The system implements a sophisticated role-based access control system that allows for fine-grained permission management. Users can be assigned specific roles (e.g., club admin, MES executive) with appropriate permissions, ensuring they can access only the information and functions relevant to their responsibilities.

5.4 Budget Management

- **Previous Implementation:** Budget tracking was done in separate spreadsheets, making it difficult to get real-time insights into spending patterns or remaining budgets. Annual budget planning was a separate process with limited integration to actual spending.

- **MES-ERP:** The system integrates budget planning and expense tracking in a single platform. The operating budget module allows for detailed budget planning, while the analytics dashboard provides real-time visibility into spending patterns, budget utilization, and financial trends.

5.5 Notification System

- **Previous Implementation:** Status updates and approvals were communicated manually via email, leading to delays and sometimes missed communications.

- **MES-ERP:** The system includes an automated notification system that sends emails to users when their request status changes, providing clear information about the current status and expected next steps.

5.6 Group Management

- **Previous Implementation:** Group affiliations were tracked manually in spreadsheets, making it difficult to enforce group-specific budget limits or permissions.
- **MES-ERP:** The system includes dedicated group management functionality, allowing administrators to create and manage groups, assign users to groups, and enforce group-specific budget allocations and permissions.

5.7 Analytics and Reporting

- **Previous Implementation:** Generating reports or analytics required manual data extraction and processing from multiple spreadsheets.
- **MES-ERP:** The system includes a built-in analytics dashboard that provides visualizations of spending patterns, budget utilization, request volumes, and other key metrics, enabling data-driven decision making.

The MES-ERP system has successfully addressed the key limitations of the previous implementation, providing a more efficient, transparent, and user-friendly platform for managing the MES financial processes. The integration of reimbursement requests, budget planning, user role management, and automated notifications into a single system has significantly improved the overall experience for both users submitting requests and administrators managing the financial processes.

6 Unit Testing

Unit testing was conducted using a combination of automated and manual testing to ensure key functionalities worked as expected. The primary objectives were:

- Verify that critical features, such as reimbursement submissions and approvals, group creation, and user management work correctly.
- Identify and resolve potential errors early in the development cycle.
- Ensure that the system remains stable after modifications.

6.1 Testing Approach

The team used **Jest** for automated testing of frontend logic and API calls, and **manual testing** for real-world scenario validation. The approach included:

- Writing unit tests for key functions such as group creation, deletion, and authentication handling.
- Conducting manual test runs where team members acted as users submitting and approving reimbursement requests.
- Gathering feedback from a small group of MES student leaders who tested the platform and provided insights.
- Running scenario-based testing sessions to simulate real-world usage, such as handling multiple reimbursement requests at once.

6.2 Unit Test Cases

The test cases covered core functionalities, including validation checks and API interactions:

Table 1: Sample Unit Test Cases

Test Case	**Expected Output**
Create a group with a unique name	Group is successfully created
Attempt to create a group with an existing name	System displays an error message
Unauthorized user attempts to access group management	Access is denied with error message

Continued on next page

Table 1 – *Continued from previous page*

Test Case	**Expected Output**
Delete a group with no assigned users	Group is successfully deleted
Attempt to delete a group with assigned users	System prevents deletion and notifies the user
Handle failed database connection during group fetch	Error is logged, and fallback UI is displayed
Register a new user with valid credentials	User is successfully created and logged in
Attempt to register with an existing email	System displays an error message
Login with correct credentials	User is authenticated and redirected to the dashboard
Login with incorrect credentials	System displays an error message
Assign a role to a user	User is successfully assigned the role
Remove a role from a user	User no longer has the role permissions
Attempt to assign an invalid role	System displays an error message
Submit a payment request with valid details	Request is successfully submitted
Submit a payment request missing required fields	System displays an error message
View all payment requests as an admin	Admin sees all requests
View only personal payment requests as a user	User sees only their own requests
Attempt to approve a request without proper permissions	Access is denied with an error message
Update request status successfully	Request status is updated

Continued on next page

Table 1 – *Continued from previous page*

Test Case	**Expected Output**
Handle failed database connection when fetching requests	Error is logged, and fallback UI is displayed

6.3 Edge Cases Tested

- Submitting a reimbursement request without required fields.
- Uploading incorrect file formats for receipts.
- Attempting to create a duplicate group.
- Handling multiple users modifying the same data simultaneously.
- Ensuring database rollback occurs when a transaction fails.
- Registering a user with an invalid email format.
- Assigning multiple roles to a user and verifying access control.
- Submitting a payment request with an incorrect amount format.
- Preventing unauthorized users from updating request statuses.
- Ensuring bulk request approvals work correctly.

7 Changes Due to Testing

Based on the testing process and feedback from users, the following modifications were made to the system:

7.1 Bug Fixes

- Fixed an issue where reimbursements submitted without attachments were still being processed.
- Addressed a bug where users could submit duplicate requests.
- Resolved a login issue where incorrect error messages were displayed.

- Fixed a bug where request status updates were not properly reflected in the UI.
- Added the ability for users to be in multiple groups.

7.2 User Feedback and Enhancements

- Improved the **receipt upload system** by making the file upload process clearer.
- Enhanced **audit logging** for better tracking of request status changes.
- Added a clearer role management interface to reduce confusion when assigning roles.
- Improved user experience by adding filters to search for specific payment requests.

7.3 Performance and Security Improvements

- Implemented basic input validation to prevent incorrect data submissions.
- Reduced system load times by optimizing database queries.
- Strengthened authentication by enforcing stricter password requirements.
- Enhanced role-based access control to prevent unauthorized modifications.

8 Automated Testing

9 Trace to Requirements

10 Trace to Modules

This section maps each requirement and test (as defined in the SRS, MG, and VnV Plan) to the specific modules that implement or support it. In doing

so, we confirm that our MES-ERP design provides complete coverage of all requirements and that each module's responsibilities are clearly delineated.

10.1 Requirement-to-Module Mapping

Functional Requirements:

- **Reimbursement Submission (FR1):**
  - *Expense Submission & Tracking Module* – Handles form submissions, attachments, and ledger updates.
  - *Database Module* – Persists the reimbursement records.
  - *GUI Module* – Presents the submission form and success/failure messages.
- **Approval Workflow (FR2):**
  - *Approval Workflow and Review Module* – Implements the routing and multi-level approval rules.
  - *Notifications & Communication Module* – Sends email notifications to requestors when status changes.
- **Expense Addition (FR3):**
  - *Expense Submission & Tracking Module* – Allows users to add line items and re-check the budget.
  - *Budget & Funding Management Module* – Validates funds and updates relevant budget lines.
- **Budget Creation & Categorization (FR4):**
  - *Budget & Funding Management Module* – Creates new budgets, enforces budget caps, and supports category definitions.
  - *GUI Module* – Provides the interface for authorized users (e.g., admins) to create or edit budget lines.
- **Notification on Reimbursement Status Change (FR5):**

  - *Notifications & Communication Module* – Checks the relevant roles and sends out email notifications when a request's status is updated.
  - *Approval Workflow and Review Module* – Triggers the sending of status updates.

- **Audit Trail Logging (FR6):**
  - *Policy & Compliance Management Module* – Logs activity to ensure compliance with MES rules.
  - *Database Module* – Stores the record of every action, including time stamps and user IDs.

Nonfunctional Requirements:

- **Performance Under Load (NFR: Performance, test-id7 & test-id9):**
  - *Database Module* – Indexed queries, concurrency handling.
  - *Expense Submission & Tracking Module* – Receives multiple simultaneous requests.
  - *Notifications & Communication Module* – Queues or throttles outgoing messages to avoid timeouts.

- **Security and Access Control (NFR: Security, test-id8):**
  - *User Authentication & Profile Management Module* – Validates user credentials, manages sessions.
  - *Approval Workflow and Review Module* – Enforces role-based checks for high-value approvals.
  - *Database Module* – Restricts direct data access, logs unauthorized attempts.

- **Data Integrity and Security (NFR: Data Integrity, test-id10):**
  - *Database Module* – Transaction atomicity and rollbacks.
  - *Policy & Compliance Management Module* – Ensures requests conform to MES rules, logs attempted tampering.

- **Usability (NFR: Usability):**
  - *GUI Module* – Provides clarity with tooltips, placeholders, and "Success!" confirmations.
  - *Notifications & Communication Module* – Furnishes immediate feedback to keep users informed.

10.2 Test-to-Module Mapping

Functional Tests (IDs 1–6):

- **Test-id1 (Submission Confirmation):**
  - *Expense Submission & Tracking Module* – Receives and stores new submissions.
  - *Database Module* – Persists ledger updates for newly submitted expenses.
  - *GUI Module* – Displays success/failure to the end user.
- **Test-id2 (Approval Workflow):**
  - *Approval Workflow and Review Module* – Executes the approval logic.
  - *Notifications & Communication Module* – Notifies the requestor upon approval.
  - *Database Module* – Records the changed status in persistent storage.
- **Test-id3 (Expense Addition):**
  - *Expense Submission & Tracking Module* – Adds new line items to an existing request.
  - *Budget & Funding Management Module* – Checks that the updated total does not exceed the available budget.
- **Test-id4 (Budget Creation & Categorization):**
  - *Budget & Funding Management Module* – Generates new budgets and enforces budget category constraints.

  - *GUI Module* – Allows an authorized user to set or adjust category limits.

- **Test-id5 (Notification on Status Change):**
  - *Approval Workflow and Review Module* – Initiates the state change event for a request.
  - *Notifications & Communication Module* – Dispatches email alerts to the user once the request is approved/denied.

- **Test-id6 (Audit Trail Logging):**
  - *Policy & Compliance Management Module* – Writes a log entry for each update or action.
  - *Database Module* – Stores audit records with timestamps and user identifiers.

Nonfunctional Tests (IDs 7–10):

- **Test-id7 (Response Time Under Load):**
  - *Expense Submission & Tracking Module* – Responsible for receiving the bulk of the requests during the load test.
  - *Database Module* – Must handle a high concurrency of inserts/updates.
  - *GUI Module* – Observes if response times degrade from the user perspective.

- **Test-id8 (Security and Access Control):**
  - *User Authentication & Profile Management Module* – Verifies session validity, enforces lockouts on repeated failures.
  - *Approval Workflow and Review Module* – Checks roles and denies access if the user lacks privileges.

- **Test-id9 (Load Handling):**
  - *Expense Submission & Tracking Module* – Supports multiple concurrent user sessions.
  - *Notifications & Communication Module* – Sends a burst of email messages, verifying no backlog or failures.

  - *Database Module* – Verifies indexing and concurrency are robust under near-peak loads.

- **Test-id10 (Data Integrity and Security):**
  - *Policy & Compliance Management Module* – Ensures no tampering of amounts or request status changes can happen without being logged.
  - *Database Module* – Maintains transaction integrity (e.g., rollbacks for partial failures), ensures data is unaltered without authorization.

10.3 Conclusion

Through the above *Trace to Modules*, we verify that:

- Every functional requirement (FR1–FR6) and nonfunctional requirement (performance, security, data integrity, usability) is addressed by one or more modules.
- Every test (IDs 1–10) directly maps to the relevant module(s) that implement or facilitate that functionality.
- We have no orphan requirements (i.e., every requirement is accounted for) and no orphan modules (i.e., each module fulfills an SRS-defined purpose).

This clear mapping minimizes duplication, ensures maintainability, and provides stakeholders and developers confidence that the MES-ERP system's architecture adheres to the specifications laid out in the SRS, MG, and VnV Plan.

11 Code Coverage Metrics

To ensure the robustness and reliability of the MES-ERP system, we will use a combination of code coverage metrics to evaluate the effectiveness of our testing. The goal of code coverage analysis is to measure the extent to which the source code is tested, helping identify untested paths and potential vulnerabilities.

11.1 Code Coverage Measurement Approach

The following strategies were used to assess the coverage of our codebase, they were implemented using the tests listed above in the document or if neccessary, manually:

- **Branch Coverage**: Ensures that both the expected inputs and the expected alerts for wrong inputs are executed (if/else).
- **Function Coverage**: Confirms that all functions and methods in the system have been called at least once.
- **Line Coverage**: Measures the percentage of total lines of code executed during testing.
- **File Coverage**: Ensures that all files and folders in the project are utilized somewhere in the website.

11.2 Coverage Results

The overall code coverage percentage or verification was determined at an approximate number from our tests. Additionally, we verified that all files and folders used in the project are correctly accessed within the website:

Category	**Coverage**
All Files Accessed	Yes
All Folders Utilized	Yes
Expected Inputs and Alerts Tested	Yes
All Functions utilized	Yes
Approximate Percent of Code Covered	50%

Table 2: File and Folder Coverage Verification

11.3 Test Case Effectiveness

Although the test cases listed above cover only approximately 50% of the total site code, they accurately represent the entire system by focusing on:

- Core functionalities such as authentication, reimbursement submissions, and approvals.
- Important workflows involving multiple user roles (students, admins, and finance officers).
- Edge cases, including incorrect data entries, invalid file uploads, and simultaneous database transactions.

Since our coverage strategy prioritizes high-impact areas, we are confident that the test suite provides a strong measure of overall system reliability. Future improvements will focus on expanding coverage for lesser-used components and non-critical paths.

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Reflection.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?
   Omar: What went well while writing this deliverable was the structured approach we took from the beginning. By clearly defining our Verification and Validation (VnV) strategy early on, we were able to efficiently organize the document and ensure all necessary components were included.

2. What pain points did you experience during this deliverable, and how did you resolve them?
   Omar: One of the main pain points I experienced during this deliverable was ensuring that the VnV plan aligned with our actual testing process and our original VnV plan. To accomodate for this I just went back and forth between documents and talked to my team to ensure I did not mix anything up.

3. Which parts of this document stemmed from speaking to your client(s) or a proxy (e.g. your peers)? Which ones were not, and why?
   Team: The majority of this document stemmed from discussions with my peers, as we collectively determined the best approach for verification and validation. Specific sections, such as requirements and changes due to testing all stem from our stakeholders. We built upon what we

know from our stakeholders for those sections. Other sections were built upon feedback received from stakeholders as well as the TA/Professor, we made sure to account for all information provided to us to ensure a complete report.

4. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)
   Team: The actual VnV activities differed slightly from the original plan, primarily due to adjustments made during the development cycle. Some test cases had to be modified or expanded as we encountered new scenarios that were not initially anticipated, such as specific security vulnerabilities and data integrity concerns. The biggest change was in prioritizing additional integration tests over certain lower-priority unit tests, since we found that issues were more likely to arise in cross-module interactions. These changes occurred because real-world implementation often reveals gaps in planning, and some verification methods turned out to be less effective than expected. In future projects, I would anticipate these changes by leaving more flexibility in the VnV plan and incorporating iterative updates based on early testing feedback.